VVRoom Textbook

Building a URL-First Angular Application

A Comprehensive Technical Guide

Table of Contents

Primers

Services

Models

Adapters

Domain Config

Chart Sources

Components

Pages

Reference

Appendix

000: Book Conventions

Document Type: Meta **Purpose:** Establish the style and structure for all textbook documents in this series

---

Overview

This `textbook/` directory contains the manuscript for a comprehensive Angular 13 programming book. The book follows the style of Apress technical publications, with a focus on learning-by-doing through the incremental construction of a real application: **vvroom**, an automobile discovery platform.

The application implements the **URL-First State Management** pattern, documented in `docs/`.

Target Audience

A junior developer who will:

- Type every line of code shown
- Understand why each decision was made
- Have a working, testable application at every checkpoint

Document Structure

Each numbered document (`001-*.md`, `002-*.md`, etc.) represents a chapter or major section. Every document follows this structure:

1. Header Block

```
# NNN: Title

**Status:** Planning | In Progress | Complete **Depends On:** List of prerequisite document
numbers **Blocks:** List of documents that depend on this one
```

2. Objective

A single paragraph stating what this section accomplishes.

3. Why (Rationale)

- Explain the reasoning behind the approach
- Reference industry standards:
- Angular Style Guide (https://angular.io/guide/styleguide)
- URL-First pattern (see `docs/README.md`)
- TypeScript best practices
- RxJS patterns
- Cite specific rules when applicable (e.g., "Style 04-10: Use redirects for default routes")

4. What (Implementation)

Step-by-step instructions with:

- **Full absolute paths** for every file (e.g., `src/app/features/home/home.component.ts`)
- **Complete file contents** — no ellipses, no "add similar code"
- **Every import statement** — never assume the reader knows what to import
- **Terminal commands** with expected output where applicable

5. Verification

- How to confirm the step worked
- What to see in the browser

- What to see in the console
- Any tests to run

6. Acceptance Criteria

Checkbox list of requirements that must be met before proceeding.

7. Next Step

Pointer to the next document in the sequence.

File Path Conventions

Always use paths relative to the project root (`~/projects/vvroom/`):

```
src/app/app.component.ts           # Root component

src/app/app.routes.ts              # Route definitions src/app/features/
home/             # Feature module directory src/app/features/discover/         #
Feature module directory src/app/framework/services/        # Shared services src/app/
framework/models/          # TypeScript interfaces
```

When referencing files in prose, use backticks and the full path:

- Correct: "Open `src/app/app.component.ts` "
- Incorrect: "Open the app component"

Code Block Conventions

New Files

When creating a new file, show the complete contents:

```
// src/app/features/home/home.component.ts

import { Component } from '@angular/core';



@Component({ selector: 'app-home', template: <h1>Home</h1> }) export class
HomeComponent {}
```

Modifications to Existing Files

When modifying an existing file, show:

- The file path
- What to find (the existing code)
- What to replace it with (the new code)

```
// src/app/app.routes.ts

// REPLACE this: export const routes: Routes = [];



// WITH this: export const routes: Routes = [ { path: '', redirectTo: 'home',
pathMatch: 'full' }, { path: 'home', component: HomeComponent } ];
```

For additions, specify where to add the code:

```
// src/app/app.routes.ts

// ADD after the existing imports: import { HomeComponent } from './features/home/
home.component';
```

Terminal Command Conventions

Show commands with their expected output:

```
$ ng serve --port 4228 --open

✔ Browser application bundle generation complete.



Initial Chunk Files   | Names         |  Raw Size vendor.js             |
vendor        |   2.05 MB | polyfills.js          | polyfills     | 339.16 kB |
styles.css            | styles        |  95.54 kB | main.js               |
main          |  47.42 kB | runtime.js            | runtime       |   6.54 kB |



Build at: 2026-02-11T20:00:00.000Z - Hash: abc123 - Time: 5000ms



 Angular Live Development Server is listening on localhost:4228
```

Naming Schema

Documents are numbered by phase:

Number	Phase
000-049	Meta documents (conventions, rubric)
050-099	Phase 0: API Contract & Naming Conventions
100-149	Phase 1: Foundation
150-199	Interlude A: TypeScript Generics Primer
200-249	Phase 2: Framework Models
250-299	Interlude B: RxJS Patterns
300-399	Phase 3: Framework Services (3A: Core, 3B: Popout, 3C: Error Handling)
400-499	Phase 4: Domain Models
500-599	Phase 5: Domain Adapters
600-649	Phase 6: Domain Configs
650-699	Phase 7: Chart Data Sources
800-899	Phase 8: Framework Components
900-949	Phase 9: Feature Components
950-999	Appendices

Reference Implementation

The **generic-prime** project (branch `angular/13`) serves as the canonical reference for all code in this book:

Property	Value
Location	`~/projects/generic-prime`
Branch	`angular/13`

When working on chapters:

- Ensure generic-prime is on the `angular/13` branch

- Copy relevant code files to vvroom
- Adapt the textbook to accurately describe the copied code

To access the reference implementation:

```
cd ~/projects/generic-prime
git checkout angular/13
```

Reference Material

Resource	Location
URL-First Architecture	`docs/README.md`
Full Architecture Spec	`docs/ARCHITECTURE-OVERVIEW.md`
State Management Spec	`docs/STATE-MANAGEMENT-SPECIFICATION.md`
Pop-out Architecture	`docs/POPOUT-ARCHITECTURE.md`
Implementation Audit	`docs/URL-FIRST-AS-IMPLEMENTED.md`
Reference Implementation	`~/projects/generic-prime` @ `angular/13`

Principles

- **No magic** — Every line of code is explained or shown
- **Compiles at every step** — The application must build and run after each section
- **Why before what** — Rationale precedes implementation
- **Full paths always** — Never ambiguous file references
- **One concept per section** — Don't combine unrelated changes
- **Test what you build** — Verification steps after every implementation

---

URL-First Compliance

Every implementation step must adhere to the URL-First State Management paradigm:

```
User Action → URL Update → State Service → Components Re-render
```

See `instructions.md` for the complete URL-First Compliance Checklist.

This document establishes the contract between author and reader. All subsequent documents in `textbook/` will adhere to these conventions.

API Contract 051: API Contract Overview

051: API Contract Overview

Status: Complete **Depends On:** None **Blocks:** 052, 053, all implementation phases

---

Objective

Document the API contract that the vvroom application will consume. This includes base URL configuration, authentication requirements, pagination conventions, error formats, and general request/response patterns.

Why

Before writing any code — models, services, or components — you must understand the shape of the data you're working with. The API contract is an **input** to the entire system:

- **Models** are defined by what the API returns
- **Adapters** translate between API responses and models
- **Services** call endpoints defined by the API
- **Components** display data shaped by the API

Without this document, developers would be guessing at data shapes, leading to runtime errors and constant refactoring.

API Overview

Base URL

Environment	Base URL
Development	`http://generic-prime.minilab/api/specs/v1`
Production	`http://generic-prime.minilab/api/specs/v1`

The API is accessed via Traefik ingress on the Kubernetes cluster. The hostname `generic-prime.minilab` resolves to the cluster ingress controller.

Authentication

None required. The API is accessible without authentication for this internal application.

Content Type

All requests and responses use JSON:

```
Content-Type: application/json
Accept: application/json
```

Pagination Convention

All list endpoints support pagination with these query parameters:

Parameter	Type	Default	Description
`page`	number	1	Page number (1-indexed)
`size`	number	20	Items per page

Paginated Response Shape

```
{
"results": [...],      // Array of items for current page "total": 1234,         //
Total items across all pages "page": 1,             // Current page number "size":
20,            // Items per page "totalPages": 62       // Total number of pages }
```

Sorting Convention

Sorting is controlled by these query parameters:

Parameter	Type	Values	Description
`sortBy`	string	field name	Field to sort by
`sortOrder`	string	`asc`, `desc`	Sort direction

Example: `?sortBy=manufacturer&sortOrder=asc`

Error Response Format

When an error occurs, the API returns:

```
{
"success": false, "error": { "code": "VALIDATION_ERROR",     // Error type identifier
"message": "Invalid year range", // Human-readable message "details":
{                    // Optional additional context "field": "yearMin", "reason":
"must be less than yearMax" } } }
```

Common Error Codes

Code	HTTP Status	Description
`VALIDATION_ERROR`	400	Invalid request parameters
`NOT_FOUND`	404	Resource not found
`INTERNAL_ERROR`	500	Server error

Highlight Parameters (h_* prefix)

The API supports "highlight" parameters for segmented statistics. These are query parameters with an `h_` prefix that tell the backend to compute statistics with a highlighted subset.

Purpose: Enable stacked bar charts showing "highlighted vs total" data.

Example Request:

```
GET /vehicles/details?manufacturer=Toyota&h_yearMin=2020&h_yearMax=2024
```

Effect on Response: Statistics include `{total, highlighted}` objects instead of simple counts:

```
{
"statistics": { "byManufacturer": { "Toyota": { "total": 665, "highlighted": 234 },
"Honda": { "total": 849, "highlighted": 0 } } } }
```

Vehicles from Toyota are highlighted (234 of 665 match the year range 2020-2024), while Honda vehicles are not highlighted (0 of 849 match).

Request/Response Flow

Implementation Files

1. API Response Interface (Framework-Level)

This interface is **domain-agnostic** and lives in the framework:

File: `src/app/framework/models/api-response.interface.ts`

```
/**

  • Generic API response interface for paginated endpoints



  • @template TData - The type of data items in the results array

 */
export interface ApiResponse<TData> { /* Array of result items for the current page /
results: TData[];



/* Total number of items across all pages / total: number;



/* Current page number (1-indexed) / page: number;



/* Number of items per page / size: number;



/* Total number of pages / totalPages: number;



/* Optional statistics data (domain-specific) / statistics?: any; }



/**



  • Generic error response from API

 */
export interface ApiErrorResponse { /* Success flag (always false for errors) /
success: false;
```

```
/* Error details / error: { /* Error code (e.g., 'VALIDATION_ERROR', 'NOT_FOUND') /
code: string;



/* Human-readable error message / message: string;



/* Optional additional error details / details?: Record<string, any>; }; }



/**



  • Generic success response wrapper

 */
export interface ApiSuccessResponse<TData> { /* Success flag (always true for
successful responses) / success: true;



/* Response data / data: TData; }



/**



  • Standard API response type (success or error)

 */
export type StandardApiResponse<TData> = ApiSuccessResponse<TData> | ApiErrorResponse;
```

2. Environment Configuration

File: `src/environments/environment.ts` (development)

```
export const environment = {

production: false, apiBaseUrl: 'http://generic-prime.minilab/api/specs/v1',
includeTestIds: true };
```

File: `src/environments/environment.prod.ts` (production)

```
export const environment = {

production: true, apiBaseUrl: 'http://generic-prime.minilab/api/specs/v1',
includeTestIds: false };
```

Verification

After implementing this section:

- **Check file exists:**

```
ls -la src/app/framework/models/api-response.interface.ts
```

- **Check environment files:**

```
grep apiBaseUrl src/environments/environment*.ts
```

- **Verify TypeScript compiles:**

```
npx tsc --noEmit
```

Acceptance Criteria

- [x] `ApiResponse<TData>` interface created in `src/app/framework/models/`

- [x] `ApiErrorResponse` interface created
- [x] `ApiSuccessResponse<TData>` interface created
- [x] `StandardApiResponse<TData>` type alias created
- [x] `environment.ts` includes `apiBaseUrl` property
- [x] `environment.prod.ts` includes `apiBaseUrl` property

Next Step

Proceed to `052-automobile-endpoints.md` for detailed documentation of each automobile-specific endpoint with request/response examples.

052: Automobile Endpoints

Status: Reference **Depends On:** 051-api-contract-overview **Blocks:** 401-403 (Domain Models), 501-503 (Domain Adapters)

Objective

Document every API endpoint used by the automobile domain, including:

- Request URL and method
- Query parameters
- Complete response JSON shapes
- Example requests and responses

This document is the source of truth for building TypeScript models and adapters.

Endpoints Summary

Endpoint	Method	Purpose
`/vehicles/details`	GET	Search vehicles with pagination and statistics
`/statistics`	GET	Fetch statistics only (without vehicle data)

Endpoint: GET /vehicles/details

The primary endpoint for vehicle discovery. Returns paginated vehicle results with aggregated statistics.

Request

URL: `{baseUrl}/vehicles/details`

Query Parameters:

Parameter	Type	Required	Description
`manufacturer`	string	No	Filter by manufacturer name (partial match)
`model`	string	No	Filter by model name (partial match)
`yearMin`	number	No	Minimum year (inclusive)
`yearMax`	number	No	Maximum year (inclusive)
`bodyClass`	string	No	Filter by body class (can be comma-separated for multiple)
`instanceCountMin`	number	No	Minimum VIN instance count
`instanceCountMax`	number	No	Maximum VIN instance count
`search`	string	No	Global search across all fields
`models`	string	No	Model combinations: `Manufacturer:Model,Manufacturer:Model`
`page`	number	No	Page number (default: 1)
`size`	number	No	Items per page (default: 20)
`sortBy`	string	No	Sort field
`sortOrder`	string	No	Sort direction: `asc` or `desc`
`h_yearMin`	number	No	Highlight: minimum year
`h_yearMax`	number	No	Highlight: maximum year
`h_manufacturer`	string	No	Highlight: manufacturer
`h_modelCombos`	string	No	Highlight: model combinations
`h_bodyClass`	string	No	Highlight: body class

Example Request

```
GET /vehicles/details?
manufacturer=Toyota&yearMin=2020&yearMax=2024&page=1&size=20&sortBy=year&sortOrder=des
c
```

Response Shape

```
{
"results": VehicleResult[], "total": number, "page": number, "size": number,
"totalPages": number, "statistics": VehicleStatistics }
```

Example Response

```
{
"results": [ { "vehicle_id": "TOY-CAM-2024-SED", "manufacturer": "Toyota", "model":
"Camry", "year": 2024, "body_class": "Sedan", "instance_count": 156, "first_seen":
"2024-01-15T10:30:00Z", "last_seen": "2024-11-20T14:22:00Z", "drive_type": "FWD",
"engine": "I4", "transmission": "Automatic", "fuel_type": "Gasoline" },
{ "vehicle_id": "TOY-RAV-2024-SUV", "manufacturer": "Toyota", "model": "RAV4", "year":
2024, "body_class": "SUV", "instance_count": 234, "first_seen":
"2024-02-10T08:15:00Z", "last_seen": "2024-11-19T16:45:00Z", "drive_type": "AWD",
"engine": "I4", "transmission": "Automatic", "fuel_type": "Hybrid" } ], "total": 234,
"page": 1, "size": 20, "totalPages": 12, "statistics": { "totalCount": 234,
"byManufacturer": { "Toyota": { "total": 234, "highlighted": 234 } }, "byBodyClass":
{ "Sedan": { "total": 89, "highlighted": 89 }, "SUV": { "total": 78, "highlighted":
78 }, "Truck": { "total": 45, "highlighted": 45 }, "Coupe": { "total": 22,
"highlighted": 22 } }, "byYear": { "2024": { "total": 67, "highlighted": 67 }, "2023":
{ "total": 58, "highlighted": 58 }, "2022": { "total": 52, "highlighted": 52 },
"2021": { "total": 34, "highlighted": 34 }, "2020": { "total": 23, "highlighted":
23 } }, "modelsByManufacturer": { "Toyota": { "Camry": { "total": 45, "highlighted":
45 }, "RAV4": { "total": 52, "highlighted": 52 }, "Corolla": { "total": 38,
"highlighted": 38 }, "Highlander": { "total": 29, "highlighted": 29 } } } } }
```

VehicleResult Object

Each item in the `results` array has this shape:

```
interface VehicleResult {

// Required fields vehicle_id: string;      // Unique ID: "MANUFACTURER-MODEL-YEAR-
BODYCLASS" manufacturer: string;    // e.g., "Toyota", "Honda", "Ford" model:
string;           // e.g., "Camry", "Accord", "F-150" year: number;            //
e.g., 2024 body_class: string;      // e.g., "Sedan", "SUV", "Truck" instance_count:
number;  // Number of VINs for this vehicle config



// Optional fields first_seen?: string;     // ISO 8601 datetime last_seen?:
string;      // ISO 8601 datetime drive_type?: string;     // e.g., "FWD", "RWD",
"AWD", "4WD" engine?: string;         // e.g., "V6", "I4", "V8", "Electric"
transmission?: string;   // e.g., "Automatic", "Manual", "CVT" fuel_type?:
string;      // e.g., "Gasoline", "Diesel", "Electric", "Hybrid" vehicle_class?:
string;  // e.g., "Passenger Car", "Light Truck" }
```

Field Notes

Field	Format	Example
`vehicle_id`	`{MFR}-{MODEL}-{YEAR}-{BODY}`	`TOY-CAM-2024-SED`
`year`	4-digit integer	`2024`
`instance_count`	Non-negative integer	`156`
`first_seen`	ISO 8601	`2024-01-15T10:30:00Z`
`body_class`	Title case	`Sedan`, `SUV`, `Truck`

Statistics Object

The `statistics` field in the response contains aggregated data for charts and summaries.

Structure

```
interface VehicleStatistics {
// Total count of vehicles matching filters totalCount: number;



// Segmented by manufacturer: { "Toyota": { total, highlighted }, ... }
byManufacturer: Record<string, { total: number; highlighted: number }>;



// Segmented by body class: { "Sedan": { total, highlighted }, ... } byBodyClass:
Record<string, { total: number; highlighted: number }>;



// Segmented by year: { "2024": { total, highlighted }, ... } byYear: Record<string,
{ total: number; highlighted: number }>;



// Models nested under manufacturers modelsByManufacturer: Record<string,
Record<string, { total: number; highlighted: number }>>; }
```

Segmented Statistics Explained

Each statistic entry contains:

- `total`: Count of all items in this category
- `highlighted`: Count of items matching highlight filters (h_* parameters)

Example without highlights:

```
{
"byManufacturer": { "Toyota": { "total": 234, "highlighted": 234 }, "Honda":
{ "total": 187, "highlighted": 187 } } }
```

When no highlight filters are applied, `highlighted` equals `total`.

Example with highlights (`h_yearMin=2022&h_yearMax=2024`):

```
{
"byManufacturer": { "Toyota": { "total": 234, "highlighted": 156 }, "Honda":
{ "total": 187, "highlighted": 98 } } }
```

Only 156 of Toyota's 234 vehicles fall within 2022-2024.

Endpoint: GET /statistics

Fetches statistics only, without vehicle data. Useful for refreshing charts without reloading the table.

Request

URL: `{baseUrl}/statistics`

Query Parameters: Same filter parameters as `/vehicles/details` (excluding `page` , `size` , `sortBy` , `sortOrder`)

Example Request

```
GET /statistics?manufacturer=Toyota&yearMin=2020
```

Response Shape

Same as the `statistics` object from `/vehicles/details` .

URL Parameter Mapping

This table maps URL query parameters to filter object properties:

URL Parameter	Filter Property	Type
`manufacturer`	`manufacturer`	string
`model`	`model`	string
`yearMin`	`yearMin`	number
`yearMax`	`yearMax`	number
`bodyClass`	`bodyClass`	string
`instanceCountMin`	`instanceCountMin`	number
`instanceCountMax`	`instanceCountMax`	number
`search`	`search`	string
`models`	`modelCombos`	string
`page`	`page`	number
`size`	`size`	number
`sortBy`	`sort`	string
`sortOrder`	`sortDirection`	string

Note: The URL uses `models` but the filter object uses `modelCombos`. The adapter handles this translation.

Model Combinations Format

The `models` parameter (and `h_modelCombos` highlight parameter) uses a specific format:

```
Manufacturer:Model,Manufacturer:Model,...
```

Examples:

- Single model: `Ford:F-150`
- Multiple models: `Ford:F-150,Toyota:Camry,Honda:Accord`
- Same manufacturer, different models: `Toyota:Camry,Toyota:Corolla`

Field Name Conventions

The API uses **snake_case** for field names:

- `vehicle_id`, `body_class`, `instance_count`, `first_seen`, `last_seen`
- `fuel_type`, `drive_type`, `vehicle_class`

The frontend models use **camelCase** internally, with adapters handling the translation.

Next Step

Proceed to `053-naming-conventions.md` to understand which code is framework (reusable across domains) vs domain-specific (automobile only).

053: Naming Conventions — Framework vs Domain

Status: Reference **Depends On:** 051, 052 **Blocks:** All implementation phases

---

Objective

Establish a clear distinction between **framework code** (domain-agnostic, reusable) and **domain code** (automobile-specific). This separation enables:

- A smaller companion book for adding new domains (e.g., "Adding Agriculture to Vroom")
- Clear understanding of which code changes when adding a new domain
- Proper placement of new code during development

Why This Matters

The vroom application is built on a **configuration-driven architecture**. The framework provides generic capabilities, and domain configuration tells it how to behave for a specific data domain.

To add a new domain (e.g., agriculture), you only implement the right column.

The framework code (left column) remains untouched.

Framework vs Domain: Complete Reference

Services

Framework (Never Changes)	Domain-Specific (Per Domain)
`ResourceManagementService`	—
`UrlStateService`	—
`ApiService`	—
`RequestCoordinatorService`	—
`DomainConfigRegistry`	—
`DomainConfigValidator`	—
`PopOutContextService`	—
`PopOutManagerService`	—
`UserPreferencesService`	—
`FilterOptionsService`	—
`PickerConfigRegistry`	—
`ErrorNotificationService`	—
`HttpErrorInterceptor`	—
`GlobalErrorHandler`	—

Note: Services are 100% framework. Domains don't create new services.

Interfaces

Framework (Never Changes)	Domain-Specific (Per Domain)
`DomainConfig<TFilters, TData, TStats>`	—
`IApiAdapter<TFilters, TData, TStats>`	`AutomobileApiAdapter` (implements)
`IFilterUrlMapper<TFilters>`	`AutomobileUrlMapper` (implements)
`ICacheKeyBuilder<TFilters>`	`AutomobileCacheKeyBuilder` (implements)
`ResourceState<TFilters, TData, TStats>`	—
`ApiResponse<TData>`	—
`FilterDefinition`	—
`TableConfig`	—
`PickerConfig`	—
`ChartConfig`	—

Models

Framework (Never Changes)	Domain-Specific (Per Domain)
—	`AutoSearchFilters`
—	`HighlightFilters`
—	`VehicleResult`
—	`VinInstance`
—	`VehicleStatistics`
—	`ManufacturerStat`
—	`ModelStat`
—	`BodyClassStat`
—	`YearStat`

Note: Models are 100% domain-specific. Each domain defines its own data shapes.

Adapters

Framework (Never Changes)	Domain-Specific (Per Domain)
Interface: `IApiAdapter`	`AutomobileApiAdapter`
Interface: `IFilterUrlMapper`	`AutomobileUrlMapper`
Interface: `ICacheKeyBuilder`	`AutomobileCacheKeyBuilder`

Pattern: Framework defines interfaces. Domains implement them.

Configurations

Framework (Never Changes)	Domain-Specific (Per Domain)
—	`automobile.filter-definitions.ts`
—	`automobile.table-config.ts`
—	`automobile.picker-configs.ts`
—	`automobile.query-control-filters.ts`
—	`automobile.highlight-filters.ts`
—	`automobile.chart-configs.ts`
—	`automobile.domain-config.ts`

Chart Data Sources

Framework (Never Changes)	Domain-Specific (Per Domain)
Interface: `ChartDataSource`	`ManufacturerChartSource`
—	`YearChartSource`
—	`BodyClassChartSource`
—	`TopModelsChartSource`

Components

Framework (Never Changes)	Domain-Specific (Per Domain)
`BaseChartComponent`	—
`BasePickerComponent`	—
`BasicResultsTableComponent`	—
`ResultsTableComponent`	—
`DynamicResultsTableComponent`	—
`QueryPanelComponent`	—
`QueryControlComponent`	—
`StatisticsPanel2Component`	—
`DockviewStatisticsPanelComponent`	—

Note: UI components are 100% framework. They render based on configuration.

Feature Components

Framework (Never Changes)	Domain-Specific (Per Domain)
`HomeComponent`	—
`PopoutComponent`	—
—	`AutomobileComponent` (landing)
—	`AutomobileDiscoverComponent` (main page)

Pattern: The discover page is domain-specific because it wires up domain configuration. Popout is framework because it renders any domain's panels.

Routes

Framework (Never Changes)	Domain-Specific (Per Domain)
`/` (redirect)	—
`/home`	—
`/popout/:gridId/:componentId`	—
—	`/automobiles`
—	`/automobiles/discover`

Directory Structure

```
src/
├── app/ │   ├── app.component.ts           # Framework │   ├──
app.config.ts              # Framework │   ├── app.routes.ts              # Framework
+ Domain routes │   └── features/ │       ├── home/                  # Framework
│       ├── popout/               # Framework │       └── automobile/           #
DOMAIN-SPECIFIC │           ├── automobile.component.ts │           └── automobile-
discover/ │ ├── framework/                    # ALL FRAMEWORK │   ├── components/ │
├── models/ │   ├── services/ │   └── tokens/ │ ├── domain-config/                #
ALL DOMAIN-SPECIFIC │   ├── domain-providers.ts        # Registers all domains │   └──
automobile/ │       ├── adapters/ │       ├── chart-sources/ │       ├── configs/
│       ├── models/ │       ├── automobile.domain-config.ts │       └── index.ts │ └──
environments/                  # Framework
```

Observable Streams (Framework)

These observable names are **framework conventions** — they don't change per domain:

Observable	Type	Description
`state$`	`Observable<ResourceState>`	Complete state object
`filters$`	`Observable<TFilters>`	Current filter values
`results$`	`Observable<TData[]>`	Current page results
`totalResults$`	`Observable<number>`	Total count
`loading$`	`Observable<boolean>`	Loading state
`error$`	`Observable<Error \`	null>	Error state
`statistics$`	`Observable<TStats>`	Statistics data
`highlights$`	`Observable<any>`	Highlight filters

Methods (Framework)

These method names are **framework conventions**:

Method	Signature	Description
`updateFilters()`	`(partial: Partial<TFilters>) => void`	Update filters via URL
`clearFilters()`	`() => void`	Reset to defaults
`refresh()`	`() => void`	Re-fetch with current filters
`getCurrentState()`	`() => ResourceState`	Get state snapshot
`getCurrentFilters()`	`() => TFilters`	Get filters snapshot

Injection Tokens (Framework)

Token	Type	Description
`DOMAIN_CONFIG`	`InjectionToken<DomainConfig>`	Provides domain configuration
`IS_POPOUT_TOKEN`	`InjectionToken<boolean>`	Indicates pop-out context

What Changes When Adding a New Domain?

To add agriculture support to vroom, you would create:

```
src/domain-config/agriculture/

├── adapters/ │   ├── agriculture-url-mapper.ts      # Implements IFilterUrlMapper │
├── agriculture-api.adapter.ts     # Implements IApiAdapter │   └── agriculture-cache-
key-builder.ts ├── chart-sources/ │   ├── region-chart-source.ts │   └── crop-chart-
source.ts ├── configs/ │   ├── agriculture.filter-definitions.ts │   ├──
agriculture.table-config.ts │   └── agriculture.chart-configs.ts ├── models/ │   ├──
agriculture.filters.ts │   ├── agriculture.data.ts │   └── agriculture.statistics.ts
├── agriculture.domain-config.ts └── index.ts



src/app/features/agriculture/ ├── agriculture.component.ts └── agriculture-discover/
└── agriculture-discover.component.ts
```

And update:

- `src/domain-config/domain-providers.ts` (register the new domain)
- `src/app/app.routes.ts` (add agriculture routes)

No changes to framework code.

Naming Patterns

Domain Config Files

Pattern: `{domain}.{type}.ts`

Examples:

- `automobile.filter-definitions.ts`
- `automobile.table-config.ts`
- `agriculture.filter-definitions.ts`

Adapter Classes

Pattern: `{Domain}{Type}` (PascalCase)

Examples:

- `AutomobileApiAdapter`

- `AutomobileUrlMapper`
- `AgricultureApiAdapter`

Model Classes

Pattern: `{DomainEntity}` (domain-meaningful names)

Examples:

- `VehicleResult` (not `AutomobileResult`)
- `AutoSearchFilters` (not `AutomobileFilters`)
- `CropYield` (for agriculture)

Feature Components

Pattern: `{Domain}DiscoverComponent`

Examples:

- `AutomobileDiscoverComponent`
- `AgricultureDiscoverComponent`

Companion Book Scope

A companion book "Adding Agriculture to Vroom" would cover only:

Phase	Documents	Content
API Contract	1	Agriculture endpoints
Domain Models	3	Filters, Data, Statistics
Domain Adapters	3	URL Mapper, API Adapter, Cache Key Builder
Domain Configs	6	Filter definitions, table, charts, etc.
Chart Sources	2-4	Domain-specific chart transformations
Feature Components	2	Landing page, Discover page
Routes	1	Add routes to app.routes.ts
Total	~18	~150 pages

Compare to the full vroom book: **65 documents, ~525 pages**.

The companion book is **~70% smaller** because all framework code is reused.

Summary

Category	Framework	Domain
Services	14	0
Interfaces	10	0 (implement them)
Models	0	9
Adapters	0	3
Configs	0	7
Chart Sources	0	4
UI Components	9	0
Feature Components	2	2
Effort to add domain	0%	100%

This document is the key to understanding vroom's architecture. Consult it whenever uncertain about where new code belongs.

101: Project Cleanup

Status: Planning **Depends On:** 000-book-conventions, 051-api-contract-overview **Blocks:** 102-app-shell

---

Learning Objectives

After completing this section, you will:

- Understand why removing boilerplate establishes ownership of your codebase
- Know how Angular environment files enable configuration without code changes
- Recognize the directory structure that separates framework code from domain code

---

Objective

Remove Angular CLI boilerplate and establish the directory structure that will support our URL-First architecture. After this section, you will have a clean foundation ready for building the vvroom application.

Why

When you generate a new Angular project with `ng new`, the CLI creates placeholder content designed to help newcomers verify their setup works. This placeholder content includes:

- A welcome message with the Angular logo
- Links to Angular documentation
- Sample text that has nothing to do with your application

This is noise. Every line of code in your project should serve a purpose. Removing boilerplate immediately establishes a professional mindset: you own every line of code in this project, and you understand why it's there.

Additionally, we need to:

- **Configure the environment** — Set the API base URL so our services know where to fetch data
- **Establish the directory structure** — Create folders that match our architecture before we need them
- **Update the HTML title** — Small detail, but professionalism shows in the details

Angular Style Guide References

- Style 04-06: Create sub-folders for feature areas
- Style 04-07: Create a folder for each feature module

What

Step 101.1: Understand the Current State

Before making changes, examine what the Angular CLI generated. Open a terminal and navigate to your project:

```
$ cd ~/projects/vvroom
$ ls -la src/app/
```

You should see:

```
total 16
drwxr-xr-x 2 user user 4096 Feb  9 10:00 . drwxr-xr-x 5 user user 4096 Feb  9 10:00 ..
-rw-r--r-- 1 user user 1234 Feb  9 10:00 app.component.ts -rw-r--r-- 1 user user  345
Feb  9 10:00 app.module.ts
```

The current `src/app/app.component.ts` contains placeholder content with an inline template showing a welcome message, the Angular logo, and documentation links. This is what we'll clean up.

Step 101.2: Clean Up AppComponent

Open `src/app/app.component.ts` and replace its entire contents with:

```
// src/app/app.component.ts

// VERSION 1 (Section 101) - Minimal placeholder // This will be replaced with the full shell in Section 102



import { Component } from '@angular/core';



@Component({ selector: 'app-root', template: `<h1>vvroom</h1> <p>Automobile Discovery Platform</p>

`, styles: [] }) export class AppComponent {}
```

What changed:

Before	After
32 lines with placeholder content	13 lines of clean code
`title` property (unused baggage)	No unnecessary properties
Inline Angular logo (base64 SVG)	Removed
Documentation links	Removed

Why this template?

The simple heading and tagline serve as a "smoke test" — when you run the app, you'll immediately see whether your changes worked. We're not adding navigation or routing yet; that comes in document 102.

Step 101.3: Configure Environment for API Access

Our application will fetch data from the automobile API. Configure the base URL in the environment file.

Open `src/environments/environment.ts` and replace its contents with:

```
// src/environments/environment.ts

export const environment = { production: false, apiBaseUrl: 'http://generic-
prime.minilab/api/specs/v1' };
```

Now create the production environment file. Open `src/environments/environment.prod.ts` and replace its contents with:

```
// src/environments/environment.prod.ts

export const environment = { production: true, apiBaseUrl: 'http://generic-
prime.minilab/api/specs/v1' };
```

Why the same URL for both?

In this application, development and production use the same API server. In a typical enterprise environment, you might have:

- Development: `http://localhost:3000/api`
- Production: `https://api.yourcompany.com/v1`

The environment file pattern allows this flexibility without changing application code.

Step 101.4: Create Directory Structure

Create the directories that will hold our framework and domain-specific code:

```
$ cd ~/projects/vvroom

# Framework directories (domain-agnostic, reusable)

$ mkdir -p src/app/framework/services $ mkdir -p src/app/framework/models $ mkdir -p src/app/framework/components $ mkdir -p src/app/framework/tokens

# Feature directories (page-level components)

$ mkdir -p src/app/features/home $ mkdir -p src/app/features/discover $ mkdir -p src/app/features/popout

# Domain configuration (automobile-specific)

$ mkdir -p src/app/domain-config/automobile/models $ mkdir -p src/app/domain-config/automobile/adapters $ mkdir -p src/app/domain-config/automobile/configs $ mkdir -p src/app/domain-config/automobile/chart-sources
```

Verify the structure:

```
$ find src/app -type d | sort
```

Expected output:

```
src/app

src/app/domain-config src/app/domain-config/automobile src/app/domain-config/
automobile/adapters src/app/domain-config/automobile/chart-sources src/app/domain-
config/automobile/configs src/app/domain-config/automobile/models src/app/features
src/app/features/discover src/app/features/home src/app/features/popout src/app/
framework src/app/framework/components src/app/framework/models src/app/framework/
services src/app/framework/tokens
```

What do these directories mean?

Directory	Purpose	Examples
`framework/services`	Domain-agnostic services	`UrlStateService`, `ApiService`
`framework/models`	TypeScript interfaces for framework	`DomainConfig`, `ApiResponse`
`framework/components`	Reusable UI components	`BaseChartComponent`, `ResultsTableComponent`
`framework/tokens`	Angular injection tokens	`DOMAIN_CONFIG`, `IS_POPOUT_TOKEN`
`features/*`	Page-level components	`HomeComponent`, `DiscoverComponent`
`domain-config/automobile/*`	Automobile-specific configuration	Filters, adapters, chart sources

This structure directly reflects the architecture described in `053-naming-conventions.md`. Framework code never changes when you add a new domain; only `domain-config/` grows.

Step 101.5: Add Placeholder Files

Empty directories are ignored by git. Add `.gitkeep` files to preserve the structure:

```
$ cd ~/projects/vvroom


# Add .gitkeep to each empty directory


$ touch src/app/framework/services/.gitkeep $ touch src/app/framework/models/.gitkeep
$ touch src/app/framework/components/.gitkeep $ touch src/app/framework/
tokens/.gitkeep $ touch src/app/features/home/.gitkeep $ touch src/app/features/
discover/.gitkeep $ touch src/app/features/popout/.gitkeep $ touch src/app/domain-
config/automobile/models/.gitkeep $ touch src/app/domain-config/automobile/
adapters/.gitkeep $ touch src/app/domain-config/automobile/configs/.gitkeep $ touch
src/app/domain-config/automobile/chart-sources/.gitkeep
```

Note: These `.gitkeep` files are a convention, not a git feature. They're empty files that exist solely to make git track otherwise-empty directories. We'll delete them as we add real files to each directory.

Step 101.6: Update Page Title

Open `src/index.html` and verify it has a meaningful title:

```
<!doctype html>

<html lang="en"> <head> <meta charset="utf-8"> <title>Vvroom - Automobile Discovery</
title> <base href="/"> <meta name="viewport" content="width=device-width, initial-
scale=1"> <link rel="icon" type="image/x-icon" href="favicon.ico"> </head> <body>
<app-root></app-root> </body> </html>
```

The only change is the `<title>` tag: from "Vvroom" to "Vvroom - Automobile Discovery".

Step 101.7: Add Base Styles

Open `src/styles.css` and add minimal global styles:

```
/ src/styles.css /

/ Reset and base styles / , ::before, *::after { box-sizing: border-box; }



body { margin: 0; font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto,
Oxygen, Ubuntu, sans-serif; font-size: 16px; line-height: 1.5; color: #333;
background-color: #f5f5f5; }



h1, h2, h3, h4, h5, h6 { margin-top: 0; margin-bottom: 0.5rem; font-weight: 500; line-
height: 1.2; }



a { color: #1976d2; text-decoration: none; }



a:hover { text-decoration: underline; }
```

Why these styles?

- `box-sizing: border-box` — Makes width/height calculations predictable (padding included)
- System font stack — Uses the operating system's native font for fast loading and native feel
- Reset margins — Browsers have inconsistent defaults; we normalize them
- Link color — A professional blue that's accessible and familiar

These styles provide a clean foundation. We'll add component-specific styles as we build the UI.

Verification

1. Build the Application

```
$ cd ~/projects/vvroom

$ ng build
```

Expected output (no errors):

```
✔ Browser application bundle generation complete.

✔ Copying assets complete. ✔ Index html generation complete.



Initial Chunk Files           | Names         |  Raw Size
main.js                       | main          |  47.42 kB |
polyfills.js                  | polyfills     | 339.16 kB |
runtime.js                    | runtime       |   6.54 kB |
styles.css                    | styles        |   1.23 kB |



Build at: 2026-02-09T17:00:00.000Z - Hash: abc123 - Time: 5000ms
```

2. Serve the Application

```
$ ng serve --open
```

Your browser should open to `http://localhost:4200` showing:

```
vvroom

Automobile Discovery Platform
```

If you see the Angular logo and "Welcome to vvroom!" links, you missed Step 101.2. Go back and update `app.component.ts`.

3. Check the Browser Tab

The browser tab should show "Vvroom - Automobile Discovery" (not just "Vvroom").

4. Verify Directory Structure

```
$ find src/app -type d | wc -l
```

Expected: `16` (the app directory plus 15 subdirectories we created)

5. Check Environment Configuration

```
$ grep -r "apiBaseUrl" src/environments/
```

Expected output:

```
src/environments/environment.ts:  apiBaseUrl: 'http://generic-prime.minilab/api/specs/v1'
src/environments/environment.prod.ts:  apiBaseUrl: 'http://generic-prime.minilab/api/specs/v1'
```

Common Problems

Symptom	Cause	Solution
`ng build` fails with "Cannot find module '@angular/core'"	Node modules not installed	Run `npm install` in the project root
Browser shows blank page	Template syntax error in `app.component.ts`	Check for missing backticks or quotes in the template
"Vvroom" title instead of "Vvroom - Automobile Discovery"	Didn't save `index.html`	Save the file and refresh the browser
Directories not created	Typo in `mkdir` command	Re-run the commands carefully; use tab completion
`ng serve` shows "Port 4200 is already in use"	Another process using the port	Kill the other process or use `ng serve --port 4201`

Key Takeaways

- **Own your code** — Remove boilerplate so every line serves a purpose you understand
- **Environment files separate configuration from code** — Change URLs without changing TypeScript
- **Directory structure reflects architecture** — Framework code in `framework/`, domain code in `domain-config/`

Acceptance Criteria

- [ ] `src/app/app.component.ts` contains only the clean template (no Angular logo, no links)
- [ ] Application displays "vvroom" heading and "Automobile Discovery Platform" tagline
- [ ] `src/environments/environment.ts` includes `apiBaseUrl` property
- [ ] `src/environments/environment.prod.ts` includes `apiBaseUrl` property
- [ ] Directory structure created: `framework/`, `features/`, `domain-config/`
- [ ] All directories contain `.gitkeep` placeholder files
- [ ] Browser tab shows "Vvroom - Automobile Discovery"
- [ ] `src/styles.css` contains base reset styles
- [ ] `ng build` completes with no errors
- [ ] `ng serve` shows the clean application

What We Accomplished

Item	Before	After
AppComponent	32 lines of boilerplate	13 lines of clean code
Environment config	No API URL	API URL configured
Directory structure	Flat `src/app/`	Organized by architecture
Page title	Generic "Vvroom"	Descriptive "Vvroom - Automobile Discovery"
Global styles	Empty	Professional base styles

Next Step

Proceed to `102-app-shell.md` to build the application shell with navigation layout.

102: App Shell

Status: Planning **Depends On:** 101-project-cleanup **Blocks:** 103-routing

---

Learning Objectives

After completing this section, you will:

- Understand the container/presentational component pattern for application layout
- Know how to use the `:host` CSS selector to style Angular components
- Be able to create a flexbox-based full-viewport layout

Objective

Build the application shell — the outermost structural component that provides consistent navigation and layout across all pages. After this section, you'll have a header with navigation links and a content area where routed components will render.

Why

Every web application needs a shell: a consistent frame that surrounds page content. The shell typically includes:

- **Header** — Application name/logo and primary navigation
- **Content area** — Where page-specific content renders
- **Optional footer** — Copyright, links, version info

Building the shell before routing has practical benefits:

- **Visual confirmation** — You see the navigation structure before wiring it up

- **Router outlet placement** — You know exactly where routed content will appear
- **Separation of concerns** — Layout logic stays in AppComponent; page logic stays in feature components

Angular Style Guide References

- Style 02-01: Use consistent naming for components
- Style 05-03: Put presentation logic in the component class

URL-First Architecture Reference

The shell is framework code — it doesn't change when you add new domains. The navigation links will eventually include domain-specific routes, but the shell structure remains constant.

What

Step 102.1: Design the Shell Layout

Before writing code, understand what we're building:

The shell has two parts:

- **Header** — Fixed at the top, contains logo and navigation
- **Main content** — Takes remaining vertical space, contains `<router-outlet>`

Step 102.2: Update AppComponent with Shell Structure

Open `src/app/app.component.ts` and replace its contents with:

```
// src/app/app.component.ts

// VERSION 2 (Section 102) - Shell with navigation // Replaces VERSION 1 from Section
101



import { Component } from '@angular/core';



@Component({ selector: 'app-root', template: <header class="app-header"> <div
class="app-header-brand"> <span class="app-header-logo">🚗</span> <span
class="app-header-title">vvroom</span> </div> <nav class="app-header-nav"> <a
class="nav-link" href="/home">Home</a> <a class="nav-link" href="/
discover">Discover</a> </nav> </header> <main class="app-content"> <p>Content
will appear here once routing is configured.</p> </main>

,

styles: [ :host { display: flex; flex-direction: column; min-height: 100vh; }



.app-header { display: flex; align-items: center; justify-content: space-
between; padding: 0 1.5rem; height: 56px; background-color: #1976d2; color:
white; box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1); }



.app-header-brand { display: flex; align-items: center; gap: 0.5rem; }



.app-header-logo { font-size: 1.5rem; }



.app-header-title { font-size: 1.25rem; font-weight: 600; letter-spacing:
-0.5px; }



.app-header-nav { display: flex; gap: 0.5rem; }

```

```
.nav-link { padding: 0.5rem 1rem; color: rgba(255, 255, 255, 0.9); text-
decoration: none; border-radius: 4px; transition: background-color 0.2s; }

.nav-link:hover { background-color: rgba(255, 255, 255, 0.1); text-decoration:
none; }

.app-content { flex: 1; padding: 1.5rem; }

]
}) export class AppComponent {}
```

What this code does:

Element	Purpose
`:host`	Styles the component itself as a flex column taking full viewport height
`.app-header`	Blue header bar with flexbox layout
`.app-header-brand`	Logo and title grouped together
`.app-header-nav`	Navigation links aligned to the right
`.nav-link`	Styled anchor tags with hover effect
`.app-content`	Main content area with `flex: 1` to fill remaining space

Why inline styles?

For small components, inline styles in the `styles` array keep everything in one file. This follows Angular's recommendation for simple components. We'll extract styles to separate files for larger components later.

Note on the emoji: We're using 🚗 as a temporary logo. In a production application, you'd use an SVG or image file. The emoji works fine for learning.

Step 102.3: Understand the Navigation Links (Temporary)

Notice the navigation links use plain `href` attributes:

```
<a class="nav-link" href="/home">Home</a>
<a class="nav-link" href="/discover">Discover</a>
```

This is intentional but temporary. These are standard HTML links that cause a full page reload. In the next document (103-routing), we'll:

- Import `RouterModule`
- Replace `href` with `routerLink`
- Add `routerLinkActive` for highlighting the current route

For now, clicking these links will show a 404 or reload the app — that's expected.

Step 102.4: Update AppModule to Import Required Modules

The current `AppModule` is minimal. We don't need any additional imports yet, but let's verify it's correct.

Open `src/app/app.module.ts` and confirm it contains:

```
// src/app/app.module.ts

import { NgModule } from '@angular/core'; import { BrowserModule } from '@angular/
platform-browser';



import { AppComponent } from './app.component';



@NgModule({ declarations: [ AppComponent ], imports: [ BrowserModule ], providers: [],
bootstrap: [AppComponent] }) export class AppModule {}
```

No changes needed yet. We'll add `RouterModule` in document 103.

Step 102.5: Understanding the Flex Layout

The shell uses flexbox for layout. Here's how it works:

```
:host {
display: flex; flex-direction: column; min-height: 100vh; }
```

This makes the component a vertical flex container that's at least viewport height.

```
.app-header {
height: 56px; / ... other styles ... / }
```

The header has a fixed height of 56px (a common Material Design height).

```
.app-content {
flex: 1; / ... other styles ... / }
```

The content area uses `flex: 1` to expand and fill all remaining vertical space.

Visual result:

```
┌──────────────────────────────────────┐
| Header (fixed 56px)                  | ├──────────────────────────────────────┤
|                                      | | Content (flexible, fills rest)     |
|                                      | |                                    |
|                                      | └──────────────────────────────────────┘
```

This ensures the app always fills the viewport, even with little content.

Step 102.6: The Host Element Pattern

Notice we style `:host` rather than adding a wrapper div:

```
:host {
display: flex; flex-direction: column; min-height: 100vh; }
```

`:host` is a CSS pseudo-class that targets the component's host element — in this case, `<app-root>`. This is an Angular best practice:

Without :host (anti-pattern):

```
<app-root>
<div class="wrapper">  <!-- Unnecessary wrapper --> <header>...</header> <main>...</
main> </div> </app-root>
```

With :host (correct):

```
<app-root>  <!-- app-root IS the flex container -->
<header>...</header> <main>...</main> </app-root>
```

One less DOM element, cleaner structure, same result.

Verification

1. Build the Application

```
$ cd ~/projects/vvroom
$ ng build
```

Expected: Build succeeds with no errors.

2. Serve the Application

```
$ ng serve --open
```

Expected: Browser opens to `http://localhost:4200`

3. Visual Check

You should see:

```
┌──────────────────────────────────────────────────────────────┐
│  🚗 vvroom                                Home    Discover   │
├──────────────────────────────────────────────────────────────┤
│                                                              │ │  Content will
appear here once routing is configured.            │
│                                                              │
└──────────────────────────────────────────────────────────────┘
```

- Blue header with white text
- Logo emoji and "vvroom" title on the left
- "Home" and "Discover" links on the right
- Gray background in the content area
- Links change background on hover

4. Test Navigation Links (Expected Behavior)

Click "Home" or "Discover":

- The page will reload or show an error
- This is expected — we haven't configured routing yet

5. Responsive Check

Resize your browser window:

- The header should stay at the top
- The content area should resize fluidly

- Navigation links should remain visible (no responsive menu yet)

6. Inspect the DOM

Open browser developer tools (F12) and inspect the HTML:

```
<app-root>
<header class="app-header">...</header> <main class="app-content">...</main> </app-root>
```

Notice there's no wrapper div — the `<app-root>` element itself is the flex container.

Common Problems

Symptom	Cause	Solution
Header not full width	Missing `:host` styles	Ensure `:host { display: flex; }` is present
Content area doesn't fill viewport	Missing `min-height: 100vh` on `:host`	Add the missing style
Emoji not displaying	System font doesn't support emoji	This is rare; try a different browser
Styles not applying	Syntax error in `styles` array	Check for missing backticks or brackets
White gap at bottom of page	`flex: 1` not on `.app-content`	Add `flex: 1` to the content area

---

Key Takeaways

- **The shell is the application frame** — It provides consistent structure across all pages
- **`:host` styles the component element** — No wrapper divs needed
- **`flex: 1` fills remaining space** — Combined with `min-height: 100vh`, creates full-viewport layouts

Acceptance Criteria

- [ ] `src/app/app.component.ts` contains the shell template with header and content area
- [ ] Header displays logo (🚗), title (vvroom), and navigation links (Home, Discover)
- [ ] Header has blue background (#1976d2) with white text
- [ ] Navigation links have hover effect
- [ ] Content area fills remaining viewport height
- [ ] `:host` selector is used for component-level layout
- [ ] `ng build` completes with no errors
- [ ] `ng serve` shows the shell correctly in the browser

What We Accomplished

Item	Before	After
AppComponent template	Simple heading	Full shell with header and content
Layout	None	Flexbox-based full-height layout
Navigation	None	Placeholder links (non-functional)
Styling	None	Professional header with hover effects
Component pattern	Basic	Uses `:host` for layout

Architecture Note

The app shell follows the **container/presentational pattern**:

- **AppComponent (container)** — Provides structure and layout
- **Feature components (presentational)** — Render inside the content area

This separation means:

- Layout changes happen in one place (AppComponent)

- Feature components focus on their specific functionality
- The shell is framework code — it works for any domain

Next Step

Proceed to `103-routing.md` to configure Angular Router and make the navigation links functional.

103: Routing

Status: Planning **Depends On:** 102-app-shell **Blocks:** 104-environment-config

---

Learning Objectives

After completing this section, you will:

- Understand how Angular Router maps URLs to components
- Know how to use `routerLink` for navigation without full page reloads
- Be able to highlight the current route using `routerLinkActive`

Objective

Configure Angular Router with routes for Home, Discover, and Popout pages. Replace the placeholder `href` links with Angular's `routerLink` directive so navigation updates the URL without reloading the page.

Why

The URL is the foundation of our application's state management. Before we build any features, we need the routing skeleton in place. Here's why routing comes early:

URLs Are State

In traditional web applications, the server generates each page. In a single-page application (SPA), the client handles navigation — but the URL remains the source of truth. When a user bookmarks `/discover?make=Toyota`, they expect to return to that exact view.

This is the core insight of **URL-First State Management**: the URL isn't just an address; it's a serialized representation of application state.

Why Not Just Use `href`?

Standard HTML links (`<a href="/home">`) work, but they cause problems:

`href` Behavior	Problem
Full page reload	Loses JavaScript state
Server request	Slower navigation
Flash of white	Poor user experience

Angular's `routerLink` directive intercepts clicks and updates the URL without reloading. The Router then renders the appropriate component. This is faster, smoother, and preserves application state.

Angular Style Guide References

- Style 04-10: Use redirects for default routes
- Style 02-05: Suffix routing modules with `-routing`

URL-First Architecture Reference

See `docs/README.md` for the full URL-First pattern. This section establishes the routing foundation that later sections will build upon.

What

Step 103.1: Create the Home Component

The Home component is a simple landing page. Create the file `src/app/features/home/home.component.ts`:

```
// src/app/features/home/home.component.ts
// VERSION 1 (Section 103) - Placeholder component


import { Component } from '@angular/core';


@Component({ selector: 'app-home', template: `
  <div class="home-container">
    <h1>Welcome to Vvroom</h1>
    <p>Your automobile discovery platform.</p>
    <p>Use the navigation above to explore vehicles.</p>
  </div>
`,
  styles: [`
    .home-container { max-width: 600px; margin: 2rem auto; text-align: center; }
    h1 { color: #1976d2; margin-bottom: 1rem; }
    p { color: #666; margin-bottom: 0.5rem; }
  `]
}) export class HomeComponent {}
```

Delete the `.gitkeep` file since the directory now has real content:

```
$ rm src/app/features/home/.gitkeep
```

Step 103.2: Create the Discover Component

The Discover component will eventually display vehicle search results. For now, it's a placeholder. Create `src/app/features/discover/discover.component.ts`:

```
// src/app/features/discover/discover.component.ts

// VERSION 1 (Section 103) - Placeholder component // This will be replaced with the
full discover page in Phase 9



import { Component } from '@angular/core';



@Component({ selector: 'app-discover',
template:  <div class="discover-container"> <h1>Discover Vehicles</h1>
<p>Vehicle search and filtering will appear here.</p> <p class="hint">Watch
the URL bar as you navigate — this is where state will live.</p> </div>

,
styles: [  .discover-container { max-width: 800px; margin: 2rem auto; }

h1 { color: #1976d2; margin-bottom: 1rem; }

p { color: #666; margin-bottom: 0.5rem; }

.hint { margin-top: 2rem; padding: 1rem; background-color: #e3f2fd; border-
left: 4px solid #1976d2; color: #1565c0; }

]
}) export class DiscoverComponent {}
```

Delete the `.gitkeep` file:

```
$ rm src/app/features/discover/.gitkeep
```

Step 103.3: Create the Popout Component

The Popout component renders in a separate browser window. It will display panels that users can pop out from the main interface. Create `src/app/features/popout/popout.component.ts`:

```
// src/app/features/popout/popout.component.ts

// VERSION 1 (Section 103) - Placeholder component // This will be replaced with the full popout implementation in Phase 3B



import { Component } from '@angular/core';



@Component({ selector: 'app-popout', template: `<div class="popout-container"> <h1>Popout Window</h1> <p>This component renders in a separate browser window.</p> <p>It will display charts and panels that communicate with the main window.</p> </div>`,

styles: [`.popout-container { padding: 1rem; }`,

`h1 { color: #1976d2; margin-bottom: 1rem; }`,

`p { color: #666; margin-bottom: 0.5rem; }`]

}) export class PopoutComponent {}
```

Delete the `.gitkeep` file:

```
$ rm src/app/features/popout/.gitkeep
```

Step 103.4: Create the Routing Module

Angular 13 uses a routing module to configure routes. Create `src/app/app-routing.module.ts`:

```
// src/app/app-routing.module.ts

// VERSION 1 (Section 103) - Basic routing configuration



import { NgModule } from '@angular/core'; import { RouterModule, Routes } from
'@angular/router';



import { HomeComponent } from './features/home/home.component'; import
{ DiscoverComponent } from './features/discover/discover.component'; import
{ PopoutComponent } from './features/popout/popout.component';



const routes: Routes = [ { path: '', redirectTo: 'home', pathMatch: 'full' }, { path:
'home', component: HomeComponent }, { path: 'discover', component:
DiscoverComponent }, { path: 'popout', component: PopoutComponent } ];



@NgModule({ imports: [RouterModule.forRoot(routes)], exports: [RouterModule] }) export
class AppRoutingModule {}
```

What each route does:

Path	Component	Purpose
`''`	(redirect)	Redirects root URL to `/home`
`home`	HomeComponent	Landing page
`discover`	DiscoverComponent	Vehicle search (placeholder)
`popout`	PopoutComponent	Pop-out window content

Why `pathMatch: 'full'`?

Without `pathMatch: 'full'`, the empty path `''` would match every URL (since every URL starts with nothing). The `pathMatch: 'full'` option ensures the redirect only triggers when the entire URL path is empty.

Step 103.5: Update AppModule

Import the routing module and declare the new components. Open `src/app/app.module.ts` and replace its contents with:

```
// src/app/app.module.ts

// VERSION 2 (Section 103) - With routing and feature components // Replaces VERSION 1 from Section 101



import { NgModule } from '@angular/core'; import { BrowserModule } from '@angular/platform-browser';



import { AppRoutingModule } from './app-routing.module'; import { AppComponent } from './app.component'; import { HomeComponent } from './features/home/home.component'; import { DiscoverComponent } from './features/discover/discover.component'; import { PopoutComponent } from './features/popout/popout.component';



@NgModule({ declarations: [ AppComponent, HomeComponent, DiscoverComponent, PopoutComponent ], imports: [ BrowserModule, AppRoutingModule ], providers: [], bootstrap: [AppComponent] }) export class AppModule {}
```

What changed:

Before	After
Only `AppComponent` declared	Four components declared
Only `BrowserModule` imported	`AppRoutingModule` also imported

Why declare components in AppModule?

In Angular 13 with NgModules, every component must be declared in exactly one module. Since we don't have feature modules yet, all components go in `AppModule`. Later, as the application grows, we might create `HomeModule`, `DiscoverModule`, etc.

Step 103.6: Add Router Outlet to AppComponent

The Router needs a place to render components. Update `src/app/app.component.ts` to add `<router-outlet>` and replace `href` with `routerLink`:

```
// src/app/app.component.ts

// VERSION 3 (Section 103) - With router outlet and routerLink // Replaces VERSION 2
from Section 102



import { Component } from '@angular/core';



@Component({ selector: 'app-root', template: <header class="app-header"> <div
class="app-header-brand"> <span class="app-header-logo">🚗</span> <span
class="app-header-title">vvroom</span> </div> <nav class="app-header-nav"> <a
class="nav-link" routerLink="/home" routerLinkActive="nav-link-active">Home</
a> <a class="nav-link" routerLink="/discover" routerLinkActive="nav-link-
active">Discover</a> </nav> </header> <main class="app-content"> <router-
outlet></router-outlet> </main>

,

styles: [ :host { display: flex; flex-direction: column; min-height: 100vh; }

.app-header { display: flex; align-items: center; justify-content: space-
between; padding: 0 1.5rem; height: 56px; background-color: #1976d2; color:
white; box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1); }

.app-header-brand { display: flex; align-items: center; gap: 0.5rem; }

.app-header-logo { font-size: 1.5rem; }

.app-header-title { font-size: 1.25rem; font-weight: 600; letter-spacing:
-0.5px; }

.app-header-nav { display: flex; gap: 0.5rem; }
```

```
.nav-link { padding: 0.5rem 1rem; color: rgba(255, 255, 255, 0.9); text-
decoration: none; border-radius: 4px; transition: background-color 0.2s; }

.nav-link:hover { background-color: rgba(255, 255, 255, 0.1); text-decoration:
none; }

.nav-link-active { background-color: rgba(255, 255, 255, 0.2); color: white; }

.app-content { flex: 1; padding: 1.5rem; }

]
}) export class AppComponent {}
```

What changed from VERSION 2:

Before	After
`href="/home"`	`routerLink="/home"`
`href="/discover"`	`routerLink="/discover"`
Static placeholder text	`<router-outlet></router-outlet>`
No active link style	`.nav-link-active` class with `routerLinkActive`

Understanding the new directives:

Directive	Purpose
`routerLink="/home"`	Navigate to `/home` without page reload
`routerLinkActive="nav-link-active"`	Add CSS class when this route is active
`<router-outlet>`	Placeholder where routed components render

Why `routerLinkActive`?

Users need visual feedback about their current location. The `routerLinkActive` directive automatically adds a CSS class when the link's route matches the current URL. When you navigate to `/home`, the Home link gets the `nav-link-active` class, giving it a slightly different background.

Step 103.7: Understanding Router Outlet

The `<router-outlet>` is a placeholder directive. When the URL changes, Angular:

- Matches the URL against the routes array
- Creates an instance of the matching component
- Inserts the component's view after `<router-outlet>`

Visual representation:

When URL is `/home`:

```
<main class="app-content">
<router-outlet></router-outlet> <app-home> <div class="home-container"> <h1>Welcome to Vvroom</h1> ... </div> </app-home> </main>
```

When URL is `/discover`:

```
<main class="app-content">
<router-outlet></router-outlet> <app-discover> <div class="discover-container"> <h1>Discover Vehicles</h1> ... </div> </app-discover> </main>
```

The component renders as a sibling *after* the outlet, not inside it. This is why we styled `.app-content` rather than `router-outlet` — the outlet itself has no dimensions.

The Aha Moment

Routes are the skeleton. The URL is where state lives.

Right now, our routes are simple: `/home`, `/discover`, `/popout`. But watch what happens when you click the navigation links:

- The URL in your browser's address bar changes
- The page does *not* reload
- The content area updates instantly

This is the foundation of URL-First architecture. In later sections, we'll add query parameters to the URL:

```
/discover?make=Toyota&year=2023&page=2
```

Every piece of state — the selected make, the year filter, the current page — will be encoded in the URL. Users can:

- Bookmark their exact search
- Share the URL with colleagues
- Use the browser's back button to undo filter changes
- Refresh without losing their place

The URL becomes a serialized snapshot of application state. This section establishes the routing skeleton that makes all of this possible.

Verification

1. Build the Application

```
$ cd ~/projects/vvroom
$ ng build
```

Expected output (no errors):

```
✔ Browser application bundle generation complete.

✔ Copying assets complete. ✔ Index html generation complete.



Initial Chunk Files           | Names         |  Raw Size
main.js                       | main          |  52.18 kB |
polyfills.js                  | polyfills     | 339.16 kB |
runtime.js                    | runtime       |   6.54 kB |
styles.css                    | styles        |   1.23 kB |



Build at: 2026-02-09T17:00:00.000Z - Hash: abc123 - Time: 5000ms
```

Note: `main.js` is slightly larger now because it includes the Router and three new components.

2. Serve the Application

```
$ ng serve --open
```

Browser opens to `http://localhost:4200`

3. Verify Redirect

When you first load `http://localhost:4200`, you should be redirected to `http://localhost:4200/home`. Check the URL bar — it should show `/home`, not just `/`.

4. Test Navigation

Click "Discover" in the navigation:

- URL changes to `http://localhost:4200/discover`
- Page does NOT reload (no white flash)
- Content area shows "Discover Vehicles" heading
- "Discover" link is highlighted (slightly brighter background)

Click "Home":

- URL changes to `http://localhost:4200/home`
- Content shows "Welcome to Vvroom"
- "Home" link is highlighted

5. Test Browser History

- Navigate to Home
- Navigate to Discover
- Click the browser's back button

Expected: You return to Home without a page reload. The URL updates and the content changes instantly.

6. Test Direct URL Access

Open a new browser tab and navigate directly to:

- `http://localhost:4200/discover`

Expected: The Discover page loads directly, with the navigation showing Discover as active.

7. Test Popout Route

Navigate to `http://localhost:4200/popout` by typing it in the address bar:

Expected: The Popout placeholder component renders.

8. Inspect Active Link Styling

Open browser developer tools (F12), navigate to Home, and inspect the Home link:

```
<a class="nav-link nav-link-active" ...>Home</a>
```

The `nav-link-active` class should be present.

Common Problems

Symptom	Cause	Solution
`ng build` fails with "Cannot find module './features/home/home.component'"	Component file not created or wrong path	Verify file exists at `src/app/features/home/home.component.ts`
"Error: No component factory found for HomeComponent"	Component not declared in AppModule	Add `HomeComponent` to the `declarations` array
Links still cause full page reload	Still using `href` instead of `routerLink`	Replace `href="/home"` with `routerLink="/home"`
"Error: Cannot find primary outlet to load 'HomeComponent'"	Missing `<router-outlet>` in template	Add `<router-outlet></router-outlet>` to AppComponent template
Active link style not showing	Typo in `routerLinkActive` class name	Ensure the class name matches: `routerLinkActive="nav-link-active"` and `.nav-link-active` in styles
Navigating to `/` shows blank content	Redirect not working	Ensure route has `pathMatch: 'full'` on the redirect
"NullInjectorError: No provider for Router"	`AppRoutingModule` not imported	Add `AppRoutingModule` to the `imports` array in `AppModule`

Key Takeaways

- **`routerLink` replaces `href`** — For SPA navigation without page reloads
- **`<router-outlet>` is a placeholder** — Routed components render after it, not inside it
- **`routerLinkActive` provides visual feedback** — Automatically adds a CSS class when the route matches

---

Acceptance Criteria

- [ ] `src/app/features/home/home.component.ts` exists with placeholder content

- [ ] `src/app/features/discover/discover.component.ts` exists with placeholder content
- [ ] `src/app/features/popout/popout.component.ts` exists with placeholder content
- [ ] `src/app/app-routing.module.ts` configures routes for home, discover, and popout
- [ ] Root path (`/`) redirects to `/home`
- [ ] `src/app/app.module.ts` declares all components and imports `AppRoutingModule`
- [ ] `AppComponent` uses `routerLink` instead of `href`
- [ ] `AppComponent` uses `routerLinkActive` for current route highlighting
- [ ] `AppComponent` includes `<router-outlet>` in the template
- [ ] Navigation works without page reload
- [ ] Browser back/forward buttons work correctly
- [ ] Active link has visual distinction (`.nav-link-active` style applied)
- [ ] `ng build` completes with no errors
- [ ] `ng serve` shows working navigation
- [ ] Pop-out button on Discover page opens `/popout` in a new window

---

What We Accomplished

Item	Before	After
Routes	None	Home, Discover, Popout configured
Navigation	`href` (full reload)	`routerLink` (SPA navigation)
Active state	None	Visual highlight on current route
Content area	Static text	Dynamic `<router-outlet>`
Components	1 (AppComponent)	4 (App, Home, Discover, Popout)
URL behavior	Ignored	Foundation for state management

Architecture Note

All three feature components (Home, Discover, Popout) are **framework components** — they exist regardless of the domain. When we add automobile-specific functionality later:

- `HomeComponent` will remain a generic landing page
- `DiscoverComponent` will become `AutomobileDiscoverComponent` (domain-specific)
- `PopoutComponent` will remain generic (renders any domain's panels)

This separation follows the naming conventions in `053-naming-conventions.md`.

Bonus: Pop-Outs Are Just Routes in New Windows

Before moving on, let's demonstrate something important: a pop-out window is nothing more than a route opened in a separate browser window. No special framework features are required — just `window.open()`.

Step 103.8: Add a Pop-Out Button to Discover

Update `src/app/features/discover/discover.component.ts` to add a button that opens the popout route in a new window:

```
// src/app/features/discover/discover.component.ts

// VERSION 2 (Section 103) - With pop-out button // Replaces VERSION 1 from earlier in this section



import { Component } from '@angular/core';



@Component({ selector: 'app-discover',
template:  <div class="discover-container"> <h1>Discover Vehicles</h1> <p>Vehicle search and filtering will appear here.</p> <p class="hint">Watch the URL bar as you navigate — this is where state will live.</p>

<div class="popout-demo"> <h2>Pop-Out Demo</h2> <p>Click the button below to open the popout route in a new window:</p> <button class="popout-button" (click)="openPopout()"> Open Pop-Out Window </button> <p class="note"> Notice: the pop-out is just <code>/popout</code> opened in a new window. Same Angular app, same routes, different window. </p> </div> </div>

,

styles: [  .discover-container { max-width: 800px; margin: 2rem auto; }

h1 { color: #1976d2; margin-bottom: 1rem; }

h2 { color: #1976d2; margin-bottom: 0.5rem; font-size: 1.25rem; }

p { color: #666; margin-bottom: 0.5rem; }

.hint { margin-top: 2rem; padding: 1rem; background-color: #e3f2fd; border-left: 4px solid #1976d2; color: #1565c0; }
```

```
.popout-demo { margin-top: 2rem; padding: 1.5rem; background-color: #f5f5f5;
border-radius: 8px; }

.popout-button { padding: 0.75rem 1.5rem; font-size: 1rem; background-color:
#1976d2; color: white; border: none; border-radius: 4px; cursor: pointer;
transition: background-color 0.2s; }

.popout-button:hover { background-color: #1565c0; }

.note { margin-top: 1rem; font-size: 0.875rem; color: #888; }

code { background-color: #e8e8e8; padding: 0.125rem 0.375rem; border-radius:
3px; font-family: monospace; }

]

}) export class DiscoverComponent { openPopout(): void { const popoutUrl = `$
{window.location.origin}/popout`; window.open( popoutUrl, 'vvroomPopout',
'width=600,height=400,menubar=no,toolbar=no,location=no,status=no' ); } }
```

What the `openPopout()` method does:

Line	Purpose
`window.location.origin`	Gets the base URL (e.g., `http://localhost:4200`)
`'/popout'`	The route we defined earlier
`'vvroomPopout'`	Window name (reuses same window if already open)
`'width=600,height=400,...'`	Window features (size, no browser chrome)

Verification: Test the Pop-Out

- Navigate to `http://localhost:4200/discover`
- Click the "Open Pop-Out Window" button
- A new browser window opens showing the Popout component
- The URL in the new window is `http://localhost:4200/popout`

Key observation: The pop-out window runs the same Angular application. It just happens to be displaying a different route in a separate window.

Why This Matters

This simple demonstration reveals the core insight behind pop-out windows:

> **Pop-outs are not special. They're just routes rendered in separate windows.**

In Phase 3B (documents 307-308), we'll add services that enable the main window and pop-out windows to communicate. But the foundation is what you see here: `window.open()` pointing to a route in your application.

When you understand this, the complexity of pop-out panels dissolves. A "pop-out chart" is just:

- A route that renders a chart component
- Opened in a new window
- With a service that syncs state between windows

Steps 1 and 2 are already working. Step 3 comes later.

Next Step

Proceed to `104-environment-config.md` to configure environment settings and verify API connectivity.

104: Environment Config Verification

Status: Planning **Depends On:** 103-routing **Blocks:** 150-typescript-generics-primer

---

Learning Objectives

After completing this section, you will:

- Understand how Angular's environment files enable build-time configuration
- Know how to verify your project compiles and runs correctly
- Recognize that Phase 1 establishes the foundation for all subsequent work

---

Objective

Verify the environment configuration from Section 101 and confirm the Phase 1 foundation is complete. This section is a consolidation checkpoint — ensuring everything from Sections 101-103 works together before proceeding to framework development.

Why

Checkpoints Prevent Cascading Errors

Before building framework services and components, we must confirm the foundation is solid. A misconfigured environment or broken route will cause confusing errors later. It's easier to diagnose problems in isolation than to debug a broken service that depends on three other broken things.

The Phase 1 Checkpoint

Document 002 (Dissection Rubric) defines this checkpoint:

> **Phase 1 Checkpoint:** Navigation between empty pages works. Browser history works. Readers observe URL changes in dev tools.

This section verifies we've achieved that checkpoint.

What

Step 104.1: Verify Environment Files

The environment files should already exist from Section 101. Confirm their contents.

Open `src/environments/environment.ts` and verify it contains:

```
// src/environments/environment.ts

export const environment = { production: false, apiBaseUrl: 'http://generic-
prime.minilab/api/specs/v1' };
```

Open `src/environments/environment.prod.ts` and verify it contains:

```
// src/environments/environment.prod.ts

export const environment = { production: true, apiBaseUrl: 'http://generic-
prime.minilab/api/specs/v1' };
```

If these files don't match, update them now. The `apiBaseUrl` property will be used by services in Phase 3.

Step 104.2: Understand Build-Time File Replacement

Angular's build system replaces environment files at build time. This is configured in `angular.json`:

```
"configurations": {

"production": { "fileReplacements": [ { "replace": "src/environments/environment.ts",
"with": "src/environments/environment.prod.ts" } ] } }
```

When you run `ng build --configuration=production`, Angular swaps `environment.ts` for `environment.prod.ts`. Your code always imports from `environment.ts`, but the actual values come from the appropriate file.

This is not runtime configuration. The values are baked into the compiled JavaScript bundle. To change the API URL, you must rebuild.

Step 104.3: Verify Project Structure

Run the following command to confirm the directory structure from Section 101:

```
$ cd ~/projects/vvroom

$ find src/app -type d | sort
```

Expected output:

```
src/app

src/app/domain-config src/app/domain-config/automobile src/app/domain-config/
automobile/adapters src/app/domain-config/automobile/chart-sources src/app/domain-
config/automobile/configs src/app/domain-config/automobile/models src/app/features
src/app/features/discover src/app/features/home src/app/features/popout src/app/
framework src/app/framework/components src/app/framework/models src/app/framework/
services src/app/framework/tokens
```

If directories are missing, create them using the commands from Section 101.

Step 104.4: Verify Component Files

Confirm the feature components from Section 103 exist:

```
$ ls -la src/app/features/*/
```

Expected output:

```
src/app/features/discover/:
discover.component.ts


src/app/features/home/: home.component.ts


src/app/features/popout/: popout.component.ts
```

Step 104.5: Build and Serve

Run the build to confirm everything compiles:

```
$ cd ~/projects/vvroom
$ ng build
```

Expected: Build succeeds with no errors.

Start the development server:

```
$ ng serve
```

Open your browser to `http://localhost:4200`.

Step 104.6: Execute Phase 1 Checkpoint Tests

Perform each test manually:

Test 1: Root Redirect

- Navigate to `http://localhost:4200`
- Observe: URL changes to `http://localhost:4200/home`
- Observe: Home component content displays

Test 2: Navigation Without Reload

- Click "Discover" in the navigation
- Observe: URL changes to `/discover`
- Observe: Page does NOT reload (no white flash)
- Observe: Discover component content displays

Test 3: Active Link Highlighting

- While on `/discover`, observe: "Discover" link has brighter background
- Click "Home"
- Observe: "Home" link now has brighter background

Test 4: Browser History

- Navigate: Home → Discover → Home → Discover
- Click browser back button
- Observe: Returns to Home without reload
- Click browser forward button
- Observe: Returns to Discover without reload

Test 5: Direct URL Access

- Open new tab
- Navigate directly to `http://localhost:4200/discover`
- Observe: Discover page loads correctly
- Observe: "Discover" link is highlighted

Test 6: Popout Route

- Navigate to `http://localhost:4200/popout`
- Observe: Popout component content displays

Phase 1 Complete

You have successfully completed Phase 1: Foundation. Your application now has:

Capability	Status
Clean project structure	Ready for framework and domain code
Environment configuration	API URL configured for both dev and prod
App shell with navigation	Header, content area, flexbox layout
Angular Router	Routes for Home, Discover, Popout
SPA navigation	No page reloads when clicking links
Browser history integration	Back/forward buttons work
Active route highlighting	Visual feedback for current location

The Phase 1 Aha Moment

Routes are the skeleton. The URL is where state lives.

Right now, your URLs are simple paths: `/home`, `/discover`, `/popout`. In Phase 3, we'll add services that read and write query parameters:

```
/discover?make=Toyota&year=2023&bodyClass=SUV&page=2
```

Every filter selection, every page navigation, every user choice will be encoded in the URL. This is URL-First State Management — and Phase 1 established the routing skeleton that makes it possible.

Verification

All verification is covered in Step 104.6. If any test fails, review the relevant section:

Failed Test	Review Section
Root redirect	103, Step 103.4 (routes)
Navigation reloads page	103, Step 103.6 (routerLink)
Active link not highlighted	103, Step 103.6 (routerLinkActive)
Browser history broken	103, Step 103.5 (AppRoutingModule)
Direct URL access fails	103, Step 103.4 (routes)
Build errors	101, 102, 103 (various)

Common Problems

Symptom	Cause	Solution
Build fails with environment error	`apiBaseUrl` not added to environment files	Update both environment files per Step 104.1
Missing directories	Section 101 steps skipped	Run mkdir commands from Section 101
Component files missing	Section 103 steps skipped	Create components per Section 103
Tests pass locally but not described here	You're ahead of the manuscript	Great! Continue to the next section

---

Key Takeaways

- **Environment files are build-time configuration** — Values are baked into the bundle, not loaded at runtime
- **Checkpoints prevent cascading errors** — Verify the foundation before building on it
- **Phase 1 establishes the routing skeleton** — All URL-First state management builds on this foundation

---

Acceptance Criteria

- [ ] `src/environments/environment.ts` contains `apiBaseUrl`
- [ ] `src/environments/environment.prod.ts` contains `apiBaseUrl`
- [ ] Directory structure matches expected layout
- [ ] All three feature components exist (Home, Discover, Popout)
- [ ] `ng build` succeeds with no errors
- [ ] `ng serve` starts without errors
- [ ] Root URL redirects to `/home`
- [ ] Navigation works without page reload
- [ ] Active links are visually highlighted
- [ ] Browser back/forward buttons work
- [ ] Direct URL access works for all routes

What We Accomplished

Phase 1 Document	Content
101: Project Cleanup	Removed boilerplate, created directories, configured environment
102: App Shell	Built header, navigation, content area with flexbox layout
103: Routing	Configured Router, created components, added routerLink
104: Verification	Confirmed all pieces work together

Next Step

Proceed to `150-typescript-generics-primer.md` (Interlude A) to build foundational understanding of TypeScript generics before tackling the framework models in Phase 2.

150: TypeScript Generics Primer

Status: Planning **Depends On:** 104-environment-config **Blocks:** 201-domain-config-interface

---

Learning Objectives

After completing this section, you will:

- Understand why generics exist and what problem they solve
- Be able to read and interpret generic type signatures
- Know the difference between concrete types and type parameters

---

Objective

Build foundational understanding of TypeScript generics before tackling the framework models in Phase 2. This is a teaching interlude, not a code implementation section — you won't modify the vvroom project here.

Why This Interlude Exists

Phase 2 introduces interfaces like this:

```
export interface DomainConfig<TFilters, TData, TStatistics> {

urlMapper: IFilterUrlMapper<TFilters>; apiAdapter: IApiAdapter<TFilters, TData,
TStatistics>; tableConfig: TableConfig<TData>; // ... }
```

If you've never worked with generics, this looks intimidating. What are `TFilters`, `TData`, and `TStatistics`? Why are there angle brackets everywhere?

This primer answers those questions. By the end, you'll read generic signatures as fluently as regular TypeScript.

What Are Generics?

The Problem: Type Safety vs Code Reuse

Consider a function that returns the first element of an array:

```
// Without generics - loses type information
function getFirst(arr: any[]): any { return arr[0]; }


const numbers = [1, 2, 3]; const first = getFirst(numbers);  // first is 'any', not
'number'
```

The `any` type works, but we've lost type safety. TypeScript can't help us if we try to call `first.toUpperCase()` on a number.

We could write separate functions for each type:

```
// Type-safe but repetitive
function getFirstNumber(arr: number[]): number { return arr[0]; }


function getFirstString(arr: string[]): string { return arr[0]; }


function getFirstVehicle(arr: Vehicle[]): Vehicle { return arr[0]; }
```

This is type-safe, but we've duplicated the same logic three times. What if we need 50 types?

The Solution: Generics

Generics let us write one function that works with any type while preserving type information:

```
// With generics - type-safe AND reusable
function getFirst<T>(arr: T[]): T { return arr[0]; }


const numbers = [1, 2, 3]; const first = getFirst(numbers);  // first is 'number'


const names = ['Alice', 'Bob']; const name = getFirst(names);     // name is 'string'


const vehicles: Vehicle[] = [/ ... /]; const vehicle = getFirst(vehicles); // vehicle
is 'Vehicle'
```

The `<T>` is a **type parameter**. When you call `getFirst(numbers)`, TypeScript infers that `T` is `number`. The function returns `T`, so the return type is also `number`.

This is the Aha Moment: Generics give us type safety without code duplication.

Reading Generic Signatures

The Anatomy of a Generic Type

```
interface Container<T> {
value: T; getValue(): T; }
```

Part	Meaning
`Container`	The interface name
`<T>`	Type parameter declaration
`value: T`	Property of type T
`getValue(): T`	Method returning type T

The `T` is a placeholder. When you use `Container<string>`, every `T` becomes `string`:

```
const box: Container<string> = {
value: 'hello', getValue() { return this.value; } };


box.value;       // type is 'string' box.getValue();  // returns 'string'
```

Multiple Type Parameters

Generics can have multiple parameters:

```
interface Pair<K, V> {
key: K; value: V; }


const entry: Pair<string, number> = { key: 'age', value: 30 };
```

Here `K` becomes `string` and `V` becomes `number`.

Conventional Type Parameter Names

Name	Convention
`T`	General "Type"
`K`	Key type (in maps/objects)
`V`	Value type (in maps/objects)
`E`	Element type (in collections)
`R`	Return type

These are conventions, not requirements. In vvroom, we use descriptive names:

Name	Meaning in vvroom
`TFilters`	Filter model type (e.g., `AutomobileFilters`)
`TData`	Data model type (e.g., `VehicleResult`)
`TStats`	Statistics model type (e.g., `VehicleStatistics`)

The `T` prefix indicates "this is a Type parameter, not a concrete type."

Concrete Types vs Type Parameters

Concrete Types

A **concrete type** is a specific, known type:

```
// Concrete types - we know exactly what these are

let name: string; let count: number; let vehicle: Vehicle; let filters:
AutomobileFilters;
```

Type Parameters

A **type parameter** is a placeholder filled in later:

```
// Type parameters - placeholders

interface Container<T> {     // T is a parameter value: T; }



// Now we fill in the parameter with a concrete type const box: Container<string> =
{ value: 'hello' };
```

Think of type parameters like function parameters:

```
// Function parameter: x is filled in when called

function double(x: number): number { return x * 2; } double(5);  // x = 5



// Type parameter: T is filled in when used interface Container<T> { value: T; } const
box: Container<string> = { value: 'hello' };  // T = string
```

Built-in Generic Types

You've already used generics — TypeScript's built-in types use them extensively.

Array<T>

```
// These are equivalent

const numbers: number[] = [1, 2, 3]; const numbers: Array<number> = [1, 2, 3];
```

`Array<number>` means "an array where every element is a number."

Promise<T>

```
// A promise that resolves to a string

const greeting: Promise<string> = Promise.resolve('hello');



// A promise that resolves to a Vehicle const vehiclePromise: Promise<Vehicle> =
fetchVehicle(id);
```

`Promise<Vehicle>` means "a promise that, when resolved, gives you a Vehicle."

Map<K, V>

```
// A map from string keys to number values

const ages: Map<string, number> = new Map(); ages.set('Alice', 30); ages.set('Bob',
25);



ages.get('Alice');  // returns number | undefined
```

Partial<T>

```
interface Vehicle {

make: string; model: string; year: number; }



// All properties become optional const partial: Partial<Vehicle> = { make:
'Toyota' // model and year are optional };
```

`Partial<Vehicle>` creates a new type where all properties of `Vehicle` are optional. This is useful for update operations where you only change some fields.

Generic Constraints

Sometimes you need to limit what types can be used as a parameter.

The Problem

```
function getLength<T>(item: T): number {

return item.length;  // Error: Property 'length' does not exist on type 'T' }
```

TypeScript doesn't know that `T` has a `length` property. What if someone passes a number?

The Solution: extends

```
interface HasLength {

length: number; }



function getLength<T extends HasLength>(item: T): number { return item.length;  // Now
TypeScript knows T has 'length' }



getLength('hello');     // OK - strings have length getLength([1, 2, 3]);   // OK -
arrays have length getLength(42);          // Error - numbers don't have length
```

The `extends` keyword constrains `T` to types that have a `length` property.

Constraints in vvroom

In Phase 2, you'll see constraints like:

```
interface IApiAdapter<TFilters, TData, TStatistics = any> {

fetchData(filters: TFilters): Observable<ApiResponse<TData>>; // ... }
```

The `= any` provides a default type. If you don't specify `TStatistics`, it defaults to `any`.

Applying Generics: A Preview of Phase 2

Here's how vvroom uses generics. Don't memorize this — just recognize the patterns.

The DomainConfig Interface

```
export interface DomainConfig<TFilters, TData, TStatistics> {

// Identity domainKey: string; displayName: string;



// Type references (so the framework knows what types to expect) filterModel:
Type<TFilters>; dataModel: Type<TData>;



// Adapters (these use the same type parameters) apiAdapter: IApiAdapter<TFilters,
TData, TStatistics>; urlMapper: IFilterUrlMapper<TFilters>;



// UI configuration tableConfig: TableConfig<TData>; }
```

Reading this signature:

- `DomainConfig` takes three type parameters
- `TFilters` flows through to `urlMapper` and `apiAdapter`
- `TData` flows through to `apiAdapter` and `tableConfig`
- `TStatistics` flows through to `apiAdapter`

Using the Interface

When we create the automobile domain config:

```
const AUTOMOBILE_CONFIG: DomainConfig<

AutomobileFilters,    // TFilters = AutomobileFilters VehicleResult,        // TData =
VehicleResult VehicleStatistics     // TStatistics = VehicleStatistics > =
{ domainKey: 'automobile', displayName: 'Automobiles', filterModel: AutomobileFilters,
dataModel: VehicleResult, apiAdapter: new AutomobileApiAdapter(), urlMapper: new
AutomobileUrlMapper(), tableConfig: automobileTableConfig, };
```

Now TypeScript knows:

- `urlMapper` must accept `AutomobileFilters`
- `tableConfig` must work with `VehicleResult`
- `apiAdapter` must return `VehicleStatistics`

If any of these are wrong, TypeScript catches the error at compile time.

Practice Exercises

Try these in your editor or the TypeScript Playground (https://www.typescriptlang.org/play).

Exercise 1: Read the Signature

What type is `result` in each case?

```
function identity<T>(value: T): T {

return value; }



const result1 = identity('hello'); const result2 = identity(42); const result3 =
identity({ name: 'Alice' });
```

<details> <summary>Answer</summary>

- `result1` is `string`
- `result2` is `number`
- `result3` is `{ name: string }`

TypeScript infers `T` from the argument type. </details>

Exercise 2: Write a Generic Function

Write a function `wrapInArray` that takes any value and returns it wrapped in an array.

```
// Your code here


wrapInArray('hello');  // should be ['hello'] with type string[]
wrapInArray(42);       // should be [42] with type number[]
```

<details> <summary>Answer</summary>

```
function wrapInArray<T>(value: T): T[] {

return [value]; }
```

</details>

Exercise 3: Multiple Type Parameters

What are the types of `key` and `value`?

```
interface Entry<K, V> {

key: K; value: V; }



const entry: Entry<string, number> = { key: 'count', value: 42 };



const { key, value } = entry;
```

<details> <summary>Answer</summary>

- `key` is `string` (K = string)

- `value` is `number` (V = number)

</details>

Common Mistakes

Mistake 1: Forgetting Type Parameters

```
// Wrong - DomainConfig needs type parameters

const config: DomainConfig = { / ... / };



// Right const config: DomainConfig<AutomobileFilters, VehicleResult,
VehicleStatistics> = { / ... / };
```

Mistake 2: Mismatched Types

```
interface Container<T> {

value: T; }



// Wrong - value type doesn't match const box: Container<string> = { value: 42  //
Error: number is not assignable to string };
```

Mistake 3: Confusing Type Parameters with Values

```
// Wrong - TFilters is a type, not a value
function processFilters<TFilters>(filters: TFilters) { console.log(TFilters);  // Error: 'TFilters' only refers to a type }


// Right - use the parameter, not the type function processFilters<TFilters>(filters: TFilters) { console.log(filters);  // OK }
```

Key Takeaways

- **Generics provide type safety without code duplication** — Write once, use with any type
- **Type parameters are placeholders** — They're filled in when you use the generic type
- **T-prefixed names indicate type parameters** — `TFilters`, `TData`, `TStats` are conventions, not requirements

---

The Aha Moment

Generics give us type safety without code duplication.

In Phase 2, you'll create interfaces with generic parameters. These interfaces define contracts that work with *any* domain:

- `DomainConfig<TFilters, TData, TStats>` works for automobiles, real estate, or any future domain
- The framework code uses these interfaces without knowing the specific types
- Domain-specific code provides concrete types (`AutomobileFilters`, `VehicleResult`)
- TypeScript ensures everything matches up at compile time

Without generics, we'd either:

- Lose type safety (use `any` everywhere)
- Duplicate code for each domain

Generics give us both type safety and reusability. That's why they're foundational to vvroom's architecture.

Acceptance Criteria

This is a teaching section with no code changes. Criteria are conceptual:

- [ ] You can explain what problem generics solve
- [ ] You can read `Interface<T, U>` and understand T and U are type parameters
- [ ] You can identify the difference between `string` (concrete type) and `T` (type parameter)
- [ ] You can explain what `DomainConfig<TFilters, TData, TStats>` means conceptually

Next Step

Proceed to `201-domain-config-interface.md` to create the central configuration interface using the generic patterns you just learned.

201: Domain Config Interface

Status: Planning **Depends On:** 150-typescript-generics-primer **Blocks:** 202-resource-management-interface, 304-domain-config-registry

---

Learning Objectives

After completing this section, you will:

- Understand how a single interface can define the contract for an entire domain's behavior
- Know how TypeScript generics enable type-safe domain configurations
- Recognize the relationship between configuration interfaces and framework services

---

Objective

Create the `DomainConfig` interface that serves as the central configuration schema for any domain in the vvroom application. This interface defines all adapters, UI configurations, and feature flags required for the framework to operate with domain-specific data.

Why

The `DomainConfig` interface is the cornerstone of vvroom's architecture. It answers a critical question: *How do we make the framework work with any domain (automobiles, real estate, inventory) without changing framework code?*

The answer is **configuration-driven design**. Instead of hard-coding domain-specific behavior into services and components, we define a configuration interface that each domain must implement. The framework reads this configuration and adapts its behavior accordingly.

This is the Phase 2 Aha Moment in action: TypeScript interfaces are executable documentation. The `DomainConfig` interface documents *exactly* what any domain must provide to work with the framework. TypeScript enforces this contract at compile time.

Benefits of Configuration-Driven Design

Approach	Adding a New Domain Requires
Hard-coded	Modifying framework services, components, routes
Configuration-driven	Creating one configuration file that implements DomainConfig

Angular Style Guide References

- Style 03-01: Use interfaces for type definitions
- Style 07-04: Create service interfaces for complex services

TypeScript Best Practices

This interface uses several TypeScript features you learned in the generics primer:

- **Generic type parameters** (`TFilters` , `TData` , `TStatistics`) — Allow type-safe domain configuration
- `Type<T>` — Angular's type for class constructors, enabling runtime instantiation
- **Optional properties** — Features like `highlightFilters` and `metadata` are not required for every domain

---

What

Step 201.1: Create the Framework Models Directory Index

Before creating individual interface files, create a barrel export file that will aggregate all model exports.

Create the file `src/app/framework/models/index.ts` :

```
// src/app/framework/models/index.ts

// VERSION 1 (Section 201) - Barrel file for framework models



// This barrel file exports all framework model interfaces. // Import from '@app/
framework/models' instead of individual files.



export * from './domain-config.interface';
```

Delete the `.gitkeep` file since the directory now has real content:

```
$ rm src/app/framework/models/.gitkeep
```

Why a barrel file?

Barrel files simplify imports throughout the application:

```
// Without barrel file (verbose):

import { DomainConfig } from './framework/models/domain-config.interface'; import
{ TableConfig } from './framework/models/table-config.interface';



// With barrel file (clean): import { DomainConfig, TableConfig } from './framework/
models';
```

We'll add more exports as we create additional interface files.

Step 201.2: Create the Domain Config Interface

Create the file `src/app/framework/models/domain-config.interface.ts`:

```
// src/app/framework/models/domain-config.interface.ts
// VERSION 1 (Section 201) - Core domain configuration interface

import { Type } from '@angular/core';

/**
  • Domain configuration interface
 *
  • Complete configuration schema for a domain-specific implementation.

  • Defines all adapters, UI configurations, and feature flags required

  • for the framework to operate with domain-specific data.
 *
  • @template TFilters - Domain-specific filter model type

  • @template TData - Domain-specific data model type

  • @template TStatistics - Domain-specific statistics model type (optional)
 *
  • @example
```

typescript

- const AUTOMOBILE_DOMAIN_CONFIG: DomainConfig<
- AutomobileFilters,
- VehicleResult,
- VehicleStatistics
- > = {
- domainName: 'automobile',
- domainLabel: 'Automobile Discovery',
- apiBaseUrl: 'http://auto-discovery.minilab/api/v1',
- filterModel: AutomobileFilters,
- dataModel: VehicleResult,
- statisticsModel: VehicleStatistics,
- apiAdapter: new AutomobileApiAdapter(),
- urlMapper: new AutomobileUrlMapper(),
- cacheKeyBuilder: new AutomobileCacheKeyBuilder(),
- tableConfig: AUTOMOBILE_TABLE_CONFIG,
- pickers: AUTOMOBILE_PICKER_CONFIGS,
- filters: AUTOMOBILE_FILTER_DEFINITIONS,
- queryControlFilters: AUTOMOBILE_QUERY_CONTROL_FILTERS,
- charts: AUTOMOBILE_CHART_CONFIGS,
- features: {
- highlights: true,
- popOuts: true,
- rowExpansion: true
- }
- };

```
 /
export interface DomainConfig<TFilters, TData, TStatistics = any> { /**


  • Unique domain identifier (lowercase, no spaces)


  • Used for routing, storage keys, and internal identification

   *

  • @example 'automobile', 'real-estate', 'inventory'

   */
domainName: string;


/**


  • Human-readable domain label


  • Used for display in UI (page titles, navigation, etc.)

   *

  • @example 'Automobile Discovery', 'Real Estate Listings'

   */
domainLabel: string;


/**
```

```
   • Base URL for domain-specific API


   • All API requests will be prefixed with this URL

    *

   • @example 'http://auto-discovery.minilab/api/v1'

    */
apiBaseUrl: string;


/**


   • Filter model class/constructor


   • Used for type checking and instantiation

    */
filterModel: Type<TFilters>;


/**


   • Data model class/constructor


   • Used for type checking and instantiation

    */
```

```
dataModel: Type<TData>;


/**


  • Statistics model class/constructor (optional)


  • Used for type checking and instantiation

   */
statisticsModel?: Type<TStatistics>;


/**


  • API adapter for data fetching


  • Implements domain-specific API calls and response transformation

   */
apiAdapter: IApiAdapter<TFilters, TData, TStatistics>;


/**


  • URL mapper for filter serialization


  • Converts between filter objects and URL parameters

   */
```

```
urlMapper: IFilterUrlMapper<TFilters>;


/**


  • Cache key builder for request coordination


  • Generates unique cache keys from filter objects

   */
cacheKeyBuilder: ICacheKeyBuilder<TFilters>;


/**


  • Table configuration for main data display


  • Defines columns, pagination, sorting, etc.

   */
tableConfig: TableConfig<TData>;


/**


  • Picker configurations


  • Array of picker configs for multi-select data pickers

   */
```

```
pickers: PickerConfig<any>[];



/**



  • Inline filter definitions for results table



  • Defines inline filter controls displayed above the results table

   */
filters: FilterDefinition[];



/**



  • Query Control filter definitions



  • Defines filters available in the Query Control component dialogs

   */
queryControlFilters: QueryFilterDefinition<TFilters>[];



/**



  • Highlight filter definitions (optional)



  • Defines highlight filters available in the Query Control component

    These filters add h_ URL parameters for segmented statistics in charts
```

```
*/ highlightFilters?: QueryFilterDefinition<any>[];


/**


  • Chart configurations


  • Defines available charts and their data sources

   */
charts: ChartConfig[];


/**


  • Chart data sources map


  • Maps dataSourceId to ChartDataSource instances


  • Used by StatisticsPanelComponent to instantiate charts

   */
chartDataSources?: Record<string, any>;


/**


  • Feature flags
```

```
  • Controls which framework features are enabled for this domain

   */
features: DomainFeatures;



/**



  • Optional metadata



  • Additional domain-specific information

   */
metadata?: DomainMetadata;



/**



  • Default filters for the domain (optional)



  • These filters are applied when the application first loads or when filters are
    cleared.

   */
defaultFilters?: Partial<TFilters>; }



/**



  • Domain feature flags
```

```
 *
  • Controls which framework features are enabled/disabled for a specific domain
 */
export interface DomainFeatures { /**
  • Enable/disable highlight system
  • Allows users to highlight specific data subsets
   */
highlights: boolean;

/**
  • Enable/disable pop-out window system
  • Allows panels to be moved to separate browser windows
   */
popOuts: boolean;

/**
  • Enable/disable row expansion in tables
```

```
  • Allows rows to be expanded to show additional details

   */
rowExpansion: boolean;


/**

  • Enable/disable statistics panel

  • Shows aggregated statistics and charts

   */
statistics?: boolean;


/**

  • Enable/disable export functionality

  • Allows users to export data to CSV, Excel, etc.

   */
export?: boolean;


/**

  • Enable/disable column management
```

```
  • Allows users to show/hide and reorder columns

   */
columnManagement?: boolean;


/**

  • Enable/disable state persistence


  • Saves user preferences to local storage

   */
statePersistence?: boolean; }


/**

  • Domain metadata

 *

  • Additional information about the domain

 */
export interface DomainMetadata { /**

  • Domain version (semantic versioning)

   */
version?: string;
```

```
/**

  • Domain description

   */
description?: string;

/**

  • Domain author/maintainer

   */
author?: string;

/**

  • Creation date

   */
createdAt?: string;

/**

  • Last update date

   */
updatedAt?: string;
```

```
/**

  • Custom metadata fields

   */
[key: string]: any; }

/**

  • Filter format configuration

 *

  • Controls how filter values are displayed and processed.

 */
export interface FilterFormat { /**

  • Number formatting options

   */
number?: FilterNumberFormat;

/**

  • Date formatting options

   */
```

```
date?: FilterDateFormat;


/**


  • Whether string matching is case-sensitive


  • @default true

   */
caseSensitive?: boolean;


/**


  • Text transformation to apply before sending to API

   */
transform?: 'lowercase' | 'uppercase' | 'titlecase' | 'trim' | 'none';


/**


  • Custom formatting function for display

   */
displayFormatter?: (value: any) => string;


/**


  • Custom parsing function for input
```

```
   */
valueParser?: (input: string) => any; }


/**


  • Number formatting options

 *

  • Based on Intl.NumberFormat options for consistency with browser standards

 */
export interface FilterNumberFormat { /**


  • Use thousand separators (commas)


  • @default true

   */
useGrouping?: boolean;


/**


  • Minimum decimal places


  • @default 0

   */
```

```
minimumFractionDigits?: number;


/**


  • Maximum decimal places


  • @default 0 for integers, 2 for decimals

   */
maximumFractionDigits?: number;


/**


  • Locale for number formatting


  • @default 'en-US'

   */
locale?: string;


/**


  • Custom format pattern

   */
pattern?: string; }
```

```
/**


  • Date formatting options


 */
export interface FilterDateFormat { /**


  • Date format pattern


  • @default 'YYYY-MM-DD'


   */
pattern?: string;


/**


  • Locale for date formatting


  • @default 'en-US'


   */
locale?: string;


/**


  • Whether to show time component
```

```
  • @default false

   */
includeTime?: boolean; }


/**

  • Filter control types

 */
export type FilterType =
```

```

```

```

```

```

/**

  • Filter operators

 */
```

```
export type FilterOperator =
```

```
/**
```

- Filter option for select/multiselect

```
 */
export interface FilterOption { value: any; label: string; icon?: string; disabled?:
boolean; }
```

```
/**
```

```
  • Filter validation rules

 */
export interface FilterValidation { minLength?: number; maxLength?: number; pattern?:
RegExp | string; custom?: (value: any) => boolean | string; }



/**



  • Chart configuration interface

 */
export interface ChartConfig { id: string; title: string; type: ChartType;
dataSourceId: string; options?: any; height?: number; width?: number | string;
visible?: boolean; collapsible?: boolean; }



/**



  • Chart types

 */
export type ChartType =
```

```
/**
  • Configuration validation error
 */
export interface ConfigValidationError { type: ConfigErrorType; field: string;
message: string; expected?: string; actual?: any; }

/**
  • Configuration error types
 */
export enum ConfigErrorType { MISSING_REQUIRED = 'MISSING_REQUIRED', INVALID_TYPE =
'INVALID_TYPE', INVALID_VALUE = 'INVALID_VALUE', EMPTY_ARRAY = 'EMPTY_ARRAY',
DUPLICATE_ID = 'DUPLICATE_ID' }

/**
  • Configuration validation result
 */
```

```
export interface ConfigValidationResult { valid: boolean; errors:
ConfigValidationError[]; warnings?: ConfigValidationError[]; }



/**



  • Default domain features



  • Used when features object is not fully specified


 */

export const DEFAULT_DOMAIN_FEATURES: DomainFeatures = { highlights: true, popOuts:
true, rowExpansion: true, statistics: true, export: true, columnManagement: true,
statePersistence: true };



/**



  • Merge partial domain features with defaults


 */

export function mergeDomainFeatures( features: Partial<DomainFeatures> ):
DomainFeatures { return { ...DEFAULT_DOMAIN_FEATURES, ...features }; }



// Forward declarations for interfaces defined in other files // These will be
properly imported once those files are created



/**



  • API adapter interface (defined in resource-management.interface.ts)


 */
```

```
export interface IApiAdapter<TFilters, TData, TStatistics = any> { fetchData(filters:
TFilters, highlights?: any): any; }



/**



  • Filter URL mapper interface (defined in resource-management.interface.ts)


 */

export interface IFilterUrlMapper<TFilters> { toUrlParams(filters: TFilters): any;
fromUrlParams(params: any): TFilters; extractHighlights?(params: any): any; }



/**



  • Cache key builder interface (defined in resource-management.interface.ts)


 */

export interface ICacheKeyBuilder<TFilters> { buildKey(filters: TFilters, highlights?:
any): string; }



/**



  • Table config interface (defined in table-config.interface.ts)


 */

export interface TableConfig<T> { tableId: string; stateKey: string; columns: any[];
dataKey: keyof T; [key: string]: any; }



/**



  • Picker config interface (defined in picker-config.interface.ts)
```

```
 */
export interface PickerConfig<T> { id: string; displayName: string; columns: any[];
[key: string]: any; }



/**



  • Filter definition interface (defined in filter-definition.interface.ts)


 */
export interface FilterDefinition { id: string; label: string; type: FilterType; [key:
string]: any; }



/**



  • Query filter definition interface (defined in filter-definition.interface.ts)


 */
export interface QueryFilterDefinition<T> { field: keyof T; label: string; type:
string; [key: string]: any; }
```

Step 201.3: Understand the Interface Structure

The `DomainConfig` interface is organized into logical sections:

Section	Properties	Purpose
Identity	`domainName`, `domainLabel`, `apiBaseUrl`	Identify the domain and where to fetch data
Type Models	`filterModel`, `dataModel`, `statisticsModel`	Runtime type information for instantiation
Adapters	`apiAdapter`, `urlMapper`, `cacheKeyBuilder`	Domain-specific data transformation logic
UI Configuration	`tableConfig`, `pickers`, `filters`, `charts`	Define how data is displayed
Feature Flags	`features`	Enable/disable framework capabilities
Metadata	`metadata`, `defaultFilters`	Optional additional information

Why so many forward declarations?

The `DomainConfig` interface references types from other files we haven't created yet. The forward declarations at the bottom are temporary placeholders. As we create each interface file (documents 202-209), we'll update the imports to use the real definitions.

This is a common pattern when building a system in dependency order: you need to reference types that will exist later. TypeScript allows this through interface merging and forward declarations.

The Aha Moment

TypeScript interfaces are executable documentation.

Look at the `DomainConfig` interface. It's not just code — it's a specification. This interface documents *exactly* what any domain must provide to work with the vvroom framework:

- An identity (`domainName`, `domainLabel`)
- A data source (`apiBaseUrl`, `apiAdapter`)
- A way to serialize state to URLs (`urlMapper`)
- UI configurations (`tableConfig`, `filters`, `charts`)
- Feature toggles (`features`)

When you create a new domain (like agriculture or real estate), the compiler becomes your guide. Try to create a `DomainConfig<AgricultureFilters, CropResult>` object, and TypeScript will tell you exactly what's missing.

This is why we invested time in the Generics Primer (Section 150). The generic parameters `TFilters`, `TData`, and `TStatistics` ensure that when you wire up an automobile domain, the `urlMapper` accepts `AutomobileFilters`, and the `tableConfig` works with `VehicleResult`. Type mismatches are caught at compile time, not runtime.

The interface is the contract. TypeScript enforces it.

Verification

1. Check File Exists

```
$ ls -la src/app/framework/models/
```

Expected output shows both files:

```
total 16

drwxr-xr-x 2 user user 4096 Feb  9 12:00 . drwxr-xr-x 5 user user 4096 Feb  9 12:00 ..
-rw-r--r-- 1 user user  123 Feb  9 12:00 index.ts -rw-r--r-- 1 user user 8234 Feb  9
12:00 domain-config.interface.ts
```

2. TypeScript Compilation Check

```
$ cd ~/projects/vvroom

$ npx tsc --noEmit src/app/framework/models/domain-config.interface.ts
```

Expected: No output (no compilation errors).

3. Verify Exports

```
$ grep "export" src/app/framework/models/domain-config.interface.ts | head -5
```

Expected output shows multiple exports:

```
export interface DomainConfig<TFilters, TData, TStatistics = any> {

export interface DomainFeatures { export interface DomainMetadata { export interface
FilterFormat { export interface FilterNumberFormat {
```

Common Problems

Symptom	Cause	Solution
`Cannot find module '@angular/core'`	Angular not installed or wrong path	Run `npm install` in project root
`Type 'X' is not generic`	Forgot generic parameters	Add type parameters: `DomainConfig<TFilters, TData>`
`Property 'X' does not exist on type 'DomainConfig'`	Using wrong property name	Check interface definition for exact property names
Red squiggles on forward declarations	Expected — these are temporary	Will be resolved in subsequent documents
`Duplicate identifier 'FilterOption'`	Name collision with another file	Use unique names or namespaces

---

Key Takeaways

- **TypeScript interfaces are executable documentation** — The `DomainConfig` interface documents exactly what any domain must provide to work with the framework

- **Generic type parameters enable type safety** — `TFilters`, `TData`, and `TStatistics` ensure type-safe configuration
- **Configuration-driven design enables extensibility** — New domains require only configuration, not framework changes

---

Acceptance Criteria

- [ ] `src/app/framework/models/domain-config.interface.ts` exists with complete interface definition
- [ ] `src/app/framework/models/index.ts` exports the domain config interface
- [ ] Interface includes all required properties: `domainName`, `domainLabel`, `apiBaseUrl`, etc.
- [ ] Interface uses TypeScript generics for type-safe configuration
- [ ] `DomainFeatures` interface defines all feature flags
- [ ] `DomainMetadata` interface provides extensible metadata structure
- [ ] Helper function `mergeDomainFeatures` is implemented
- [ ] `DEFAULT_DOMAIN_FEATURES` constant is defined
- [ ] TypeScript compilation succeeds with no errors
- [ ] JSDoc comments document all public interfaces and properties

---

Next Step

Proceed to `202-resource-management-interface.md` to define the adapters that `DomainConfig` references.

202: Resource Management Interface

Status: Planning **Depends On:** 201-domain-config-interface **Blocks:** 301-url-state-service, 306-resource-management-service

---

Learning Objectives

After completing this section, you will:

- Understand the adapter pattern and why it isolates domain-specific logic from framework code
- Know how to define interfaces that different domains can implement
- Recognize the relationship between URL state, API requests, and caching

---

Objective

Create the resource management interfaces that define how domains interact with URLs, APIs, and caching. These interfaces establish the contract that domain-specific adapters must fulfill for the framework's URL-First architecture to function.

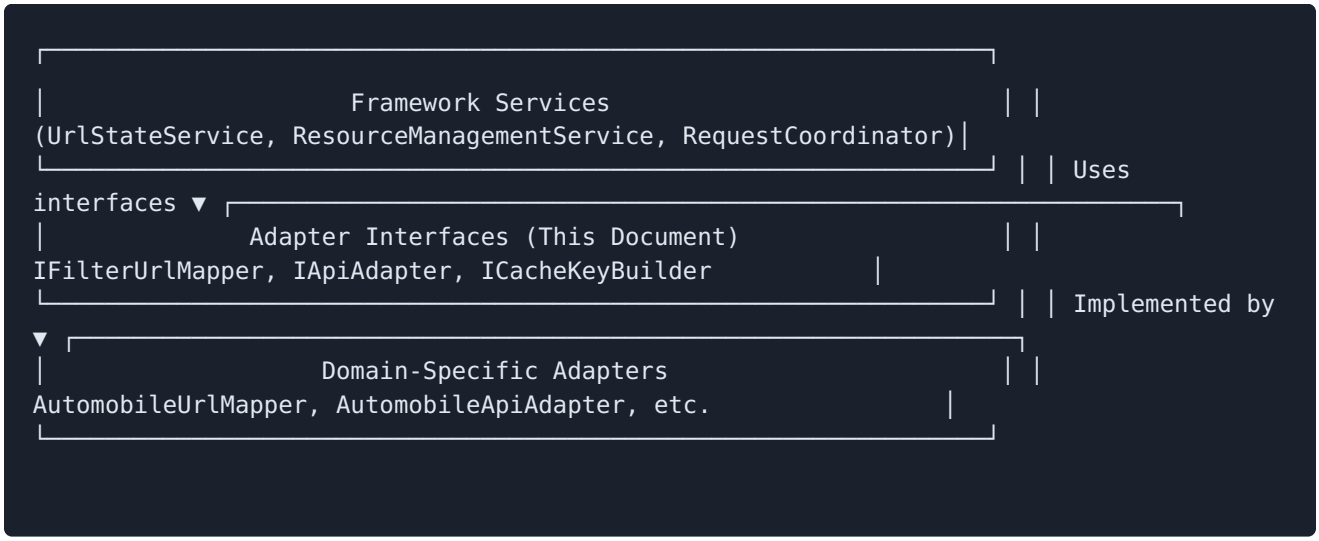
Why

The URL-First architecture requires three key operations:

- **URL <-> Filters**: Convert between URL query parameters and domain filter objects
- **Filters -> API**: Fetch data from the API based on current filters
- **Filters -> Cache Key**: Generate cache keys to prevent duplicate requests

Each domain (automobiles, real estate, inventory) has different filter structures and API endpoints. The resource management interfaces define the contract that each domain's adapters must implement.

This is the Adapter Pattern in action:



Framework services depend on *interfaces*, not concrete implementations. This means:

- Framework code never changes when adding new domains
- Each domain provides its own adapter implementations
- Type safety is maintained throughout

URL-First Architecture Reference

The resource management interfaces directly support the URL-First pattern (see `docs/README.md`):

- `IFilterUrlMapper` enables the URL to be the single source of truth
- `IApiAdapter` transforms URL-derived filters into API requests
- `ICacheKeyBuilder` prevents duplicate API calls when URL doesn't change

What

Step 202.1: Create the Resource Management Interface

Create the file `src/app/framework/models/resource-management.interface.ts`:

```
// src/app/framework/models/resource-management.interface.ts
// VERSION 1 (Section 202) - Adapter interfaces for URL-First architecture

import { Observable } from 'rxjs'; import { Params } from '@angular/router';

/**

  • Adapter for mapping filters to/from URL parameters
 *
  • This interface defines how a domain converts between its filter model

  • and URL query parameters. Implementing this interface enables the

  • URL-First architecture where the URL is the single source of truth.
 *
  • @template TFilters - The shape of the filter object
 *
  • @example

  •
```

typescript

- class AutomobileUrlMapper implements IFilterUrlMapper<AutomobileFilters> {
- toUrlParams(filters: AutomobileFilters): Params {
- const params: Params = {};
- if (filters.manufacturer?.length) {
- params['manufacturer'] = filters.manufacturer.join(',');
- }
- if (filters.yearMin) {
- params['yearMin'] = filters.yearMin.toString();
- }
- return params;
- }

*

- fromUrlParams(params: Params): AutomobileFilters {
- return {
- manufacturer: params['manufacturer']?.split(',') || [],
- yearMin: params['yearMin'] ? parseInt(params['yearMin'], 10) : null,
- yearMax: params['yearMax'] ? parseInt(params['yearMax'], 10) : null
- };
- }
- }

```
 /
export interface IFilterUrlMapper<TFilters> { /**


  • Convert filters to URL query parameters

   *

  • Takes a domain filter object and converts it to URL-safe query parameters.


  • Empty or null values should be omitted from the result.

   *

  • @param filters - Filter object


  • @returns URL query parameters

   */
toUrlParams(filters: TFilters): Params;


/**


  • Convert URL query parameters to filters

   *

  • Takes URL query parameters and constructs a domain filter object.


  • Missing parameters should result in null/empty values in the filter object.
```

```
   *
  • @param params - URL query parameters
  • @returns Filter object
   */
fromUrlParams(params: Params): TFilters;

/**
  • Extract highlight filters from URL parameters (optional)
   *
  • Domain-specific strategy for identifying "highlight" parameters.
    Highlight filters use h_ prefix and provide segmented statistics.
  • If not implemented, the framework assumes no highlights are present.
   *
  • @param params - URL query parameters
  • @returns Highlight filters object (domain-specific structure)
   *
  • @example
  •
```

typescript

```
  • extractHighlights(params: Params): AutomobileHighlights {
  • return {
  • manufacturer: params['h_manufacturer']?.split(',') || [],
  • yearMin: params['h_yearMin'] ? parseInt(params['h_yearMin'], 10) : null
  • };
  • }
```

```
   /
extractHighlights?(params: Params): any; }


/**


  • Response from API adapter

 *

  • Standardized response format that all API adapters must return.


  • This allows framework services to work with any domain's API.

 *

  • @template TData - The type of individual data items


  • @template TStatistics - The type of statistics object (optional)

 */
export interface ApiAdapterResponse<TData, TStatistics = any> { /**


  • Array of data results

   */
results: TData[];


/**
```

```
  • Total count of results (unpaginated)

  • Used for pagination display ("Showing 1-20 of 1,234 results")
   */
total: number;

/**

  • Optional statistics/aggregations

  • Domain-specific summary data (e.g., manufacturer counts, year distribution)
   */
statistics?: TStatistics; }

/**

  • Adapter for fetching data from API
 *
  • This interface defines how a domain fetches data from its API.

  • The adapter is responsible for:

  • - Constructing API requests from filters
```

```
  • - Transforming API responses to the standardized format

  • - Handling domain-specific API quirks
*
  • @template TFilters - The shape of the filter object

  • @template TData - The shape of individual data items

  • @template TStatistics - The shape of statistics object (optional)
*
  • @example

  •
```

typescript

- class AutomobileApiAdapter implements IApiAdapter<AutomobileFilters, Vehicle, VehicleStats> {
- constructor(private http: HttpClient, private baseUrl: string) {}

*

• fetchData(

• filters: AutomobileFilters,

• highlights?: AutomobileHighlights

•): Observable<ApiAdapterResponse<Vehicle, VehicleStats>> {

• const params = this.buildParams(filters, highlights);

• return this.http.get<ApiResponse>(`${this.baseUrl}/vehicles` , { params }).pipe(

• map(response => ({

• results: response.data,

• total: response.meta.total,

• statistics: response.stats

• }))

•);

• }

• }

```
 /
export interface IApiAdapter<TFilters, TData, TStatistics = any> { /**



  • Fetch data from API based on filters


   *


  • @param filters - Filter object derived from URL parameters


    @param highlights - Optional highlight filters (h_ parameters for segmented
statistics)


  • @returns Observable of API response with results, total count, and optional
    statistics


   */
fetchData( filters: TFilters, highlights?: any ): Observable<ApiAdapterResponse<TData,
TStatistics>>; }



/**



  • Adapter for building cache keys from filters


 *


  • This interface defines how a domain generates cache keys.



  • Cache keys are used by the RequestCoordinator to:



  • - Prevent duplicate in-flight requests
```

```
  • - Enable response caching for identical filter combinations
 *
  • @template TFilters - The shape of the filter object
 *
  • @example
  •
```

typescript

• class AutomobileCacheKeyBuilder implements ICacheKeyBuilder<AutomobileFilters> {

• buildKey(filters: AutomobileFilters, highlights?: AutomobileHighlights): string {

• const filterParts = [

• filters.manufacturer?.join(',') || '',

• filters.yearMin?.toString() || '',

• filters.yearMax?.toString() || ''

•];

*

• const highlightParts = highlights ? [

• highlights.manufacturer?.join(',') || ''

•] : [];

*

• return [...filterParts, ...highlightParts].join('|');

• }

• }

```
 /
export interface ICacheKeyBuilder<TFilters> { /**


  • Build a unique cache key from filters

   *

  • The key must be unique for each distinct filter combination.


  • Highlight filters must be included to ensure segmented statistics


  • are cached separately from non-highlighted requests.

   *

  • @param filters - Filter object


  • @param highlights - Optional highlight filters (must be included in cache key)


  • @returns Cache key string

   */
buildKey(filters: TFilters, highlights?: any): string; }


/**


  • Configuration for ResourceManagementService
```

```
 *
  • This interface bundles all the adapters and settings needed
  • to initialize the ResourceManagementService for a specific domain.
 *
  • @template TFilters - The shape of the filter object
  • @template TData - The shape of individual data items
  • @template TStatistics - The shape of statistics object (optional)
 */
export interface ResourceManagementConfig<TFilters, TData, TStatistics = any> { /**
  • Filter to URL parameter mapper
   */
filterMapper: IFilterUrlMapper<TFilters>;

/**
  • API data fetcher
   */
apiAdapter: IApiAdapter<TFilters, TData, TStatistics>;
```

```
/**


  • Cache key builder

   */
cacheKeyBuilder: ICacheKeyBuilder<TFilters>;


/**


  • Default filter values


  • Applied when application first loads or filters are cleared

   */
defaultFilters: TFilters;


/**


  • Whether to enable automatic data fetching on filter changes


  • When true, changing the URL automatically triggers API requests


  • @default true

   */
autoFetch?: boolean;
```

```
/**

  • Cache TTL in milliseconds

  • How long cached responses are considered valid

  • @default 30000 (30 seconds)

   */
cacheTTL?: number; }

/**

  • State managed by ResourceManagementService
 *
  • This interface represents the complete state of a domain's data

  • as managed by the ResourceManagementService.
 *
  • @template TFilters - The shape of the filter object

  • @template TData - The shape of individual data items

  • @template TStatistics - The shape of statistics object (optional)
```

```
 */
export interface ResourceState<TFilters, TData, TStatistics = any> { /**


  • Current filter values


  • Derived from URL query parameters

   */
filters: TFilters;


/**


  • Data results from the API

   */
results: TData[];


/**


  • Total count of results (unpaginated)

   */
totalResults: number;


/**


  • Loading state
```

```
  • True while API request is in progress

   */
loading: boolean;

/**

  • Error state

  • Contains error object if last request failed, null otherwise

   */
error: Error | null;

/**

  • Optional statistics/aggregations from API

   */
statistics?: TStatistics;

/**  Optional highlight filters (h_ parameters)

  • Used for data segmentation in charts

   */
highlights?: any; }
```

```
/**


  • Default values for ResourceManagementConfig optional fields


 */
export const RESOURCE_MANAGEMENT_DEFAULTS = { autoFetch: true, cacheTTL: 30000 // 30
seconds };
```

Step 202.2: Update the Barrel Export

Update `src/app/framework/models/index.ts` to include the new interface:

```
// src/app/framework/models/index.ts

// VERSION 2 (Section 202) - Added resource management exports // Replaces VERSION 1
from Section 201



// This barrel file exports all framework model interfaces. // Import from '@app/
framework/models' instead of individual files.



export * from './domain-config.interface'; export * from './resource-
management.interface';
```

Step 202.3: Understand the Adapter Pattern

The three adapter interfaces form a cohesive system:

Why three separate interfaces?

Single Responsibility Principle. Each interface does one thing:

Interface	Responsibility
`IFilterUrlMapper`	URL serialization/deserialization
`IApiAdapter`	API communication
`ICacheKeyBuilder`	Cache key generation

This separation means you can:

- Change URL format without touching API code
- Switch APIs without changing URL handling
- Modify caching strategy independently

Verification

1. Check File Exists

```
$ ls -la src/app/framework/models/resource-management.interface.ts
```

Expected output shows the file exists:

```
-rw-r--r-- 1 user user 6543 Feb  9 12:30 src/app/framework/models/resource-
management.interface.ts
```

2. TypeScript Compilation Check

```
$ cd ~/projects/vvroom
$ npx tsc --noEmit src/app/framework/models/resource-management.interface.ts
```

Expected: No output (no compilation errors).

3. Verify Exports

```
$ grep "^export" src/app/framework/models/resource-management.interface.ts
```

Expected output shows interface exports:

```
export interface IFilterUrlMapper<TFilters> {
export interface ApiAdapterResponse<TData, TStatistics = any> { export interface
IApiAdapter<TFilters, TData, TStatistics = any> { export interface
ICacheKeyBuilder<TFilters> { export interface ResourceManagementConfig<TFilters,
TData, TStatistics = any> { export interface ResourceState<TFilters, TData,
TStatistics = any> { export const RESOURCE_MANAGEMENT_DEFAULTS = {
```

4. Verify Barrel Export

```
$ grep "resource-management" src/app/framework/models/index.ts
```

Expected output:

```
export * from './resource-management.interface';
```

Common Problems

Symptom	Cause	Solution
`Cannot find module 'rxjs'`	RxJS not installed	Run `npm install rxjs`
`Cannot find module '@angular/router'`	Angular Router not installed	Included with Angular core
`Observable is not generic`	Wrong RxJS import	Use `import { Observable } from 'rxjs';`
`Params is not defined`	Missing router import	Add `import { Params } from '@angular/router';`
Interface properties showing as errors	Domain config using wrong types	Update domain-config.interface.ts to import from here

---

Key Takeaways

- **The adapter pattern isolates domain-specific logic** — Framework services depend on interfaces, not implementations
- **Three adapters form a cohesive system** — URL mapping, API fetching, and cache key building work together
- **ResourceState represents the complete data state** — Loading, results, errors, and filters in one place

---

Acceptance Criteria

- [ ] `src/app/framework/models/resource-management.interface.ts` exists with all interfaces
- [ ] `IFilterUrlMapper<TFilters>` defines `toUrlParams`, `fromUrlParams`, and optional `extractHighlights`
- [ ] `IApiAdapter<TFilters, TData, TStatistics>` defines `fetchData` returning Observable
- [ ] `ICacheKeyBuilder<TFilters>` defines `buildKey` method
- [ ] `ApiAdapterResponse<TData, TStatistics>` defines standard response format
- [ ] `ResourceManagementConfig` bundles all adapters with configuration
- [ ] `ResourceState` captures complete domain data state
- [ ] Barrel file (`index.ts`) exports all resource management interfaces
- [ ] TypeScript compilation succeeds with no errors
- [ ] All interfaces have JSDoc documentation with examples

Next Step

Proceed to `203-filter-definition-interface.md` to define the filter configuration interfaces used in Query Control components.

203: Filter Definition Interface

Status: Planning **Depends On:** 201-domain-config-interface **Blocks:** 601-filter-definitions, 806-query-panel-component

---

Learning Objectives

After completing this section, you will:

- Understand how configuration objects define UI behavior without custom components
- Know how to use TypeScript generics to create type-safe filter definitions
- Recognize the relationship between filter definitions and URL state synchronization

---

Objective

Create the `FilterDefinition` interface that defines how filters appear in the Query Control component. This interface specifies filter types, labels, URL parameter mappings, and range configurations — everything the framework needs to render appropriate filter UI.

Why

The vvroom application has a powerful Query Control component that lets users filter data. But instead of hard-coding filter types for automobiles, we use **configuration-driven UI**.

Consider these automobile filters:

- **Manufacturer**: Multi-select dropdown (Toyota, Ford, Honda...)
- **Year Range**: Two number inputs (min/max)
- **Body Class**: Multi-select dropdown (Sedan, SUV, Truck...)

- **Model Search**: Text input with autocomplete

Each filter type requires different UI rendering. The `FilterDefinition` interface describes each filter's characteristics, and the Query Control component reads this configuration to render the appropriate UI.

This is declarative programming:

```
// Declarative: describe WHAT you want

const yearFilter: FilterDefinition<AutomobileFilters> = { field: 'year', label: 'Model Year', type: 'range', urlParams: { min: 'yearMin', max: 'yearMax' }, rangeConfig: { valueType: 'integer', minLabel: 'Start Year', maxLabel: 'End Year', defaultRange: { min: 1900, max: 2025 } } };



// The framework figures out HOW to render it
```

Compare to imperative code where you'd write custom component logic for each filter type.

URL-First Integration

Each `FilterDefinition` includes `urlParams` — the URL query parameter name(s) for this filter. This directly supports the URL-First architecture:

Filter Type	urlParams Value	URL Example
Multiselect	`'manufacturer'`	`?manufacturer=Ford,Toyota`
Range	`{ min: 'yearMin', max: 'yearMax' }`	`?yearMin=2020&yearMax=2024`
Text	`'model'`	`?model=Camry`

What

Step 203.1: Create the Filter Definition Interface

Create the file `src/app/framework/models/filter-definition.interface.ts`:

```
// src/app/framework/models/filter-definition.interface.ts
// VERSION 1 (Section 203) - Filter definition interfaces for Query Control

/**
  • Filter Definition Interface
 *
  • Defines the structure for filterable fields in the Query Control component.

  • This is domain-agnostic and works with any domain configuration.
 */
/**
  • Configuration for range-type filters
 *
  • Supports integer, decimal, and datetime ranges with flexible formatting.
 *
  • @example

  •
```

typescript

- // Year range (integer, no grouping)
- rangeConfig: {
- valueType: 'integer',
- minLabel: 'Start Year',
- maxLabel: 'End Year',
- minPlaceholder: 'e.g., 1980',
- maxPlaceholder: 'e.g., 2023',
- step: 1,
- useGrouping: false,
- defaultRange: { min: 1900, max: new Date().getFullYear() }
- }

*

- // Price range (decimal, with grouping)
- rangeConfig: {
- valueType: 'decimal',
- minLabel: 'Min Price',
- maxLabel: 'Max Price',
- step: 0.01,
- decimalPlaces: 2,
- useGrouping: true,
- defaultRange: { min: 0, max: 1000000 }
- }

```
 /
export interface RangeConfig { /**


  • The type of values in the range


  • - 'integer': Whole numbers (years, counts)


  • - 'decimal': Floating point numbers (prices, measurements)


  • - 'datetime': Date/time values (ISO 8601 format)

   */
valueType: 'integer' | 'decimal' | 'datetime';


/**


  • Label for the minimum value input


  • @example "Start Year", "Min Price"

   */
minLabel: string;


/**


  • Label for the maximum value input
```

```
   * @example "End Year", "Max Price"
   */
  maxLabel: string;

  /**
   * Placeholder text for min input
   * @example "e.g., 1980"
   */
  minPlaceholder?: string;

  /**
   * Placeholder text for max input
   * @example "e.g., 2023"
   */
  maxPlaceholder?: string;

  /**
   * Step increment for numeric inputs
```

```
   * @default 1 for integer, 0.01 for decimal
   */
  step?: number;

  /**
   * Number of decimal places for 'decimal' type
   *
   * @default 2
   */
  decimalPlaces?: number;

  /**
   * Whether to use thousand separators in display
   *
   * Set to false for years to avoid "2,024"
   *
   * @default false
   */
  useGrouping?: boolean;

  /**
```

```
  • Default min/max values when API doesn't provide them
   */
defaultRange?: { min: number; max: number }; }

/**
  • Defines a filterable field in the domain
 *
  • This interface is used in DomainConfig.queryControlFilters array
  • to define all available filters for the Query Control component.
 *
  • @template T - The filter model type that contains these fields
 *
  • @example
  •
```

typescript

- // Multiselect filter for manufacturer
- const manufacturerFilter: FilterDefinition<AutomobileFilters> = {
- field: 'manufacturer',
- label: 'Manufacturer',
- type: 'multiselect',
- optionsEndpoint: '/agg/manufacturers',
- urlParams: 'manufacturer',
- searchPlaceholder: 'Search manufacturers...',
- dialogTitle: 'Select Manufacturers'
- };

*

- // Range filter for year
- const yearFilter: FilterDefinition<AutomobileFilters> = {
- field: 'year',
- label: 'Model Year',
- type: 'range',
- urlParams: { min: 'yearMin', max: 'yearMax' },
- rangeConfig: {
- valueType: 'integer',
- minLabel: 'Start Year',
- maxLabel: 'End Year',
- useGrouping: false,
- defaultRange: { min: 1900, max: 2025 }
- }
- };

```
 /
export interface FilterDefinition<T = any> { /**


  • Unique field identifier matching a property in the filter model


  • Uses keyof T for type safety

   */
field: keyof T;


/**


  • Display label for the filter shown in Query Control component

   */
label: string;


/**


  • Filter type determines the UI component rendered


  • - 'multiselect': Dropdown with checkboxes


  • - 'range': Two inputs for min/max values


  • - 'text': Single text input
```

```
  • - 'date': Date picker

   */
type: 'multiselect' | 'range' | 'text' | 'date';


/**

  • API endpoint for fetching multiselect filter options

  • Relative to the domain's apiBaseUrl

   *

  • @example 'agg/manufacturers'

  • Fetches from: {apiBaseUrl}/agg/manufacturers

   */
optionsEndpoint?: string;


/**

  • Transformer function to convert API response to FilterOption[]

  • Required when API response doesn't match expected format

   *

  • @example
```

```
  * optionsTransformer: (response) => response.values.map(v => ({
  *   value: v.name,
  *   label: v.name,
  *   count: v.count
  * }))
   */
optionsTransformer?: (response: any) => FilterOption[];

/**
  * URL parameter name(s) for state synchronization
   *
  * For simple filters (text, multiselect): string
  * For range filters: object with min and max keys
   *
  * @example
```

```
  • urlParams: 'manufacturer'           // ?manufacturer=Ford,Toyota


  • urlParams: { min: 'yearMin', max: 'yearMax' }  // ?yearMin=2020&yearMax=2024

   */
urlParams: string | { min: string; max: string };


/**


  • Placeholder text for search box in multiselect dialogs

   */
searchPlaceholder?: string;


/**


  • Subtitle text shown in filter dialog/modal

   */
dialogSubtitle?: string;


/**


  • Title for the filter dialog


  • @default "Select {label}"
```

```
   */
dialogTitle?: string;


/**


  • Configuration for type='range' filters


  • Required when type is 'range'

   */
rangeConfig?: RangeConfig; }


/**


  • Option for multiselect filters

 *

  • Represents a single selectable option in a multiselect dropdown.

 *

  • @example


  •
```

typescript

```
// API response format (from /agg/manufacturers endpoint)
{
field: "manufacturer",
values: [
{ value: "Toyota", label: "Toyota", count: 1543 },
{ value: "Honda", label: "Honda", count: 1234 },
{ value: "Ford", label: "Ford", count: 987 }
]
}
```

```
 /
export interface FilterOption { /**


  • The actual value used when filter is selected


  • Sent to API and stored in URL

   */
value: string | number;


/**


  • Display text shown to user in dropdown

   */
label: string;


/**


  • Optional count of items matching this option


  • Displayed as badge: "Toyota (1,543)"

   */
count?: number; }


/**
```

```
   • Default transformer for standard API aggregation responses
  *
   • Converts the standard aggregation response format to FilterOption[].
   • Use this when your API returns the expected format.
  *
   • @param response - API response with { field, values } structure
   • @returns Array of FilterOption objects
  *
   • @example
   •
```

typescript

- // API response:
- { field: "manufacturer", values: [{ value: "Ford", count: 100 }] }

*

- // Result:
- [{ value: "Ford", label: "Ford", count: 100 }]

```
 /
export function defaultOptionsTransformer(response: { field: string; values:
Array<{ value: string | number; count?: number }>; }): FilterOption[] { return
response.values.map((v) => ({ value: v.value, label: String(v.value), count:
v.count })); }



/**



  • Get URL parameter names from a filter definition

 *

  • Utility function to extract URL parameter names regardless of format.

 *

  • @param filter - Filter definition



  • @returns Array of URL parameter names

 *

  • @example



  •
```

typescript

- getUrlParamNames({ urlParams: 'manufacturer' }) // ['manufacturer']
- getUrlParamNames({ urlParams: { min: 'yearMin', max: 'yearMax' } }) // ['yearMin', 'yearMax']

```
 /
export function getUrlParamNames(filter: FilterDefinition): string[] { if (typeof
filter.urlParams === 'string') { return [filter.urlParams]; } return
[filter.urlParams.min, filter.urlParams.max]; }



/**



  • Check if a filter definition is a range filter


 *


  • @param filter - Filter definition



  • @returns True if filter type is 'range'


 */
export function isRangeFilter(filter: FilterDefinition): boolean { return filter.type
=== 'range'; }



/**



  • Check if a filter definition has URL range params


 *


  • @param urlParams - URL params from filter definition



  • @returns True if urlParams is a min/max object


 */
```

```
export function hasRangeUrlParams( urlParams: string | { min: string; max: string } ):
urlParams is { min: string; max: string } { return typeof urlParams === 'object' &&
'min' in urlParams && 'max' in urlParams; }
```

Step 203.2: Update the Barrel Export

Update `src/app/framework/models/index.ts` to include the new interface:

```
// src/app/framework/models/index.ts


// This barrel file exports all framework model interfaces. // Import from '@app/
framework/models' instead of individual files.



export * from './domain-config.interface'; export * from './resource-
management.interface'; export * from './filter-definition.interface';
```

Step 203.3: Understand Filter Types

The `FilterDefinition` interface supports four filter types:

Type	UI Component	URL Example	Use Case
`multiselect`	Dropdown with checkboxes	`?manufacturer=Ford,Toyota`	Categorical data with known options
`range`	Two number inputs	`?yearMin=2020&yearMax=2024`	Numeric or date ranges
`text`	Single text input	`?model=Camry`	Free-form text search
`date`	Date picker	`?date=2024-01-15`	Single date selection

Multiselect filters fetch options from an API endpoint. The `optionsEndpoint` specifies where to get the list, and `optionsTransformer` converts the response to `FilterOption[]` format.

Range filters require `rangeConfig` to specify:

- Value type (integer, decimal, datetime)
- Labels for min/max inputs
- Formatting options (decimal places, grouping)
- Default range when API doesn't provide bounds

Step 203.4: Example Automobile Filter Configuration

Here's how automobile filters would be configured (you'll create this in Phase 6):

```
// Preview: src/app/domain-config/automobile/configs/automobile.query-control-
filters.ts


import { FilterDefinition } from '@app/framework/models'; import { AutomobileFilters }
from '../models/automobile.filters';



export const AUTOMOBILE_QUERY_CONTROL_FILTERS: FilterDefinition<AutomobileFilters>[] =
[ { field: 'manufacturer', label: 'Manufacturer', type: 'multiselect',
optionsEndpoint: 'agg/manufacturer', urlParams: 'manufacturer', searchPlaceholder:
'Search manufacturers...', dialogTitle: 'Select Manufacturers' }, { field: 'year',
label: 'Model Year', type: 'range', urlParams: { min: 'yearMin', max: 'yearMax' },
rangeConfig: { valueType: 'integer', minLabel: 'Start Year', maxLabel: 'End Year',
minPlaceholder: 'e.g., 1980', maxPlaceholder: 'e.g., 2024', step: 1, useGrouping:
false, defaultRange: { min: 1900, max: new Date().getFullYear() } } }, { field:
'bodyClass', label: 'Body Class', type: 'multiselect', optionsEndpoint: 'agg/
body_class', urlParams: 'bodyClass', searchPlaceholder: 'Search body types...' } ];
```

This configuration tells the Query Control component exactly what to render without any custom code.

Verification

1. Check File Exists

```
$ ls -la src/app/framework/models/filter-definition.interface.ts
```

Expected output shows the file exists.

2. TypeScript Compilation Check

```
$ cd ~/projects/vvroom

$ npx tsc --noEmit src/app/framework/models/filter-definition.interface.ts
```

Expected: No output (no compilation errors).

3. Verify Exports

```
$ grep "^export" src/app/framework/models/filter-definition.interface.ts
```

Expected output:

```
export interface RangeConfig {

export interface FilterDefinition<T = any> { export interface FilterOption { export
function defaultOptionsTransformer(response: { export function
getUrlParamNames(filter: FilterDefinition): string[] { export function
isRangeFilter(filter: FilterDefinition): boolean { export function hasRangeUrlParams(
```

4. Verify Barrel Export

```
$ grep "filter-definition" src/app/framework/models/index.ts
```

Expected output:

```
export * from './filter-definition.interface';
```

Common Problems

Symptom	Cause	Solution
`Type 'keyof T' cannot be used`	TypeScript version too old	Ensure TypeScript 4.x or higher
Range filter not rendering	Missing `rangeConfig` property	Add `rangeConfig` when `type: 'range'`
Options not loading	Wrong `optionsEndpoint` value	Check API endpoint path is correct
Filter not updating URL	Wrong `urlParams` value	Verify URL param name matches expected
Type error on `urlParams`	Using wrong format	Use string for simple, object for range

---

Key Takeaways

- **Configuration-driven UI eliminates custom components** — Define filter behavior in configuration, not code
- **TypeScript generics ensure type safety** — `FilterDefinition<T>` validates that `field` matches a property in T
- **URL parameter mapping enables URL-First architecture** — Each filter knows exactly which URL params it controls

---

Acceptance Criteria

- [ ] `src/app/framework/models/filter-definition.interface.ts` exists
- [ ] `FilterDefinition<T>` interface is generic with type-safe `field` property
- [ ] `RangeConfig` interface defines all range filter options
- [ ] `FilterOption` interface defines multiselect option structure
- [ ] `urlParams` property supports both string and min/max object formats
- [ ] Utility functions `getUrlParamNames`, `isRangeFilter`, `hasRangeUrlParams` are implemented
- [ ] `defaultOptionsTransformer` function handles standard API response format
- [ ] Barrel file exports all filter definition types

- [ ] TypeScript compilation succeeds with no errors
- [ ] All interfaces have JSDoc documentation with examples

Next Step

Proceed to `204-table-config-interface.md` to define the table configuration interface for displaying results.

204: Table Config Interface

Status: Planning **Depends On:** 201-domain-config-interface **Blocks:** 602-table-config, 803-basic-results-table

---

Learning Objectives

After completing this section, you will:

- Understand how configuration objects can drive complex UI components like data tables
- Know how to use TypeScript generics for type-safe column definitions
- Recognize the benefits of configuration-driven tables versus custom table components

---

Objective

Create the `TableConfig` interface that defines how data tables are configured for display. This interface specifies columns, pagination, sorting, and other table behaviors — allowing the framework to render domain-appropriate tables without custom code.

Why

Data tables are central to vvroom's UI. Users browse automobile data in tables, sort columns, paginate through results, and expand rows for details. Instead of building a custom table component for automobiles, we use PrimeNG Table with **configuration-driven behavior**.

The problem with custom table components:

```
// Anti-pattern: Custom component for each domain

@Component({ selector: 'automobile-table',
template:  <p-table [value]="vehicles"> <ng-template pTemplate="header"> <tr>
<th pSortableColumn="manufacturer">Manufacturer</th> <th
pSortableColumn="model">Model</th> <th pSortableColumn="year">Year</th> <!--
Hard-coded for automobiles --> </tr> </ng-template> </p-table>

})
export class AutomobileTableComponent { }
```

If you add a real estate domain, you'd need a completely new `RealEstateTableComponent`.

The configuration-driven approach:

```
// Better: Generic component reads configuration

@Component({ selector: 'results-table', template:  <p-table
[value]="data" [columns]="config.columns"> <ng-template pTemplate="header">
<tr> <th *ngFor="let col of config.columns" [pSortableColumn]="col.field">
{{ col.header }} </th> </tr> </ng-template> </p-table>

})
export class ResultsTableComponent { @Input() config: TableConfig<any>; @Input()
data: any[]; }
```

Now the same component works for any domain — just provide different configuration.

PrimeNG Integration

The `TableConfig` interface is designed to work with PrimeNG Table. Column definitions map directly to PrimeNG's expectations:

TableConfig Property	PrimeNG Feature
`columns[].sortable`	`pSortableColumn` directive
`columns[].filterable`	Column filter templates
`paginator`, `rows`	`[paginator]`, `[rows]` bindings
`stateStorage`, `stateKey`	State persistence features

What

Step 204.1: Create the Table Config Interface

Create the file `src/app/framework/models/table-config.interface.ts`:

```
// src/app/framework/models/table-config.interface.ts
// VERSION 1 (Section 204) - Table configuration for PrimeNG Table


/**


  • Table Configuration Interfaces

 *

  • Configuration-driven approach for PrimeNG Table.


  • These interfaces provide type-safe table configuration without


  • requiring custom table wrapper components.

 *

  • @example


  •
```

typescript

- const vehicleTableConfig: TableConfig<Vehicle> = {
- tableId: 'vehicle-table',
- stateKey: 'vehicle-table-state',
- dataKey: 'id',
- columns: [
- { field: 'manufacturer', header: 'Manufacturer', sortable: true },
- { field: 'model', header: 'Model', sortable: true, filterable: true }
-],
- expandable: true,
- selectable: false,
- paginator: true,
- rows: 20,
- lazy: true,
- stateStorage: 'local'
- };

```
 /

/**


  • PrimeNG column configuration

 *

  • Defines a single column in a PrimeNG Table with type-safe field reference.

 *

  • @template T - The data model type

 */
export interface PrimeNGColumn<T> { /**


  • Field name from data model (type-safe via keyof T)

   */
field: keyof T;


/**


  • Display header text

   */
header: string;


/**
```

```
  • Enable column sorting


  • @default false

   */
sortable?: boolean;


/**


  • Enable column filtering


  • @default false

   */
filterable?: boolean;


/**


  • Filter match mode for PrimeNG


  • @default 'contains'

   */
filterMatchMode?:
```

```
/**
  • Enable column reordering
  • @default true
   */
reorderable?: boolean;
```

```
/**


  • Column width (CSS value: '100px', '20%', etc.)

   */
width?: string;


/**


  • Frozen column (stick to left/right during horizontal scroll)


  • @default false

   */
frozen?: boolean;


/**


  • Alignment of column content


  • @default 'left'

   */
align?: 'left' | 'center' | 'right';


/**
```

```
  • Custom CSS classes for column

   */
styleClass?: string;


/**

  • Data type for better filtering/sorting

   */
dataType?: 'text' | 'numeric' | 'date' | 'boolean'; }


/**

  • Table configuration for PrimeNG Table

 *

  • Comprehensive configuration object that drives PrimeNG Table behavior.


  • Use this instead of creating custom table wrapper components.

 *

  • @template T - The data model type

 */
export interface TableConfig<T> { /**
```

```
  • Unique identifier for this table instance

  • Used for debugging and state management
   */
tableId: string;

/**
  • State storage key for PrimeNG's stateStorage feature

  • Used to persist column order, filters, etc. in local/session storage
   */
stateKey: string;

/**
  • Column definitions
   */
columns: PrimeNGColumn<T>[];

/**
  • Unique key field for row tracking (required for expandable/selectable)
```

```
   • Should be a unique identifier field like 'id' or 'vin'

    */
dataKey: keyof T;


/**

   • Enable row expansion


   • When true, rows can be expanded to show additional details


   • @default false

    */
expandable?: boolean;


/**

   • Enable row selection


   • @default false

    */
selectable?: boolean;


/**
```

```
   • Selection mode if selectable is true


   • @default 'multiple'

    */
selectionMode?: 'single' | 'multiple';


/**

   • Enable paginator


   • @default true

    */
paginator?: boolean;


/**

   • Number of rows per page


   • @default 20

    */
rows?: number;


/**
```

```
  • Rows per page options for paginator dropdown

  • @default [10, 20, 50, 100]
   */
rowsPerPageOptions?: number[];

/**
  • Enable lazy loading mode (server-side pagination/sorting/filtering)

  • When true, table emits events instead of handling data locally

  • @default true
   */
lazy?: boolean;

/**
  • State storage mode for persisting table state

  • - 'local': localStorage (persists across sessions)

  • - 'session': sessionStorage (cleared on browser close)
```

```
   * - null: no persistence

   * @default 'local'

    */
stateStorage?: 'local' | 'session' | null;

/**

   * Enable responsive mode

   * @default true

    */
responsive?: boolean;

/**

   * Responsive layout mode

   * - 'scroll': Horizontal scrollbar on small screens

   * - 'stack': Stack columns vertically on small screens

   * @default 'scroll'
```

```
   */
responsiveLayout?: 'scroll' | 'stack';

/**

  • Enable column resizing

  • @default false

   */
resizableColumns?: boolean;

/**

  • Column resize mode

  • - 'fit': Resizing a column adjusts adjacent columns

  • - 'expand': Resizing a column expands table width

  • @default 'fit'

   */
columnResizeMode?: 'fit' | 'expand';

/**
```

```
  • Enable column reordering via drag-and-drop

  • @default true
   */
reorderableColumns?: boolean;

/**

  • CSS style classes for table element
   */
styleClass?: string;

/**

  • Show gridlines between cells

  • @default true
   */
gridlines?: boolean;

/**

  • Alternate row colors (zebra striping)
```

```
   • @default true

    */
stripedRows?: boolean;


/**

   • Show table caption/header

    */
showCaption?: boolean;


/**

   • Caption text

    */
caption?: string;


/**

   • Enable virtual scrolling for large datasets

   • When true, only visible rows are rendered

   • @default false
```

```
   */
virtualScroll?: boolean;


/**


  • Virtual scroll item size (height in pixels)


  • Required if virtualScroll is true

   */
virtualScrollItemSize?: number;


/**


  • Show loading indicator


  • Typically bound to loading state from service


  • @default false

   */
loading?: boolean; }


/**


  • Table state for managing table UI state
```

```
 *
  • Represents the runtime state of a table instance.

  • Used by components to track selection, expansion, pagination, etc.
 *
  • @template T - The data model type
 */
export interface TableState<T> { /**

  • Currently selected rows
   */
selection: T[];

/**

  • Expanded row keys

  • Key is the value of dataKey field, value is boolean
   */
expandedRowKeys: { [key: string]: boolean };

/**

  • First row index (for pagination, 0-indexed)
```

```
   */
first: number;

/**
  • Rows per page
   */
rows: number;

/**
  • Total records (for pagination display)
   */
totalRecords: number;

/**
  • Current sort field
   */
sortField?: keyof T;

/**
  • Sort order (1 = ascending, -1 = descending)
```

```
   */
sortOrder?: 1 | -1;


/**


  • Active filters (PrimeNG filter format)

   */
filters?: { [key: string]: any }; }


/**


  • Create default table configuration

 *

  • Utility function that provides sensible defaults for common table setups.


  • Override specific properties as needed.

 *

  • @template T - The data model type


  • @param tableId - Unique table identifier


  • @param dataKey - Unique key field for row tracking


  • @param columns - Column definitions
```

```
  • @returns Complete TableConfig with defaults applied
*
  • @example
  •
```

typescript

• const config = getDefaultTableConfig<Vehicle>(

• 'vehicle-table',

• 'vin',

• [

• { field: 'manufacturer', header: 'Manufacturer', sortable: true },

• { field: 'model', header: 'Model', sortable: true }

•]

•);

```
 /
export function getDefaultTableConfig<T>( tableId: string, dataKey: keyof T, columns:
PrimeNGColumn<T>[] ): TableConfig<T> { return { tableId, stateKey: ${tableId}-
state, columns, dataKey, expandable: false, selectable: false, selectionMode:
'multiple', paginator: true, rows: 20, rowsPerPageOptions: [10, 20, 50, 100], lazy:
true, stateStorage: 'local', responsive: true, responsiveLayout: 'scroll',
resizableColumns: false, columnResizeMode: 'fit', reorderableColumns: true,
styleClass: 'p-datatable-striped p-datatable-gridlines', gridlines: true, stripedRows:
true, showCaption: false, virtualScroll: false, loading: false }; }



/**



  • Get visible columns from configuration

 *

  • Filters out hidden columns and returns array suitable for template iteration.



  • Currently returns all columns; extend to support column visibility.

 *

  • @template T - The data model type



  • @param config - Table configuration



  • @returns Array of visible columns

 */
export function getVisibleColumns<T>( config: TableConfig<T> ): PrimeNGColumn<T>[]
{ return config.columns; }
```

```
/**


  • Extract PrimeNG Table bindings from config

 *

  • Converts TableConfig to object suitable for PrimeNG Table attribute binding.


  • Use with spread operator: <p-table ...getTableBindings(config)>

 *

  • @template T - The data model type


  • @param config - Table configuration


  • @returns Object with PrimeNG Table bindings

 */
export function getTableBindings<T>(config: TableConfig<T>): { dataKey: keyof T;
stateStorage: 'local' | 'session' | null; stateKey: string; paginator: boolean; rows:
number; rowsPerPageOptions: number[]; lazy: boolean; reorderableColumns: boolean;
resizableColumns: boolean; columnResizeMode: 'fit' | 'expand'; responsive: boolean;
responsiveLayout: 'scroll' | 'stack'; styleClass: string; } { return { dataKey:
config.dataKey, stateStorage: config.stateStorage ?? 'local', stateKey:
config.stateKey, paginator: config.paginator ?? true, rows: config.rows ?? 20,
rowsPerPageOptions: config.rowsPerPageOptions ?? [10, 20, 50, 100], lazy:
config.lazy ?? true, reorderableColumns: config.reorderableColumns ?? true,
resizableColumns: config.resizableColumns ?? false, columnResizeMode:
config.columnResizeMode ?? 'fit', responsive: config.responsive ?? true,
responsiveLayout: config.responsiveLayout ?? 'scroll', styleClass:
config.styleClass ?? 'p-datatable-striped p-datatable-gridlines' }; }



/**
```

```
  • Create default table state

 *

  • @template T - The data model type



  • @param rows - Initial rows per page



  • @returns Default TableState

 */
export function getDefaultTableState<T>(rows: number = 20): TableState<T> { return
{ selection: [], expandedRowKeys: {}, first: 0, rows, totalRecords: 0, sortField:
undefined, sortOrder: undefined, filters: undefined }; }
```

Step 204.2: Update the Barrel Export

Update `src/app/framework/models/index.ts` to include the new interface:

```
// src/app/framework/models/index.ts

// This barrel file exports all framework model interfaces. // Import from '@app/
framework/models' instead of individual files.



export * from './domain-config.interface'; export * from './resource-
management.interface'; export * from './filter-definition.interface'; export * from
'./table-config.interface';
```

Step 204.3: Example Automobile Table Configuration

Here's how an automobile table would be configured (you'll create this in Phase 6):

```
// Preview: src/app/domain-config/automobile/configs/automobile.table-config.ts


import { TableConfig, getDefaultTableConfig } from '@app/framework/models'; import
{ VehicleResult } from '../models/automobile.data';



export const AUTOMOBILE_TABLE_CONFIG: TableConfig<VehicleResult> =
{ ...getDefaultTableConfig<VehicleResult>('automobile-results', 'vin', []), columns:
[ { field: 'manufacturer', header: 'Manufacturer', sortable: true, width: '150px' },
{ field: 'model', header: 'Model', sortable: true, width: '150px' }, { field: 'year',
header: 'Year', sortable: true, width: '80px', align: 'center' }, { field:
'bodyClass', header: 'Body Class', sortable: true }, { field: 'engineCylinders',
header: 'Cylinders', sortable: true, width: '100px', align: 'center' }, { field:
'fuelType', header: 'Fuel Type', sortable: true } ], expandable: true, rows: 25 };
```

The `getDefaultTableConfig` helper provides sensible defaults, and we override only what's specific to automobiles.

Verification

1. Check File Exists

```
$ ls -la src/app/framework/models/table-config.interface.ts
```

Expected output shows the file exists.

2. TypeScript Compilation Check

```
$ cd ~/projects/vvroom

$ npx tsc --noEmit src/app/framework/models/table-config.interface.ts
```

Expected: No output (no compilation errors).

3. Verify Exports

```
$ grep "^export" src/app/framework/models/table-config.interface.ts
```

Expected output:

```
export interface PrimeNGColumn<T> {

export interface TableConfig<T> { export interface TableState<T> { export function
getDefaultTableConfig<T>( export function getVisibleColumns<T>( export function
getTableBindings<T>(config: TableConfig<T>): { export function
getDefaultTableState<T>(rows: number = 20): TableState<T> {
```

4. Verify Barrel Export

```
$ grep "table-config" src/app/framework/models/index.ts
```

Expected output:

```
export * from './table-config.interface';
```

Common Problems

Symptom	Cause	Solution
`keyof T` shows error	TypeScript strict mode issue	Ensure T extends object type
Column field not recognized	Field name typo	Use IDE autocomplete for `field` property
Sorting not working	`sortable: true` missing	Add `sortable: true` to column definition
State not persisting	`stateStorage` is null	Set `stateStorage: 'local'` or `'session'`
Virtual scroll not working	Missing `virtualScrollItemSize`	Set item height when using virtual scroll

---

Key Takeaways

- **Configuration replaces custom components** — One generic table component serves all domains
- **Type-safe column definitions** — `keyof T` ensures field names match data model properties
- **Utility functions reduce boilerplate** — `getDefaultTableConfig` provides sensible defaults

---

Acceptance Criteria

- [ ] `src/app/framework/models/table-config.interface.ts` exists
- [ ] `PrimeNGColumn<T>` interface defines type-safe column configuration
- [ ] `TableConfig<T>` interface includes all PrimeNG Table options
- [ ] `TableState<T>` interface captures runtime table state
- [ ] `getDefaultTableConfig` utility function is implemented
- [ ] `getTableBindings` utility function is implemented
- [ ] `getVisibleColumns` utility function is implemented
- [ ] `getDefaultTableState` utility function is implemented
- [ ] Barrel file exports all table configuration types
- [ ] TypeScript compilation succeeds with no errors
- [ ] All interfaces have JSDoc documentation with examples

Next Step

Proceed to `205-picker-config-interface.md` to define the picker configuration interface for multi-select data pickers.

205: Picker Config Interface

Status: Planning **Depends On:** 204-table-config-interface **Blocks:** 603-picker-configs, 311-picker-config-registry, 802-base-picker-component

---

Learning Objectives

After completing this section, you will:

- Understand how pickers extend table functionality with selection and URL synchronization
- Know how to configure row key generation and serialization for URL state
- Recognize the relationship between picker selections and URL query parameters

---

Objective

Create the `PickerConfig` interface that defines how selection pickers work. Pickers are specialized tables with checkboxes that let users select items and synchronize those selections with URL parameters — supporting the URL-First architecture.

Why

Pickers are tables with selection capabilities. In vvroom, users might:

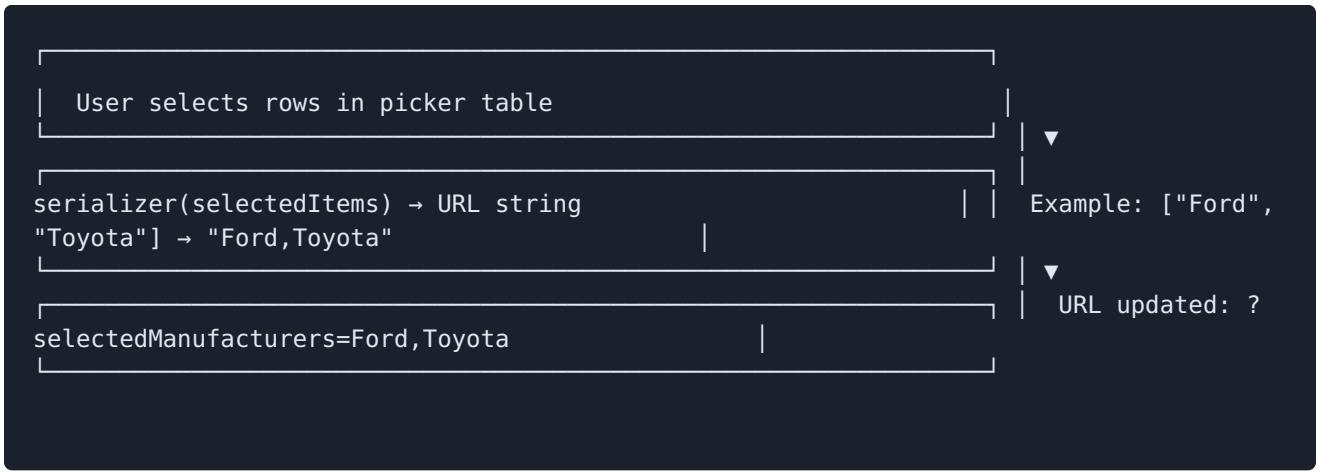
- Select specific VINs to compare
- Choose manufacturers to focus on
- Pick models for detailed analysis

These selections must be:

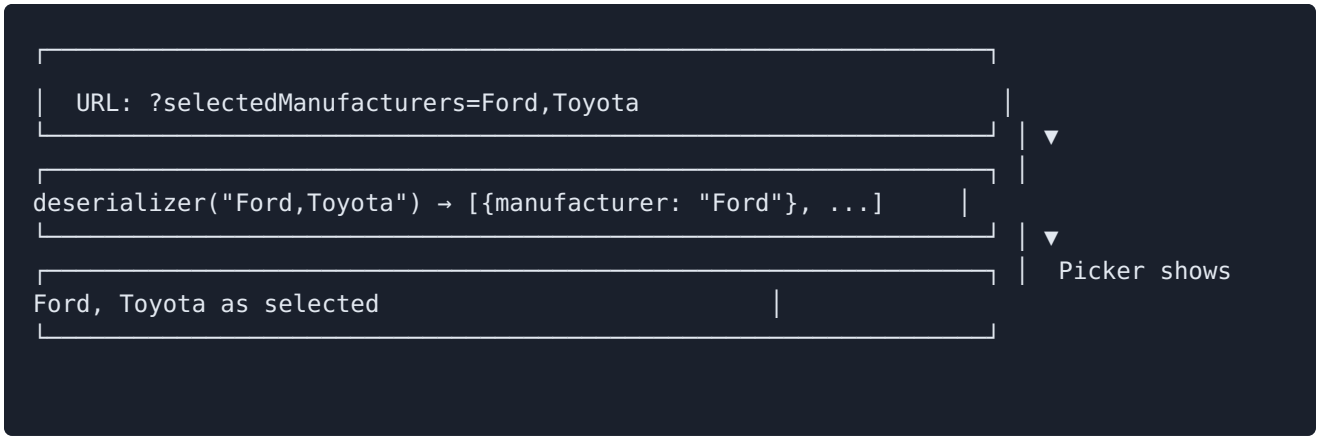
- **Persisted in the URL** — Sharing a URL shares the selection

- **Restored from URL** — Loading a URL restores the selection
- **Type-safe** — The framework knows what types are being selected

The picker flow:



Restoring from URL:



The `PickerConfig` interface defines the serialization/deserialization logic, API endpoints, and column configurations for each picker.

What

Step 205.1: Create the Picker Config Interface

Create the file `src/app/framework/models/picker-config.interface.ts`:

```
// src/app/framework/models/picker-config.interface.ts

// VERSION 1 (Section 205) - Picker configuration for selection tables



import { Observable } from 'rxjs'; import { PrimeNGColumn } from './table-
config.interface';



/**



  • Picker Configuration Interfaces


 *


  • Configuration-driven approach for selection pickers (multi-select tables).



  • Pickers use PrimeNG Table with checkboxes for row selection and



  • synchronize selections with URL query parameters.


 *


  • @example



  •
```

typescript

- const manufacturerPickerConfig: PickerConfig<Manufacturer> = {
- id: 'manufacturer-picker',
- displayName: 'Select Manufacturers',
- columns: [
- { field: 'name', header: 'Manufacturer', sortable: true },
- { field: 'country', header: 'Country', sortable: true }
-],
- api: {
- fetchData: (params) => http.get('/api/manufacturers', { params }),
- responseTransformer: (r) => ({ results: r.data, total: r.total })
- },
- row: {
- keyGenerator: (row) => row.id,
- keyParser: (key) => ({ id: key })
- },
- selection: {
- mode: 'multiple',
- urlParam: 'manufacturers',
- serializer: (items) => items.map(i => i.id).join(','),
- deserializer: (url) => url.split(',').map(id => ({ id }))
- },
- pagination: {
- mode: 'server',
- defaultPageSize: 20
- }
- };

```
 /

/**

  • Picker API configuration

 *

  • Defines how the picker fetches data from the API.

 *

  • @template T - The data model type

 */
export interface PickerApiConfig<T> { /**

  • Function to fetch data from API

  • Receives pagination/filter params, returns Observable of raw response

   */
fetchData: (params: PickerApiParams) => Observable<any>;

/**

  • Transform API response to standard format

  • Required because different APIs have different response shapes
```

```
   */
responseTransformer: (response: any) => PickerApiResponse<T>;


/**


  • Optional parameter mapper


  • Maps internal picker params to API-specific format

   *

  • @example


  • paramMapper: (params) => ({

      offset: params.page  params.size,

  •   limit: params.size,


  •   sort: params.sortField


  • })

   */
paramMapper?: (params: PickerApiParams) => any; }


/**


  • Picker API request parameters
```

```
 *

  • Standard parameters sent to the API fetch function.

 */
export interface PickerApiParams { /**


  • Current page (0-indexed)

   */
page: number;


/**


  • Page size (rows per page)

   */
size: number;


/**


  • Sort field name

   */
sortField?: string;


/**


  • Sort order (1 = ascending, -1 = descending)
```

```
   */
sortOrder?: 1 | -1;

/**

  • Search/filter term from search box

   */
search?: string;

/**

  • Additional filters (domain-specific)

   */
filters?: { [key: string]: any }; }

/**

  • Picker API response format

 *

  • Standard response format that all pickers expect.

 *

  • @template T - The data model type

 */
```

```
export interface PickerApiResponse<T> { /**


  • Array of result items for current page

   */
results: T[];


/**


  • Total number of items (for pagination)

   */
total: number;


/**


  • Current page number (optional, for verification)

   */
page?: number;


/**


  • Page size (optional, for verification)

   */
size?: number;
```

```
/**


  • Total pages (optional, computed from total/size)

   */
totalPages?: number; }


/**


  • Picker row configuration

 *

  • Defines how to generate unique keys for rows and parse them back.


  • Keys are used for selection tracking and URL serialization.

 *

  • @template T - The data model type

 */
export interface PickerRowConfig<T> { /**


  • Generate unique key from row data


  • Used for selection tracking and URL serialization

   *

  • @example
```

```
   • // Simple ID-based key

   • keyGenerator: (row) => row.id
    *
   • // Composite key

   • keyGenerator: (row) => `${row.manufacturer}|${row.model}`
    */
keyGenerator: (row: T) => string;

/**

   • Parse key back to partial row data

   • Used for URL hydration before data loads
    *
   • @example

   • // Simple ID-based parse

   • keyParser: (key) => ({ id: key })
```

```
   *
  • // Composite key parse
  • keyParser: (key) => {
  •   const [manufacturer, model] = key.split('|');
  •   return { manufacturer, model };
  • }
   */
keyParser: (key: string) => Partial<T>; }

/**
  • Picker selection configuration
 *
  • Defines selection behavior and URL synchronization.
 *
  • @template T - The data model type
 */
export interface PickerSelectionConfig<T> { /**
```

```
   • Selection mode


   • - 'single': Only one item can be selected


   • - 'multiple': Multiple items can be selected


   • @default 'multiple'

    */
mode: 'single' | 'multiple';


/**

   • URL query parameter name for storing selections

    *

   • @example


   • urlParam: 'selectedManufacturers'


   • // URL becomes: ?selectedManufacturers=Ford,Toyota,Honda

    */
urlParam: string;
```

```
/**

  • Serialize selected items to URL string

   *

  • @example

  • // Comma-separated IDs

  • serializer: (items) => items.map(i => i.id).join(',')

   *

  • // Composite keys

  • serializer: (items) => items.map(i => ${i.make}:${i.model}).join(',')

   */
serializer: (items: T[]) => string;

/**

  • Deserialize URL string to partial item data

  • Returns partial objects that will be matched against loaded data

   *

  • @example
```

```
  • // Comma-separated IDs

  • deserializer: (url) => url.split(',').map(id => ({ id }))
   *
  • // Composite keys

  • deserializer: (url) => url.split(',').map(pair => {

  •   const [make, model] = pair.split(':');

  •   return { make, model };

  • })
   */
deserializer: (urlString: string) => Partial<T>[];

/**

  • Generate key from partial item (from deserializer)

  • If not provided, uses row.keyGenerator
   */
```

```
keyGenerator?: (item: Partial<T>) => string; }



/**



  • Picker pagination configuration

 */
export interface PickerPaginationConfig { /**



  • Pagination mode



  • - 'server': Server-side pagination (API handles paging)



  • - 'client': Client-side pagination (load all, page locally)



  • @default 'server'

   */
mode: 'server' | 'client';



/**



  • Default page size



  • @default 20

   */
```

```
defaultPageSize?: number;


/**


  • Page size options for dropdown


  • @default [10, 20, 50, 100]

   */
pageSizeOptions?: number[]; }


/**


  • Picker caching configuration

 */
export interface PickerCachingConfig { /**


  • Enable caching of picker data


  • @default false

   */
enabled: boolean;


/**
```

```
  • Cache TTL in milliseconds


  • @default 300000 (5 minutes)

   */
ttl?: number; }


/**


  • Complete picker configuration

 *

  • @template T - The data model type

 */
export interface PickerConfig<T> { /**


  • Unique identifier for this picker


  • Used for registry lookup and state management

   */
id: string;


/**


  • Display name for picker (shown in UI headers)
```

```
   */
displayName: string;

/**

  • Column definitions

  • Reuses PrimeNGColumn from table config for consistency

   */
columns: PrimeNGColumn<T>[];

/**

  • API configuration

   */
api: PickerApiConfig<T>;

/**

  • Row key configuration

   */
row: PickerRowConfig<T>;

/**
```

```
  • Selection configuration

   */
selection: PickerSelectionConfig<T>;

/**

  • Pagination configuration

   */
pagination: PickerPaginationConfig;

/**

  • Optional caching configuration

   */
caching?: PickerCachingConfig;

/**

  • Optional description/help text

   */
description?: string;

/**
```

```
   * Show search box


   * @default true

    */
showSearch?: boolean;


/**


   * Search placeholder text


   * @default 'Search...'

    */
searchPlaceholder?: string; }


/**


   * Picker selection event

 *

   * Emitted when user changes selection.

 *

   * @template T - The data model type

 */
```

```
export interface PickerSelectionEvent<T> { /**


  • Picker ID that emitted the event

   */
pickerId: string;


/**


  • Selected item objects

   */
selections: T[];


/**


  • Selected item keys (from keyGenerator)

   */
selectedKeys: string[];


/**


  • Serialized URL parameter value

   */
urlValue: string; }
```

```
/**


  • Picker state

 *

  • Internal state tracking for picker component.

 *

  • @template T - The data model type

 */
export interface PickerState<T> { /**


  • All loaded data for current page

   */
data: T[];


/**


  • Total count (for pagination)

   */
totalCount: number;


/**


  • Selected row keys (as Set for O(1) lookup)
```

```
   */
selectedKeys: Set<string>;

/**

  • Selected row objects

   */
selectedItems: T[];

/**

  • Loading state

   */
loading: boolean;

/**

  • Error state

   */
error: Error | null;

/**

  • Current page (0-indexed)

   */
```

```
currentPage: number;


/**


  • Page size

   */
pageSize: number;


/**


  • Search term from search box

   */
searchTerm: string;


/**


  • Current sort field

   */
sortField?: string;


/**


  • Current sort order

   */
```

```
sortOrder?: 1 | -1;


/**


  • Pending hydration keys (from URL, before data loads)


  • These are keys that should be selected once data loads

   */
pendingHydration: string[];


/**


  • Whether data has been loaded at least once

   */
dataLoaded: boolean; }


/**


  • Create default picker state

 *

  • @template T - The data model type


  • @param pageSize - Initial page size
```

```
  * @returns Default PickerState

 */
export function getDefaultPickerState<T>(pageSize: number = 20): PickerState<T>
{ return { data: [], totalCount: 0, selectedKeys: new Set<string>(), selectedItems:
[], loading: false, error: null, currentPage: 0, pageSize, searchTerm: '', sortField:
undefined, sortOrder: undefined, pendingHydration: [], dataLoaded: false }; }



/**



  * Create default picker pagination config

 *

  * @returns Default PickerPaginationConfig

 */
export function getDefaultPaginationConfig(): PickerPaginationConfig { return { mode:
'server', defaultPageSize: 20, pageSizeOptions: [10, 20, 50, 100] }; }



/**



  * Create default picker caching config

 *

  * @param enabled - Whether caching is enabled



  * @returns Default PickerCachingConfig

 */
```

```
export function getDefaultCachingConfig(enabled: boolean = false): PickerCachingConfig
{ return { enabled, ttl: 300000 // 5 minutes }; }
```

Step 205.2: Update the Barrel Export

Update `src/app/framework/models/index.ts` to include the new interface:

```
// src/app/framework/models/index.ts

// This barrel file exports all framework model interfaces. // Import from '@app/
framework/models' instead of individual files.



export * from './domain-config.interface'; export * from './resource-
management.interface'; export * from './filter-definition.interface'; export * from
'./table-config.interface'; export * from './picker-config.interface';
```

Step 205.3: Understand the Picker Architecture

Pickers have three key responsibilities:

Responsibility	Configuration	Purpose
Data Fetching	`api`	Load data from API with pagination/search
Row Identity	`row`	Generate/parse unique keys for each row
URL Sync	`selection`	Serialize/deserialize selections to/from URL

The hydration problem:

When a user loads a URL like `?manufacturers=Ford,Toyota`, the picker must:

- Parse the URL to get selection keys: `["Ford", "Toyota"]`
- Store these as `pendingHydration`
- Fetch data from API

- Match loaded data against pending keys
- Move matched items to `selectedItems`

This is why `keyParser` exists — it creates partial objects that can be matched against full data objects.

Step 205.4: Example Automobile Picker Configuration

Here's how an automobile picker would be configured (you'll create this in Phase 6):

```
// Preview: src/app/domain-config/automobile/configs/automobile.picker-configs.ts

import { PickerConfig, getDefaultPaginationConfig } from '@app/framework/models';



interface ManufacturerOption { name: string; country: string; vehicleCount: number; }



export const MANUFACTURER_PICKER_CONFIG: PickerConfig<ManufacturerOption> = { id:
'manufacturer-picker', displayName: 'Select Manufacturers', columns: [ { field:
'name', header: 'Manufacturer', sortable: true }, { field: 'country', header:
'Country', sortable: true }, { field: 'vehicleCount', header: 'Vehicles', sortable:
true, align: 'right' } ], api: { fetchData: (params) => { // Injected via service
return null as any; }, responseTransformer: (response) => ({ results: response.data,
total: response.meta.total }) }, row: { keyGenerator: (row) => row.name, keyParser:
(key) => ({ name: key }) }, selection: { mode: 'multiple', urlParam: 'manufacturer',
serializer: (items) => items.map(i => i.name).join(','), deserializer: (url) =>
url.split(',').map(name => ({ name })) }, pagination: getDefaultPaginationConfig(),
showSearch: true, searchPlaceholder: 'Search manufacturers...' };
```

Verification

1. Check File Exists

```
$ ls -la src/app/framework/models/picker-config.interface.ts
```

Expected output shows the file exists.

2. TypeScript Compilation Check

```
$ cd ~/projects/vvroom
$ npx tsc --noEmit src/app/framework/models/picker-config.interface.ts
```

Expected: No output (no compilation errors).

3. Verify Exports

```
$ grep "^export" src/app/framework/models/picker-config.interface.ts
```

Expected output:

```
export interface PickerApiConfig<T> {
export interface PickerApiParams { export interface PickerApiResponse<T> { export
interface PickerRowConfig<T> { export interface PickerSelectionConfig<T> { export
interface PickerPaginationConfig { export interface PickerCachingConfig { export
interface PickerConfig<T> { export interface PickerSelectionEvent<T> { export
interface PickerState<T> { export function getDefaultPickerState<T>(pageSize: number =
20): PickerState<T> { export function getDefaultPaginationConfig():
PickerPaginationConfig { export function getDefaultCachingConfig(enabled: boolean =
false): PickerCachingConfig {
```

4. Verify Barrel Export

```
$ grep "picker-config" src/app/framework/models/index.ts
```

Expected output:

```
export * from './picker-config.interface';
```

Common Problems

Symptom	Cause	Solution
`Observable is not defined`	Missing RxJS import	Add `import { Observable } from 'rxjs';`
`PrimeNGColumn is not defined`	Missing import	Add `import { PrimeNGColumn } from './table-config.interface';`
Selections not persisting	Wrong `urlParam`	Verify URL parameter name matches expected
Hydration not working	`keyParser` returns wrong shape	Ensure partial object matches data structure
Duplicate selections	`keyGenerator` not unique	Use unique identifier field(s)

Key Takeaways

- **Pickers extend tables with URL-synchronized selection** — User selections become shareable URLs
- **Key generation/parsing enables hydration** — Convert between data objects and URL-safe strings
- **Serialization/deserialization are symmetric** — `deserialize(serialize(items))` should match original keys

---

Acceptance Criteria

- [ ] `src/app/framework/models/picker-config.interface.ts` exists
- [ ] `PickerConfig<T>` interface defines complete picker configuration
- [ ] `PickerApiConfig<T>` defines data fetching with transformer
- [ ] `PickerRowConfig<T>` defines key generation and parsing
- [ ] `PickerSelectionConfig<T>` defines URL serialization/deserialization
- [ ] `PickerState<T>` includes `pendingHydration` for URL restoration
- [ ] `PickerSelectionEvent<T>` defines selection change events
- [ ] Utility functions for default configs are implemented
- [ ] Barrel file exports all picker configuration types
- [ ] TypeScript compilation succeeds with no errors

- [ ] All interfaces have JSDoc documentation with examples

Next Step

Proceed to `206-api-response-interface.md` to define the standard API response interface.

206: API Response Interface

Status: Planning **Depends On:** 201-domain-config-interface **Blocks:** 302-api-service

---

Learning Objectives

After completing this section, you will:

- Understand why standardized API response formats simplify frontend development
- Know how TypeScript generics create type-safe response handling
- Recognize the relationship between API responses and pagination

---

Objective

Create the `ApiResponse` interface that defines the standard format for paginated API responses. This interface ensures consistency between backend responses and frontend expectations, enabling type-safe data handling throughout the application.

Why

APIs return data in various formats. Without a standard, every component must handle response parsing differently:

```
// Anti-pattern: Ad-hoc response handling

this.http.get('/vehicles').subscribe(response => { // Is it response.data?
response.results? response.items? // Is total in response.total? response.meta.total?
response.pagination.count? this.vehicles = response.data; // Hope this is right...
this.total = response.meta?.total ?? response.total ?? 0; // Defensive coding });
```

With a standardized `ApiResponse` interface:

```
// Better: Type-safe response handling

this.http.get<ApiResponse<Vehicle>>('/vehicles').subscribe(response => { this.vehicles
= response.results; // TypeScript knows this exists this.total =
response.total;      // TypeScript knows this exists });
```

API Contract Alignment

The `ApiResponse` interface aligns with the API contract defined in document 051 (API Contract Overview). The API server returns responses in this exact format:

```
{

"results": [...], "total": 1234, "page": 1, "size": 20, "totalPages": 62,
"statistics": { ... } }
```

By defining this interface, the frontend and backend agree on the response shape. TypeScript enforces this agreement at compile time.

What

Step 206.1: Create the API Response Interface

Create the file `src/app/framework/models/api-response.interface.ts`:

```
// src/app/framework/models/api-response.interface.ts
// VERSION 1 (Section 206) - Standardized API response format


/**


  • Generic API response interface for paginated endpoints

 *

  • This interface defines the standard response format for all paginated


  • API endpoints in the vvroom application. Both the frontend and backend


  • agree on this format, enabling type-safe data handling.

 *

  • @template TData - The type of data items in the results array

 *

  • @example


  •
```

typescript

- interface Vehicle {
- vin: string;
- manufacturer: string;
- model: string;
- year: number;
- }

*

- // Typed HTTP request
- this.http.get<ApiResponse<Vehicle>>('/api/vehicles', { params })
- .subscribe(response => {
- console.log(response.results); // Vehicle[]
- console.log(response.total); // number
- console.log(response.page); // number
- console.log(response.totalPages); // number
- });

```
 /
export interface ApiResponse<TData> { /**


  • Array of result items for the current page

   */
results: TData[];


/**


  • Total number of items across all pages


  • Used for pagination display: "Showing 1-20 of 1,234 results"

   */
total: number;


/**


  • Current page number (1-indexed)


  • Page 1 is the first page

   */
page: number;


/**
```

```
   • Number of items per page

   • Matches the size query parameter
    */
size: number;

/**

   • Total number of pages

   • Computed as: Math.ceil(total / size)
    */
totalPages: number;

/**

   • Optional statistics data (domain-specific)

   • Contains aggregations, counts, and other computed data
    *
   • @example
```

```
  • statistics: {

  •   manufacturerCounts: { Ford: 150, Toyota: 120, Honda: 89 },

  •   yearRange: { min: 1990, max: 2024 },

  •   totalVins: 1234

  • }

   */
statistics?: any; }

/**

  • Generic error response from API

 *

  • Defines the standard error format returned by the API server.

  • All API errors follow this structure for consistent error handling.

 *

  • @example

  •
```

json

- {
- "success": false,
- "error": {
- "code": "VALIDATION_ERROR",
- "message": "Invalid year range: min cannot exceed max",
- "details": {
- "field": "yearMin",
- "value": 2025,
- "constraint": "must be less than or equal to yearMax"
- }
- }
- }

```
 /
export interface ApiErrorResponse { /**


  • Success flag (always false for errors)

   */
success: false;


/**


  • Error details

   */
error: { /**


  • Error code for programmatic handling


  • @example 'VALIDATION_ERROR', 'NOT_FOUND', 'UNAUTHORIZED'

     */
code: string;


/**


  • Human-readable error message


  • Suitable for display to users
```

```
     */
message: string;


/**


  • Optional additional error details


  • Contains field-specific validation info, etc.

     */
details?: Record<string, any>; }; }


/**


  • Generic success response wrapper

 *

  • For non-paginated endpoints that return a single object or operation result.

 *

  • @template TData - The type of the response data

 *

  • @example


  •
```

typescript

- // Create vehicle response
- interface CreateVehicleResult {
- vin: string;
- created: boolean;
- }

*

- this.http.post<ApiSuccessResponse<CreateVehicleResult>>('/api/vehicles', vehicle)
- .subscribe(response => {
- if (response.success) {
- console.log('Created:', response.data.vin);
- }
- });

```
 /
export interface ApiSuccessResponse<TData> { /**


  • Success flag (always true for successful responses)

   */
success: true;


/**


  • Response data

   */
data: TData; }


/**


  • Standard API response type (success or error)

 *

  • Union type for endpoints that may return either success or error.


  • Use type guards to narrow the type:

 *

  • @example
```

```
•
```

typescript

```
• function handleResponse(response: StandardApiResponse<Vehicle>) {
• if (response.success) {
• // TypeScript knows response.data exists here
• console.log(response.data);
• } else {
• // TypeScript knows response.error exists here
• console.error(response.error.message);
• }
• }
```

```
 /
export type StandardApiResponse<TData> = ApiSuccessResponse<TData> | ApiErrorResponse;


/**


  • Type guard: Check if response is a success response

 *

  • @param response - API response to check


  • @returns True if response is a success response

 *

  • @example


  •
```

typescript

• const response = await firstValueFrom(http.get<StandardApiResponse<Vehicle>>(url));

• if (isSuccessResponse(response)) {

• console.log(response.data); // TypeScript knows data exists

• }

```
 /
export function isSuccessResponse<T>( response: StandardApiResponse<T> ): response is
ApiSuccessResponse<T> { return response.success === true; }



/**



  • Type guard: Check if response is an error response

 *

  • @param response - API response to check



  • @returns True if response is an error response

 *

  • @example



  •
```

typescript

• const response = await firstValueFrom(http.get<StandardApiResponse<Vehicle>>(url));

• if (isErrorResponse(response)) {

• console.error(response.error.message); // TypeScript knows error exists

• }

```
 /
export function isErrorResponse<T>( response: StandardApiResponse<T> ): response is
ApiErrorResponse { return response.success === false; }



/**



  • Empty paginated response

 *

  • Utility constant for initial state or empty results.

 *

  • @example



  •
```

typescript

- const [response, setResponse] = useState<ApiResponse<Vehicle>>(EMPTY_API_RESPONSE);

```
 /
export const EMPTY_API_RESPONSE: ApiResponse<never> = { results: [], total: 0, page:
1, size: 20, totalPages: 0 };



/**


  • Create an empty ApiResponse with specified type

 *

  • @template T - The data type


  • @param size - Page size (default: 20)


  • @returns Empty ApiResponse

 *

  • @example


  •
```

typescript

- const initialState: ApiResponse<Vehicle> = createEmptyResponse(25);

```
 /
export function createEmptyResponse<T>(size: number = 20): ApiResponse<T> { return
{ results: [], total: 0, page: 1, size, totalPages: 0 }; }



/**



  • Calculate total pages from total and page size

 *

  • @param total - Total number of items



  • @param size - Page size



  • @returns Total number of pages

 *

  • @example



  •
```

typescript

- calculateTotalPages(100, 20); // Returns 5
- calculateTotalPages(101, 20); // Returns 6
- calculateTotalPages(0, 20); // Returns 0

```
 /
export function calculateTotalPages(total: number, size: number): number { if (total
<= 0 || size <= 0) { return 0; } return Math.ceil(total / size); }



/**



  • Check if there are more pages after the current page


 *


  • @param page - Current page (1-indexed)



  • @param totalPages - Total number of pages



  • @returns True if there are more pages


 */
export function hasNextPage(page: number, totalPages: number): boolean { return page <
totalPages; }



/**



  • Check if there are previous pages before the current page


 *


  • @param page - Current page (1-indexed)



  • @returns True if there are previous pages
```

```
 */
export function hasPreviousPage(page: number): boolean { return page > 1; }
```

Step 206.2: Update the Barrel Export

Update `src/app/framework/models/index.ts` to include the new interface:

```
// src/app/framework/models/index.ts

// This barrel file exports all framework model interfaces. // Import from '@app/
framework/models' instead of individual files.



export * from './domain-config.interface'; export * from './resource-
management.interface'; export * from './filter-definition.interface'; export * from
'./table-config.interface'; export * from './picker-config.interface'; export * from
'./api-response.interface';
```

Step 206.3: Understand the Response Types

The API response interfaces cover three scenarios:

Interface	Use Case	Example
`ApiResponse<T>`	Paginated list endpoints	`GET /vehicles?page=1&size=20`
`ApiSuccessResponse<T>`	Non-paginated success responses	`POST /vehicles` (create)
`ApiErrorResponse`	Error responses	Any failed request

The discriminated union pattern:

`StandardApiResponse<T>` uses a discriminated union based on the `success` property:

```
// TypeScript can narrow the type based on success

const response: StandardApiResponse<Vehicle> = await getVehicle(vin);



if (response.success) { // Here, TypeScript knows response is
ApiSuccessResponse<Vehicle> console.log(response.data.manufacturer); } else { // Here,
TypeScript knows response is ApiErrorResponse console.error(response.error.code); }
```

This pattern eliminates runtime type errors by leveraging TypeScript's type narrowing.

Step 206.4: Example Usage in Services

Here's how the API response interfaces are used (preview for Phase 3):

```
// Preview: API Service usage

import { ApiResponse, isSuccessResponse } from '@app/framework/models';



@Injectable({ providedIn: 'root' }) export class AutomobileApiService
{ constructor(private http: HttpClient) {}



getVehicles(params: HttpParams): Observable<ApiResponse<Vehicle>> { return
this.http.get<ApiResponse<Vehicle>>('/api/vehicles', { params }); }



// Usage in component loadVehicles(): void
{ this.apiService.getVehicles(params).subscribe(response => { // Full type safety -
TypeScript knows the shape this.vehicles = response.results; this.total =
response.total; this.currentPage = response.page; this.totalPages =
response.totalPages; }); } }
```

Verification

1. Check File Exists

```
$ ls -la src/app/framework/models/api-response.interface.ts
```

Expected output shows the file exists.

2. TypeScript Compilation Check

```
$ cd ~/projects/vvroom

$ npx tsc --noEmit src/app/framework/models/api-response.interface.ts
```

Expected: No output (no compilation errors).

3. Verify Exports

```
$ grep "^export" src/app/framework/models/api-response.interface.ts
```

Expected output:

```
export interface ApiResponse<TData> {

export interface ApiErrorResponse { export interface ApiSuccessResponse<TData>
{ export type StandardApiResponse<TData> = ApiSuccessResponse<TData> |
ApiErrorResponse; export function isSuccessResponse<T>( export function
isErrorResponse<T>( export const EMPTY_API_RESPONSE: ApiResponse<never> = { export
function createEmptyResponse<T>(size: number = 20): ApiResponse<T> { export function
calculateTotalPages(total: number, size: number): number { export function
hasNextPage(page: number, totalPages: number): boolean { export function
hasPreviousPage(page: number): boolean {
```

4. Verify Barrel Export

```
$ grep "api-response" src/app/framework/models/index.ts
```

Expected output:

```
export * from './api-response.interface';
```

Common Problems

Symptom	Cause	Solution
`Property 'results' does not exist`	Response not typed	Add generic: `http.get<ApiResponse<T>>(...)`
Type guard not narrowing	Using wrong comparison	Use `response.success === true` (strict)
`statistics` showing error	Using without check	Add optional chaining: `response.statistics?.field`
Page calculation wrong	Off-by-one error	Remember: API pages are 1-indexed
Empty response type error	Using wrong generic	Use `createEmptyResponse<YourType>()`

Key Takeaways

- **Standardized response formats simplify development** — One interface, consistent handling everywhere
- **Type guards enable safe type narrowing** — `isSuccessResponse()` and `isErrorResponse()` for discriminated unions
- **Utility functions reduce boilerplate** — `calculateTotalPages()`, `hasNextPage()`, etc.

Acceptance Criteria

- [ ] `src/app/framework/models/api-response.interface.ts` exists
- [ ] `ApiResponse<TData>` interface defines paginated response format
- [ ] `ApiErrorResponse` interface defines error response format

- [ ] `ApiSuccessResponse<TData>` interface defines success wrapper
- [ ] `StandardApiResponse<TData>` type union is defined
- [ ] Type guards `isSuccessResponse` and `isErrorResponse` are implemented
- [ ] Utility functions for pagination are implemented
- [ ] `EMPTY_API_RESPONSE` constant is defined
- [ ] Barrel file exports all API response types
- [ ] TypeScript compilation succeeds with no errors
- [ ] All interfaces have JSDoc documentation with examples

Next Step

Proceed to `207-pagination-interface.md` to define pagination parameter interfaces.

207: Pagination Interface

Status: Planning **Depends On:** 206-api-response-interface **Blocks:** 302-api-service, 803-basic-results-table

---

Learning Objectives

After completing this section, you will:

- Understand the difference between pagination parameters (request) and metadata (response)
- Know how to combine pagination with sorting for API requests
- Recognize the relationship between URL state and pagination

---

Objective

Create the pagination interfaces that define how pagination and sorting parameters are structured for API requests and responses. These interfaces complement the `ApiResponse` interface by defining the input side of paginated queries.

Why

Pagination in vvroom follows the URL-First architecture. When a user navigates to:

```
/discover?manufacturer=Ford&page=2&size=50&sortBy=year&sortOrder=desc
```

The application must:

- Parse `page=2`, `size=50` into `PaginationParams`
- Parse `sortBy=year`, `sortOrder=desc` into `SortParams`

- Send these to the API
- Receive `PaginationMetadata` in the response

The interfaces in this document define these structures, ensuring consistency between URL state, API requests, and response handling.

Request flow:

What

Step 207.1: Create the Pagination Interface

Create the file `src/app/framework/models/pagination.interface.ts`:

```
// src/app/framework/models/pagination.interface.ts
// VERSION 1 (Section 207) - Pagination and sorting interfaces


/**


  • Pagination Interfaces

 *

  • These interfaces define pagination and sorting parameters for API requests


  • and responses. They complement the ApiResponse interface by defining


  • the input/output structures for paginated queries.

 */

/**


  • Pagination parameters for API requests

 *

  • Represents the pagination portion of an API request.


  • These values are derived from URL query parameters.

 *

  • @example
```

```
•
```

typescript

- // From URL: ?page=2&size=50
- const params: PaginationParams = {
- page: 2,
- size: 50
- };

*

- // Convert to HTTP params
- const httpParams = new HttpParams()
- .set('page', params.page.toString())
- .set('size', params.size.toString());

```
 /
export interface PaginationParams { /**


  • Page number (1-indexed)


  • Page 1 is the first page of results

   */
page: number;


/**


  • Number of items per page


  • Common values: 10, 20, 50, 100

   */
size: number; }


/**


  • Pagination metadata from API responses

 *

  • Contains complete pagination information returned by the API.


  • Use this to display pagination controls and navigation.
```

```
*
 • @example
 •
```

typescript

• // Display pagination info

• const metadata: PaginationMetadata = response.pagination;

• console.log(`Page ${metadata.page} of ${metadata.totalPages}`);

• console.log(`Showing ${metadata.size} of ${metadata.total} results`);

*

• if (metadata.hasMore) {

• showNextButton();

• }

```
 /
export interface PaginationMetadata { /**


  • Current page number (1-indexed)

   */
page: number;


/**


  • Number of items per page

   */
size: number;


/**


  • Total number of items across all pages

   */
total: number;


/**


  • Total number of pages


  • Computed as: Math.ceil(total / size)
```

```
   */
totalPages: number;


/**


  • Whether there are more pages after current page


  • Equivalent to: page < totalPages

   */
hasMore: boolean; }


/**


  • Sorting parameters for API requests

 *

  • Represents the sorting portion of an API request.


  • These values are derived from URL query parameters.

 *

  • @example


  •
```

typescript

- // From URL: ?sortBy=year&sortOrder=desc
- const params: SortParams = {
- sortBy: 'year',
- sortOrder: 'desc'
- };

*

- // Convert to HTTP params (API-specific format)
- const sortParam = `${params.sortBy}:${params.sortOrder}`;
- // Results in: sort=year:desc

```
 /
export interface SortParams { /**


  • Field to sort by


  • Must match a field name in the data model

   */
sortBy?: string;


/**


  • Sort direction


  • - 'asc': Ascending (A-Z, 1-9, oldest first)


  • - 'desc': Descending (Z-A, 9-1, newest first)

   */
sortOrder?: 'asc' | 'desc'; }


/**


  • Combined pagination and sort parameters

 *

  • Convenience interface for endpoints that support both pagination and sorting.
```

```
*
 • @example


 •
```

typescript

• // Build from URL state

• const params: PaginatedSortParams = {

• page: 1,

• size: 20,

• sortBy: 'manufacturer',

• sortOrder: 'asc'

• };

```
 /
export interface PaginatedSortParams extends PaginationParams, SortParams {}


/**


  • Default pagination parameters

 */
export const DEFAULT_PAGINATION: PaginationParams = { page: 1, size: 20 };


/**


  • Default sort parameters (no sorting)

 */
export const DEFAULT_SORT: SortParams = { sortBy: undefined, sortOrder: undefined };


/**


  • Common page size options

 */
export const PAGE_SIZE_OPTIONS: number[] = [10, 20, 50, 100];


/**


  • Create pagination metadata from response data
```

```
 *
  • Utility function to construct PaginationMetadata from API response.
 *
  • @param page - Current page number
  • @param size - Items per page
  • @param total - Total items
  • @returns Complete PaginationMetadata object
 *
  • @example
  •
```

typescript

• const response = await fetchData();

• const metadata = createPaginationMetadata(

• response.page,

• response.size,

• response.total

•);

```
 /
export function createPaginationMetadata( page: number, size: number, total: number ):
PaginationMetadata { const totalPages = size > 0 ? Math.ceil(total / size) : 0; return
{ page, size, total, totalPages, hasMore: page < totalPages }; }



/**



  • Parse pagination from URL query parameters

 *

  • @param params - URL query parameters



  • @param defaults - Default values to use if not in URL



  • @returns Parsed PaginationParams

 *

  • @example



  •
```

typescript

- const urlParams = new URLSearchParams(location.search);
- const pagination = parsePaginationFromUrl({
- page: urlParams.get('page'),
- size: urlParams.get('size')
- });

```
 /
export function parsePaginationFromUrl( params: { page?: string | null; size?: string
| null }, defaults: PaginationParams = DEFAULT_PAGINATION ): PaginationParams { return
{ page: params.page ? parseInt(params.page, 10) : defaults.page, size: params.size ?
parseInt(params.size, 10) : defaults.size }; }



/**



  • Parse sort from URL query parameters

 *

  • @param params - URL query parameters



  • @returns Parsed SortParams

 *

  • @example



  •
```

typescript

- const urlParams = new URLSearchParams(location.search);
- const sort = parseSortFromUrl({
- sortBy: urlParams.get('sortBy'),
- sortOrder: urlParams.get('sortOrder')
- });

```
 /
export function parseSortFromUrl( params: { sortBy?: string | null; sortOrder?: string
| null } ): SortParams { const sortOrder = params.sortOrder?.toLowerCase(); return
{ sortBy: params.sortBy || undefined, sortOrder: sortOrder === 'asc' || sortOrder ===
'desc' ? sortOrder : undefined }; }



/**



  • Convert pagination params to URL query parameters

 *

  • @param pagination - Pagination parameters



  • @returns Object with string values for URL

 *

  • @example



  •
```

typescript

- const urlParams = paginationToUrlParams({ page: 2, size: 50 });
- // { page: '2', size: '50' }

```
 /
export function paginationToUrlParams( pagination: PaginationParams ): { page: string;
size: string } { return { page: pagination.page.toString(), size:
pagination.size.toString() }; }



/**



  • Convert sort params to URL query parameters

 *

  • @param sort - Sort parameters



  • @returns Object with string values for URL (empty object if no sort)

 *

  • @example



  •
```

typescript

- const urlParams = sortToUrlParams({ sortBy: 'year', sortOrder: 'desc' });
- // { sortBy: 'year', sortOrder: 'desc' }

*

- const noSort = sortToUrlParams({});
- // {}

```
 /
export function sortToUrlParams( sort: SortParams ): { sortBy?: string; sortOrder?:
string } { const result: { sortBy?: string; sortOrder?: string } = {};



if (sort.sortBy) { result.sortBy = sort.sortBy; }



if (sort.sortOrder) { result.sortOrder = sort.sortOrder; }



return result; }



/**



  • Calculate the range of items shown on current page

 *

  • @param page - Current page (1-indexed)



  • @param size - Items per page



  • @param total - Total items



  • @returns Object with start and end indices (1-indexed, inclusive)

 *

  • @example
```

```
•
```

typescript

- const range = getPageRange(2, 20, 45);
- // { start: 21, end: 40 }

*

- const lastPage = getPageRange(3, 20, 45);
- // { start: 41, end: 45 }

```
 /
export function getPageRange( page: number, size: number, total: number ): { start:
number; end: number } { if (total === 0) { return { start: 0, end: 0 }; }



const start = (page - 1) * size + 1; const end = Math.min(page * size, total);



return { start, end }; }



/**



  • Format pagination for display

 *

  • @param page - Current page



  • @param size - Items per page



  • @param total - Total items



  • @returns Formatted string like "Showing 21-40 of 1,234 results"

 *

  • @example



  •
```

typescript

- const display = formatPaginationDisplay(2, 20, 1234);
- // "Showing 21-40 of 1,234 results"

```
 /
export function formatPaginationDisplay( page: number, size: number, total: number ):
string { if (total === 0) { return 'No results'; }



const { start, end } = getPageRange(page, size, total); const formattedTotal =
total.toLocaleString();



return `Showing ${start}-${end} of ${formattedTotal} results`; }
```

Step 207.2: Update the Barrel Export

Update `src/app/framework/models/index.ts` to include the new interface:

```
// src/app/framework/models/index.ts


// This barrel file exports all framework model interfaces. // Import from '@app/
framework/models' instead of individual files.



export * from './domain-config.interface'; export * from './resource-
management.interface'; export * from './filter-definition.interface'; export * from
'./table-config.interface'; export * from './picker-config.interface'; export * from
'./api-response.interface'; export * from './pagination.interface';
```

Step 207.3: Understand the Pagination Flow

The pagination interfaces connect URL state to API requests:

Layer	Interface	Example
URL	String params	`?page=2&size=50&sortBy=year`
Parse	`PaginationParams`, `SortParams`	`{ page: 2, size: 50 }`, `{ sortBy: 'year' }`
API Request	Combined params	`GET /vehicles?page=2&size=50&sort=year`
API Response	`PaginationMetadata` in `ApiResponse`	`{ page: 2, size: 50, total: 1234, ... }`

The utility functions handle conversions between these layers.

Step 207.4: Example Usage

Here's how pagination interfaces are used throughout the application:

```
// In a component

import { PaginationParams, SortParams, parsePaginationFromUrl, parseSortFromUrl,
formatPaginationDisplay } from '@app/framework/models';



@Component({...}) export class DiscoverComponent implements OnInit { pagination:
PaginationMetadata;



constructor(private route: ActivatedRoute) {}



ngOnInit(): void { this.route.queryParams.subscribe(params => { // Parse pagination
from URL const pagination = parsePaginationFromUrl({ page: params['page'], size:
params['size'] });



// Parse sort from URL const sort = parseSortFromUrl({ sortBy: params['sortBy'],
sortOrder: params['sortOrder'] });



// Fetch data with these params this.loadData(pagination, sort); }); }



get paginationDisplay(): string { return
formatPaginationDisplay( this.pagination.page, this.pagination.size,
this.pagination.total ); } }
```

Verification

1. Check File Exists

```
$ ls -la src/app/framework/models/pagination.interface.ts
```

Expected output shows the file exists.

2. TypeScript Compilation Check

```
$ cd ~/projects/vvroom
$ npx tsc --noEmit src/app/framework/models/pagination.interface.ts
```

Expected: No output (no compilation errors).

3. Verify Exports

```
$ grep "^export" src/app/framework/models/pagination.interface.ts
```

Expected output:

```
export interface PaginationParams {
export interface PaginationMetadata { export interface SortParams { export interface PaginatedSortParams extends PaginationParams, SortParams {} export const DEFAULT_PAGINATION: PaginationParams = { export const DEFAULT_SORT: SortParams = { export const PAGE_SIZE_OPTIONS: number[] = [10, 20, 50, 100]; export function createPaginationMetadata( export function parsePaginationFromUrl( export function parseSortFromUrl( export function paginationToUrlParams( export function sortToUrlParams( export function getPageRange( export function formatPaginationDisplay(
```

4. Verify Barrel Export

```
$ grep "pagination" src/app/framework/models/index.ts
```

Expected output:

```
export * from './pagination.interface';
```

Common Problems

Symptom	Cause	Solution
Page showing 0 results	Page is 0-indexed in code but 1-indexed in API	Use 1-indexed pages consistently
Sort not applied	`sortBy` is undefined	Check URL param parsing
Pagination display wrong	Off-by-one in range calculation	Use `getPageRange` helper
`hasMore` always false	Total not set correctly	Verify API returns total count
NaN in pagination	String not parsed	Use `parseInt()` with radix 10

Key Takeaways

- **Separate input params from output metadata** — `PaginationParams` for requests, `PaginationMetadata` for responses
- **Utility functions handle URL <-> object conversion** — Consistent parsing and formatting
- **1-indexed pages match user expectations** — Page 1 is the first page, not page 0

Acceptance Criteria

- [ ] `src/app/framework/models/pagination.interface.ts` exists
- [ ] `PaginationParams` interface defines page and size
- [ ] `PaginationMetadata` interface includes hasMore boolean
- [ ] `SortParams` interface defines sortBy and sortOrder
- [ ] `PaginatedSortParams` combines both interfaces
- [ ] Default constants `DEFAULT_PAGINATION`, `DEFAULT_SORT` are defined
- [ ] `PAGE_SIZE_OPTIONS` constant provides common page sizes
- [ ] Parsing functions handle null/undefined gracefully
- [ ] `formatPaginationDisplay` produces user-friendly output
- [ ] Barrel file exports all pagination types
- [ ] TypeScript compilation succeeds with no errors

Next Step

Proceed to `208-popout-interface.md` to define the pop-out window communication interfaces.

208: Popout Interface

Status: Planning **Depends On:** 201-domain-config-interface **Blocks:** 307-popout-context-service, 308-popout-manager-service, 904-popout-component

---

Learning Objectives

After completing this section, you will:

- Understand how pop-out windows communicate with the parent application
- Know how to use the BroadcastChannel API for cross-window messaging
- Recognize the challenges of maintaining state synchronization across windows

---

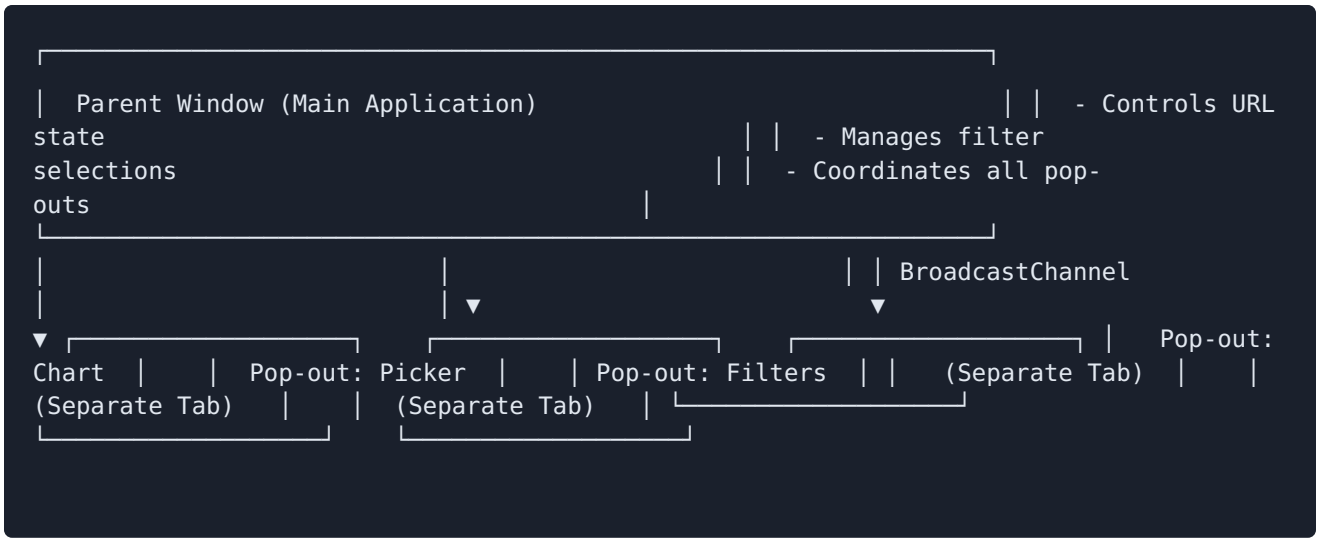
Objective

Create the popout interfaces that define how pop-out windows communicate with the parent application. These interfaces establish the message protocol, window configuration, and context tracking needed for the pop-out feature.

Why

The vvroom application supports "pop-out" functionality: users can detach panels (charts, pickers, filters) into separate browser windows. This is powerful for multi-monitor setups where users might want a chart on one screen while filtering on another.

The challenge: How do separate browser windows share state?



Solution: The BroadcastChannel API allows communication between browser contexts (tabs, windows, iframes) on the same origin.

The popout interfaces define:

- **Message types** — What kinds of messages can be sent
- **Message payloads** — What data each message contains
- **Window configuration** — Size, position, features
- **Context tracking** — Is this window a pop-out? Which panel?

What

Step 208.1: Create the Popout Interface

Create the file `src/app/framework/models/popout.interface.ts`:

```
// src/app/framework/models/popout.interface.ts
// VERSION 1 (Section 208) - Pop-out window communication interfaces


/**


  • Popout Interfaces

 *

  • Defines the communication protocol between the parent application


  • and pop-out windows. Uses BroadcastChannel API for cross-window messaging.

 */

/**


  • Pop-out message structure

 *

  • All messages between parent and pop-out windows follow this format.

 *

  • @template T - The payload type for this message

 */
export interface PopOutMessage<T = any> { /**


  • Message type identifier
```

```
   */
type: PopOutMessageType;


/**


  • Message payload (type depends on message type)

   */
payload?: T;


/**


  • Timestamp when message was created


  • Used for debugging and ordering

   */
timestamp?: number; }


/**


  • Pop-out message types

 *

  • Enumeration of all possible message types in the pop-out protocol.

 */
export enum PopOutMessageType { /**
```

```
   • State update from parent to pop-out


   • Sent when URL/filter state changes

    */
STATE_UPDATE = 'STATE_UPDATE',


/**


   • Request to close a pop-out window


   • Sent from parent when pop-out should be closed

    */
CLOSE_POPOUT = 'CLOSE_POPOUT',


/**


   • Pop-out panel is ready to receive messages


   • Sent from pop-out after initialization

    */
PANEL_READY = 'PANEL_READY',


/**
```

```
   • Picker selection changed in pop-out


   • Sent from pop-out picker to update parent URL

    */
PICKER_SELECTION_CHANGE = 'PICKER_SELECTION_CHANGE',


/**


   • Filter added in pop-out


   • Sent from pop-out filter panel to update parent URL

    */
FILTER_ADD = 'FILTER_ADD',


/**


   • Filter removed in pop-out


   • Sent from pop-out filter panel to update parent URL

    */
FILTER_REMOVE = 'FILTER_REMOVE',


/**
```

```
  • Highlight removed in pop-out

    Sent from pop-out to remove h_ parameter
*/ HIGHLIGHT_REMOVE = 'HIGHLIGHT_REMOVE',


/**

  • Clear all highlights

    Sent from pop-out to remove all h_ parameters
*/ CLEAR_HIGHLIGHTS = 'CLEAR_HIGHLIGHTS',


/**

  • Clear all filters


  • Sent from pop-out to reset all filters

   */
CLEAR_ALL_FILTERS = 'CLEAR_ALL_FILTERS',


/**

  • URL parameters changed in parent


  • Sent to pop-outs when URL updates
```

```
   */
URL_PARAMS_CHANGED = 'URL_PARAMS_CHANGED',


/**


  • Request URL parameter sync


  • Pop-out requests current URL state from parent

   */
URL_PARAMS_SYNC = 'URL_PARAMS_SYNC',


/**


  • Chart element clicked in pop-out


  • Sent to parent to apply filter from chart click

   */
CHART_CLICK = 'CHART_CLICK' }


/**


  • Payload for picker selection change messages

 */
export interface PickerSelectionPayload { /**
```

```
  • Picker configuration ID

   */
configId: string;


/**

  • URL parameter name for this picker's selections

   */
urlParam: string;


/**

  • Serialized selection value for URL

   */
urlValue: string; }


/**

  • Reference to an open pop-out window

 *

  • Used by PopOutManagerService to track and manage pop-out windows.

 */
export interface PopOutWindowRef { /**
```

```
   • Reference to the Window object

    */
window: Window;

/**

   • BroadcastChannel for communication

    */
channel: BroadcastChannel;

/**

   • Interval ID for checking if window is still open

    */
checkInterval: number;

/**

   • Panel ID within the grid

    */
panelId: string;

/**
```

```
  • Panel type (e.g., 'chart', 'picker', 'filter')

   */
panelType: string; }


/**


  • Window features for pop-out creation

 *

  • Configures the appearance and behavior of pop-out windows.

 */
export interface PopOutWindowFeatures { /**


  • Window width in pixels


  • @default 1200

   */
width?: number;


/**


  • Window height in pixels


  • @default 800
```

```
   */
height?: number;

/**

  • Left position in pixels

  • @default 100

   */
left?: number;

/**

  • Top position in pixels

  • @default 100

   */
top?: number;

/**

  • Show menu bar

  • @default false
```

```
   */
menubar?: boolean;

/**

  • Show toolbar

  • @default false

   */
toolbar?: boolean;

/**

  • Show location/address bar

  • @default false

   */
location?: boolean;

/**

  • Show status bar

  • @default false
```

```
   */
status?: boolean;


/**


  • Allow window resizing


  • @default true

   */
resizable?: boolean;


/**


  • Show scrollbars when needed


  • @default true

   */
scrollbars?: boolean; }


/**


  • Route parameters for pop-out URLs
```

```
 *

  • Pop-out windows use a special route format:


  • /popout/:gridId/:panelId/:type

 */
export interface PopOutRouteParams { /**


  • Grid container ID (e.g., 'automobile-discover')

   */
gridId: string;


/**


  • Panel ID within the grid (e.g., 'chart-year')

   */
panelId: string;


/**


  • Panel type (e.g., 'chart', 'picker', 'table')

   */
type: string; }


/**
```

```
   • Pop-out context information
 *
   • Provides context about whether the current window is a pop-out
   • and what panel it represents.
 */
export interface PopOutContext { /**
   • Whether this window is a pop-out (vs main application)
    */
isPopOut: boolean;

/**
   • Panel ID if this is a pop-out
    */
panelId?: string;

/**
   • Grid ID if this is a pop-out
    */
gridId?: string;
```

```
/**

  • Panel type if this is a pop-out

   */
panelType?: string; }

/**

  • Build window features string for window.open()

 *

  • Converts PopOutWindowFeatures to the comma-separated string

  • required by window.open().

 *

  • @param features - Window feature configuration

  • @returns Feature string for window.open()

 *

  • @example

  •
```

typescript

- const features = buildWindowFeatures({ width: 800, height: 600 });
- // "width=800,height=600,left=100,top=100,menubar=no,..."

*

- window.open('/popout/grid/panel/chart', 'popout-panel', features);

```
 /
export function buildWindowFeatures(features: PopOutWindowFeatures): string { const
{ width = 1200, height = 800, left = 100, top = 100, menubar = false, toolbar = false,
location = false, status = false, resizable = true, scrollbars = true } = features;



const boolToYesNo = (val: boolean) => (val ? 'yes' : 'no');



return [ `width=${width}`, `height=${height}`, `left=${left}`, `top=${top}`,
`menubar=${boolToYesNo(menubar)}`, `toolbar=${boolToYesNo(toolbar)}`,
`location=${boolToYesNo(location)}`, `status=${boolToYesNo(status)}`,
`resizable=${boolToYesNo(resizable)}`, `scrollbars=$
{boolToYesNo(scrollbars)}` ].join(','); }



/**



  • Parse pop-out route to extract context

 *

  • Parses a pop-out URL path to extract grid ID, panel ID, and type.

 *

  • @param url - URL path to parse



  • @returns PopOutContext if URL matches pop-out pattern, null otherwise

 *

  • @example



  •
```

typescript

- const context = parsePopOutRoute('/popout/automobile-discover/chart-year/chart');
- // {
- // isPopOut: true,
- // gridId: 'automobile-discover',
- // panelId: 'chart-year',
- // panelType: 'chart'
- // }

*

- const notPopout = parsePopOutRoute('/discover');
- // null

```
 /
export function parsePopOutRoute(url: string): PopOutContext | null { // Match /
popout/:gridId/:panelId/:type const match = url.match(/^\/popout\/([^/]+)\/([^/]+)\/
([^/?]+)/);



if (!match) { return null; }



return { isPopOut: true, gridId: match[1], panelId: match[2], panelType: match[3] }; }



/**



  • Build pop-out URL from route parameters

 *

  • @param params - Route parameters



  • @param queryString - Optional query string to append



  • @returns Pop-out URL path

 *

  • @example



  •
```

typescript

```
const url = buildPopOutUrl({
gridId: 'automobile-discover',
panelId: 'chart-year',
type: 'chart'
}, '?manufacturer=Ford');
// '/popout/automobile-discover/chart-year/chart?manufacturer=Ford'
```

```
 /
export function buildPopOutUrl( params: PopOutRouteParams, queryString: string = '' ):
string { return `/popout/${params.gridId}/${params.panelId}/${params.type}$
{queryString}`; }



/**


  • Create a pop-out message with timestamp

 *

  • @param type - Message type


  • @param payload - Message payload


  • @returns Complete PopOutMessage

 *

  • @example


  •
```

typescript

- const message = createPopOutMessage(
- PopOutMessageType.STATE_UPDATE,
- { filters: currentFilters }
-);
- channel.postMessage(message);

```
 /
export function createPopOutMessage<T>( type: PopOutMessageType, payload?: T ):
PopOutMessage<T> { return { type, payload, timestamp: Date.now() }; }



/**



  • Default window features for charts


 */
export const CHART_POPOUT_FEATURES: PopOutWindowFeatures = { width: 900, height: 600,
resizable: true, scrollbars: false };



/**



  • Default window features for pickers


 */
export const PICKER_POPOUT_FEATURES: PopOutWindowFeatures = { width: 800, height: 700,
resizable: true, scrollbars: true };



/**



  • Default window features for filter panels


 */
export const FILTER_POPOUT_FEATURES: PopOutWindowFeatures = { width: 500, height: 600,
resizable: true, scrollbars: true };



/**
```

```
  • BroadcastChannel name prefix


  • Each pop-out gets a unique channel: ${CHANNEL_PREFIX}-${panelId}

 */
export const POPOUT_CHANNEL_PREFIX = 'vvroom-popout';


/**


  • Create BroadcastChannel name for a panel

 *

  • @param panelId - Panel identifier


  • @returns Channel name

 */
export function getPopOutChannelName(panelId: string): string { return `${POPOUT_CHANNEL_PREFIX}-${panelId}`; }
```

Step 208.2: Update the Barrel Export

Update `src/app/framework/models/index.ts` to include the new interface:

```
// src/app/framework/models/index.ts


// This barrel file exports all framework model interfaces. // Import from '@app/
framework/models' instead of individual files.



export * from './domain-config.interface'; export * from './resource-
management.interface'; export * from './filter-definition.interface'; export * from
'./table-config.interface'; export * from './picker-config.interface'; export * from
'./api-response.interface'; export * from './pagination.interface'; export * from './
popout.interface';
```

Step 208.3: Understand the Pop-out Architecture

Pop-out communication follows a parent-child pattern:

Parent Window Responsibilities:

- Owns the URL state (single source of truth)
- Opens pop-out windows with specific routes
- Sends STATE_UPDATE when URL changes
- Receives messages from pop-outs (filter changes, etc.)
- Updates URL based on pop-out messages

Pop-out Window Responsibilities:

- Renders a single panel (chart, picker, etc.)
- Listens for STATE_UPDATE from parent
- Sends user actions to parent (clicks, selections)
- Closes cleanly when parent closes or navigates away

Message Flow Example:

Verification

1. Check File Exists

```
$ ls -la src/app/framework/models/popout.interface.ts
```

Expected output shows the file exists.

2. TypeScript Compilation Check

```
$ cd ~/projects/vvroom
$ npx tsc --noEmit src/app/framework/models/popout.interface.ts
```

Expected: No output (no compilation errors).

3. Verify Exports

```
$ grep "^export" src/app/framework/models/popout.interface.ts
```

Expected output:

```
export interface PopOutMessage<T = any> {

export enum PopOutMessageType { export interface PickerSelectionPayload { export
interface PopOutWindowRef { export interface PopOutWindowFeatures { export interface
PopOutRouteParams { export interface PopOutContext { export function
buildWindowFeatures(features: PopOutWindowFeatures): string { export function
parsePopOutRoute(url: string): PopOutContext | null { export function
buildPopOutUrl( export function createPopOutMessage<T>( export const
CHART_POPOUT_FEATURES: PopOutWindowFeatures = { export const PICKER_POPOUT_FEATURES:
PopOutWindowFeatures = { export const FILTER_POPOUT_FEATURES: PopOutWindowFeatures =
{ export const POPOUT_CHANNEL_PREFIX = 'vvroom-popout'; export function
getPopOutChannelName(panelId: string): string {
```

4. Verify Barrel Export

```
$ grep "popout" src/app/framework/models/index.ts
```

Expected output:

```
export * from './popout.interface';
```

Common Problems

Symptom	Cause	Solution
BroadcastChannel not defined	Older browser	Check browser compatibility or use polyfill
Pop-out shows blank	Route not configured	Add pop-out routes to app.routes.ts
Messages not received	Wrong channel name	Verify both windows use same channel name
Pop-out loses state on refresh	URL not preserved	Include query params in pop-out URL
Window features ignored	Browser restrictions	Some browsers limit window.open() features

---

Key Takeaways

- **BroadcastChannel enables cross-window communication** — Same-origin windows can send messages
- **The parent owns URL state** — Pop-outs request changes, parent applies them
- **Message types create a protocol** — Typed messages ensure consistent communication

---

Acceptance Criteria

- [ ] `src/app/framework/models/popout.interface.ts` exists
- [ ] `PopOutMessage<T>` interface defines message structure
- [ ] `PopOutMessageType` enum lists all message types
- [ ] `PopOutWindowRef` interface tracks open windows
- [ ] `PopOutWindowFeatures` interface configures window appearance
- [ ] `PopOutContext` interface identifies pop-out windows
- [ ] `buildWindowFeatures` function generates feature string
- [ ] `parsePopOutRoute` function extracts context from URL
- [ ] Default feature constants for charts, pickers, filters exist
- [ ] Barrel file exports all popout types
- [ ] TypeScript compilation succeeds with no errors

Next Step

Proceed to `209-error-notification-interface.md` to define the error notification interfaces.

209: Error Notification Interface

Status: Planning **Depends On:** 201-domain-config-interface **Blocks:** 312-error-notification-service, 313-http-error-interceptor

---

Learning Objectives

After completing this section, you will:

- Understand how centralized error handling simplifies application-wide error management
- Know how to categorize errors by source and severity for appropriate user feedback
- Recognize the relationship between HTTP interceptors and error notification services

---

Objective

Create the error notification interfaces that define how errors are categorized, formatted, and displayed to users. These interfaces establish a consistent error handling pattern using PrimeNG Toast for user-friendly error messages.

Why

Errors happen. APIs fail, networks drop, validation rules are violated. How you handle these errors determines user experience:

Poor error handling:

```
Error: [object Object]
```

Good error handling:

```
Connection Error

Unable to reach the server. Please check your network connection. [Retry] [Dismiss]
```

The error notification interfaces define:

- **Error categories** — Network, validation, authorization, server, client
- **Severity levels** — Maps to PrimeNG Toast colors (success, info, warn, error)
- **Notification structure** — Summary, detail, timestamp, original error
- **Display options** — Auto-hide duration, closable, sticky

Centralized error handling benefits:

Without Centralization	With Centralization
Error handling in every component	Single error service
Inconsistent error messages	Consistent formatting
Easy to miss errors	All errors captured
No logging	Centralized logging

What

Step 209.1: Create the Error Notification Interface

Create the file `src/app/framework/models/error-notification.interface.ts`:

```
// src/app/framework/models/error-notification.interface.ts
// VERSION 1 (Section 209) - Error notification interfaces


/**


  • Error Notification Interfaces

 *

  • Provides comprehensive error categorization and notification configuration


  • for user-facing error messages using PrimeNG Toast.

 */

/**


  • Error category enumeration

 *

  • Categorizes errors by their source and nature for appropriate handling


  • and user messaging.

 */
export enum ErrorCategory { /**


  • Network-related errors (connection issues, timeouts, etc.)
```

```
  • HTTP status 0 or network failures

   */
NETWORK = 'NETWORK',

/**

  • Validation errors (invalid input, business rule violations)

  • HTTP status 400, 422

   */
VALIDATION = 'VALIDATION',

/**

  • Authorization/authentication errors

  • HTTP status 401 (unauthorized), 403 (forbidden)

   */
AUTHORIZATION = 'AUTHORIZATION',

/**

  • Server-side errors
```

```
  • HTTP status 5xx

   */
SERVER = 'SERVER',

/**

  • Client-side errors (JavaScript errors, component errors)

  • Runtime errors in the application

   */
CLIENT = 'CLIENT',

/**

  • Application-level errors (business logic, state errors)

  • Custom application errors

   */
APPLICATION = 'APPLICATION',

/**

  • Unknown or uncategorized errors
```

```
  • Fallback for unrecognized error types

   */
UNKNOWN = 'UNKNOWN' }


/**


  • Error severity levels

 *

  • Maps to PrimeNG Toast severity levels for visual feedback.


  • - 'success': Green (not typically used for errors)


  • - 'info': Blue (informational messages)


  • - 'warn': Yellow/Orange (warnings, validation errors)


  • - 'error': Red (critical errors)

 */
export type ErrorSeverity = 'success' | 'info' | 'warn' | 'error';


/**


  • Error notification data structure
```

```
*
 • Contains all information needed to display an error to the user.
*
 • @example


 •
```

typescript

- const notification: ErrorNotification = {
- category: ErrorCategory.NETWORK,
- severity: 'error',
- summary: 'Connection Error',
- detail: 'Unable to reach the server. Please check your network connection.',
- timestamp: new Date().toISOString(),
- status: 0
- };

```
 /
export interface ErrorNotification { /**


  • Error category for categorization and routing

   */
category: ErrorCategory;


/**


  • Display severity (maps to PrimeNG Toast severity)

   */
severity: ErrorSeverity;


/**


  • Brief error summary (shown as toast title)


  • Should be short and actionable

   */
summary: string;


/**


  • Detailed error message (shown as toast body)
```

```
  • Provides more context about the error

   */
detail: string;


/**

  • Optional error code for debugging

  • From API or generated locally

   */
code?: string;


/**

  • Timestamp of when error occurred

  • ISO 8601 format

   */
timestamp?: string;


/**

  • URL where error occurred (for HTTP errors)
```

```
   */
url?: string;


/**


  • HTTP status code (for network errors)

   */
status?: number;


/**


  • Original error object (for logging/debugging)


  • Not displayed to user, but useful for console/reporting

   */
originalError?: any; }


/**


  • Error display options

 *

  • Configuration for how errors should be displayed to users.


  • Passed to PrimeNG Toast component.
```

```
 */
export interface ErrorDisplayOptions { /**


  • Auto-hide duration in milliseconds


  • Set to 0 or null to prevent auto-hide


  • @default 5000 (5 seconds)

   */
life?: number;


/**


  • Whether to show close button


  • @default true

   */
closable?: boolean;


/**


  • Whether to show in sticky mode (no auto-hide)


  • @default false
```

```
   */
sticky?: boolean;



/**



  • Custom CSS class for the toast

   */
styleClass?: string;



/**



  • Toast position key (used with multiple toast containers)



  • @default 'app-toast'

   */
key?: string; }



/**



  • Default error display options

 */
export const DEFAULT_ERROR_DISPLAY_OPTIONS: ErrorDisplayOptions = { life: 5000,
closable: true, sticky: false, key: 'app-toast' };



/**
```

```
  • Error severity mapping configuration

 *

  • Maps error categories to default severity levels.



  • - NETWORK, SERVER, CLIENT, APPLICATION, UNKNOWN → 'error' (red)



  • - VALIDATION, AUTHORIZATION → 'warn' (yellow)

 */
export const ERROR_CATEGORY_SEVERITY_MAP: Record<ErrorCategory, ErrorSeverity> =
{ [ErrorCategory.NETWORK]: 'error', [ErrorCategory.VALIDATION]: 'warn',
[ErrorCategory.AUTHORIZATION]: 'warn', [ErrorCategory.SERVER]: 'error',
[ErrorCategory.CLIENT]: 'error', [ErrorCategory.APPLICATION]: 'error',
[ErrorCategory.UNKNOWN]: 'error' };



/**



  • Determine error category from HTTP status code

 *

  • @param status - HTTP status code



  • @returns Appropriate error category

 *

  • @example
```

```
•
```

typescript

- getErrorCategoryFromStatus(401); // ErrorCategory.AUTHORIZATION
- getErrorCategoryFromStatus(500); // ErrorCategory.SERVER
- getErrorCategoryFromStatus(0); // ErrorCategory.NETWORK

```
 /
export function getErrorCategoryFromStatus(status: number): ErrorCategory { // Status
0 typically means network failure if (status === 0) { return ErrorCategory.NETWORK; }



// Authentication/authorization errors if (status === 401 || status === 403) { return
ErrorCategory.AUTHORIZATION; }



// Validation errors if (status === 400 || status === 422) { return
ErrorCategory.VALIDATION; }



// Server errors if (status >= 500 && status < 600) { return ErrorCategory.SERVER; }



// Other client errors if (status >= 400 && status < 500) { return
ErrorCategory.CLIENT; }



return ErrorCategory.UNKNOWN; }



/**



  • Determine error category from error code string

 *

  • @param code - Error code string



  • @returns Appropriate error category

 *

  • @example
```

```
•
```

typescript

- getErrorCategoryFromCode('NETWORK_TIMEOUT'); // ErrorCategory.NETWORK
- getErrorCategoryFromCode('VALIDATION_FAILED'); // ErrorCategory.VALIDATION
- getErrorCategoryFromCode('UNAUTHORIZED'); // ErrorCategory.AUTHORIZATION

```
 /
export function getErrorCategoryFromCode(code: string): ErrorCategory { const
upperCode = code.toUpperCase();



if (upperCode.includes('NETWORK') || upperCode.includes('TIMEOUT')) { return
ErrorCategory.NETWORK; }



if (upperCode.includes('VALIDATION') || upperCode.includes('INVALID')) { return
ErrorCategory.VALIDATION; }



if ( upperCode.includes('UNAUTHORIZED') || upperCode.includes('FORBIDDEN') ||
upperCode.includes('AUTH') ) { return ErrorCategory.AUTHORIZATION; }



if (upperCode.includes('SERVER') || upperCode.includes('INTERNAL')) { return
ErrorCategory.SERVER; }



if (upperCode.includes('CLIENT')) { return ErrorCategory.CLIENT; }



return ErrorCategory.UNKNOWN; }



/**



  • Get user-friendly summary for error category

 *

  • @param category - Error category



  • @returns Summary text suitable for toast title
```

```
 */
export function getSummaryForCategory(category: ErrorCategory): string { switch
(category) { case ErrorCategory.NETWORK: return 'Connection Error'; case
ErrorCategory.VALIDATION: return 'Validation Error'; case ErrorCategory.AUTHORIZATION:
return 'Access Denied'; case ErrorCategory.SERVER: return 'Server Error'; case
ErrorCategory.CLIENT: return 'Application Error'; case ErrorCategory.APPLICATION:
return 'Operation Failed'; case ErrorCategory.UNKNOWN: default: return 'Error'; } }



/**



  • Create error notification from HTTP error

 *

  • Converts an HTTP error response to an ErrorNotification.



  • Typically called from HTTP error interceptor.

 *

  • @param error - HTTP error object (from HttpErrorResponse or interceptor)



  • @returns ErrorNotification object ready for display

 *

  • @example



  •
```

typescript

```
// In HTTP interceptor
catchError((error: HttpErrorResponse) => {
const notification = createErrorNotificationFromHttpError({
status: error.status,
message: error.message,
url: error.url,
code: error.error?.code
});
this.errorService.show(notification);
return throwError(() => error);
})
```

```
 /
export function createErrorNotificationFromHttpError(error: { status?: number;
message?: string; url?: string; code?: string; timestamp?: string; }):
ErrorNotification { const status = error.status || 0; const code = error.code ||
'UNKNOWN_ERROR'; const category = getErrorCategoryFromStatus(status); const severity =
ERROR_CATEGORY_SEVERITY_MAP[category];



return { category, severity, summary: getSummaryForCategory(category), detail:
error.message || 'An unexpected error occurred', code, timestamp: error.timestamp ||
new Date().toISOString(), url: error.url, status, originalError: error }; }



/**



  • Create error notification from generic Error

 *

  • Converts a JavaScript Error to an ErrorNotification.



  • Typically called from global error handler.

 *

  • @param error - JavaScript Error object



  • @returns ErrorNotification object ready for display

 *

  • @example



  •
```

typescript

```
// In global error handler
handleError(error: Error): void {
const notification = createErrorNotificationFromError(error);
this.errorService.show(notification);
}
```

```
 /
export function createErrorNotificationFromError(error: Error): ErrorNotification
{ const code = (error as any).code; const category = code ?
getErrorCategoryFromCode(code) : ErrorCategory.CLIENT; const severity =
ERROR_CATEGORY_SEVERITY_MAP[category];



return { category, severity, summary: getSummaryForCategory(category), detail:
error.message || 'An unexpected error occurred', code, timestamp: new
Date().toISOString(), originalError: error }; }



/**



  • Create a custom error notification

 *

  • For application-specific errors that don't come from HTTP or JavaScript errors.

 *

  • @param summary - Brief error summary



  • @param detail - Detailed error message



  • @param category - Error category (default: APPLICATION)



  • @returns ErrorNotification object

 *

  • @example
```

```
•
```

typescript

• // Business logic error

• if (cart.items.length === 0) {

• const notification = createCustomErrorNotification(

• 'Empty Cart',

• 'Please add items to your cart before checkout.',

• ErrorCategory.APPLICATION

•);

• this.errorService.show(notification);

• }

```
 /
export function createCustomErrorNotification( summary: string, detail: string,
category: ErrorCategory = ErrorCategory.APPLICATION ): ErrorNotification { const
severity = ERROR_CATEGORY_SEVERITY_MAP[category];



return { category, severity, summary, detail, timestamp: new Date().toISOString() }; }



/**



  • Default display options by error category


 *


  • Some error types should be more prominent than others.



  • - Authorization errors are sticky (user must acknowledge)



  • - Validation errors auto-hide quickly


 */
export const CATEGORY_DISPLAY_OPTIONS: Record<ErrorCategory,
Partial<ErrorDisplayOptions>> = { [ErrorCategory.NETWORK]: { life: 8000 },
[ErrorCategory.VALIDATION]: { life: 4000 }, [ErrorCategory.AUTHORIZATION]: { sticky:
true, closable: true }, [ErrorCategory.SERVER]: { life: 6000 },
[ErrorCategory.CLIENT]: { life: 5000 }, [ErrorCategory.APPLICATION]: { life: 5000 },
[ErrorCategory.UNKNOWN]: { life: 5000 } };



/**



  • Merge display options with category defaults
```

```
 *

  • @param category - Error category



  • @param options - Custom display options



  • @returns Merged display options


 */
export function mergeDisplayOptions( category: ErrorCategory, options?:
Partial<ErrorDisplayOptions> ): ErrorDisplayOptions { return
{ ...DEFAULT_ERROR_DISPLAY_OPTIONS, ...CATEGORY_DISPLAY_OPTIONS[category], ...options
}; }
```

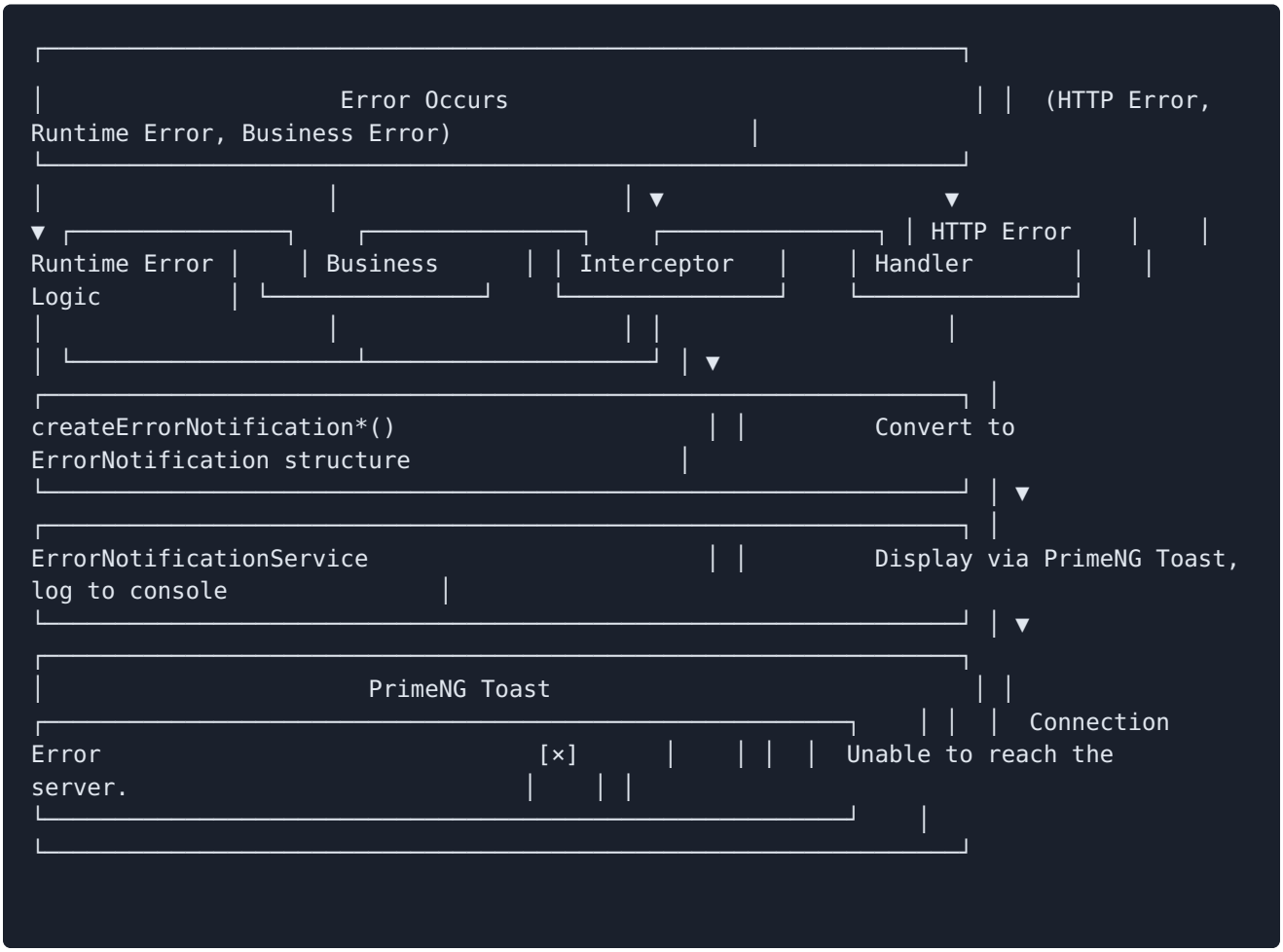
Step 209.2: Update the Barrel Export

Update `src/app/framework/models/index.ts` to include the new interface:

```
// src/app/framework/models/index.ts


// This barrel file exports all framework model interfaces. // Import from '@app/
framework/models' instead of individual files.



export * from './domain-config.interface'; export * from './resource-
management.interface'; export * from './filter-definition.interface'; export * from
'./table-config.interface'; export * from './picker-config.interface'; export * from
'./api-response.interface'; export * from './pagination.interface'; export * from './
popout.interface'; export * from './error-notification.interface';
```

Step 209.3: Understand the Error Handling Flow

Error handling in vvroom follows this pattern:



Step 209.4: Phase 2 Complete

Congratulations! You have completed Phase 2: Framework Models.

What you built:

Document	Interface	Purpose
201	`DomainConfig<T>`	Central configuration for domains
202	`IApiAdapter<T>`, `IFilterUrlMapper<T>`	URL-First adapters
203	`FilterDefinition<T>`	Query Control filters
204	`TableConfig<T>`, `PrimeNGColumn<T>`	Data table configuration
205	`PickerConfig<T>`	Selection picker configuration
206	`ApiResponse<T>`	Standard API response format
207	`PaginationParams`, `SortParams`	Pagination utilities
208	`PopOutMessage`, `PopOutContext`	Pop-out communication
209	`ErrorNotification`, `ErrorCategory`	Error handling

Phase 2 Aha Moment achieved: "TypeScript interfaces are executable documentation."

These interfaces don't execute at runtime — they have zero impact on bundle size. But they provide:

- **Compile-time safety** — TypeScript catches configuration errors
- **IDE support** — Autocomplete and inline documentation
- **Living documentation** — The interface IS the specification

---

Verification

1. Check File Exists

```
$ ls -la src/app/framework/models/error-notification.interface.ts
```

Expected output shows the file exists.

2. TypeScript Compilation Check

```
$ cd ~/projects/vvroom
$ npx tsc --noEmit src/app/framework/models/error-notification.interface.ts
```

Expected: No output (no compilation errors).

3. Verify All Model Files Exist

```
$ ls -1 src/app/framework/models/
```

Expected output:

```
api-response.interface.ts
domain-config.interface.ts error-notification.interface.ts filter-
definition.interface.ts index.ts pagination.interface.ts picker-config.interface.ts
popout.interface.ts resource-management.interface.ts table-config.interface.ts
```

4. Verify Barrel Exports All Interfaces

```
$ grep "export" src/app/framework/models/index.ts
```

Expected: All 9 interface files are exported.

5. Phase 2 Checkpoint

```
$ cd ~/projects/vvroom
$ ng build
```

Expected: Build succeeds with no errors. All interfaces compile.

Common Problems

Symptom	Cause	Solution
`ErrorCategory is not defined`	Missing import	Import from the interface file
Toast not appearing	MessageService not provided	Add to providers in AppModule
Severity color wrong	Wrong severity string	Use: 'success', 'info', 'warn', 'error'
Timestamp format wrong	Not ISO 8601	Use `new Date().toISOString()`
originalError circular reference	Logging issue	JSON.stringify with replacer

---

Key Takeaways

- **Centralized error handling provides consistency** — One place to format and display all errors
- **Error categories enable smart handling** — Different categories get different treatment
- **Factory functions simplify error creation** — `createErrorNotificationFromHttpError()`, etc.

Acceptance Criteria

- [ ] `src/app/framework/models/error-notification.interface.ts` exists
- [ ] `ErrorCategory` enum defines all error categories
- [ ] `ErrorSeverity` type matches PrimeNG Toast severities
- [ ] `ErrorNotification` interface captures all error information
- [ ] `ErrorDisplayOptions` interface configures toast behavior
- [ ] `getErrorCategoryFromStatus` correctly categorizes HTTP errors
- [ ] `getErrorCategoryFromCode` correctly categorizes error codes
- [ ] Factory functions create notifications from different error sources
- [ ] `CATEGORY_DISPLAY_OPTIONS` provides sensible defaults per category
- [ ] Barrel file exports all error notification types

- [ ] TypeScript compilation succeeds with no errors

Phase 2 Checkpoint

Before proceeding to Phase 3 (Framework Services), verify:

- [ ] All 9 interface files exist in `src/app/framework/models/`
- [ ] Barrel file (`index.ts`) exports all interfaces
- [ ] `ng build` completes with no errors
- [ ] Each interface has JSDoc documentation
- [ ] No runtime code exists yet — these are all types

Next Step

Proceed to `250-rxjs-patterns-primer.md` (Interlude B) to learn the RxJS patterns needed for Phase 3 services.

250: RxJS Patterns Primer

Status: Planning **Depends On:** 209-error-notification-interface **Blocks:** 301-url-state-service

---

Learning Objectives

After completing this section, you will:

- Understand why Observables are well-suited for modeling state that changes over time
- Know how to use the key RxJS operators used in vvroom: `switchMap`, `combineLatest`, `distinctUntilChanged`, `shareReplay`
- Be able to apply proper cleanup patterns to prevent memory leaks

---

Objective

Build practical RxJS knowledge before tackling the framework services in Phase 3. This is a teaching interlude focused on the specific RxJS patterns used in vvroom — it's practical, not comprehensive.

Why This Interlude Exists

Phase 3 introduces services like `UrlStateService` and `ResourceManagementService` that use RxJS extensively:

```
// From UrlStateService (Section 301)

this.filters$ = this.route.queryParams.pipe( map(params =>
this.urlMapper.fromUrlParams(params)), distinctUntilChanged((a, b) =>
JSON.stringify(a) === JSON.stringify(b)), shareReplay(1) );



// From ResourceManagementService (Section 306) this.results$ =
this.filters$.pipe( switchMap(filters => this.apiAdapter.fetchData(filters)),
map(response => response.results), shareReplay(1) );
```

If you've never used RxJS, this code is confusing. What does `switchMap` do? Why `shareReplay(1)`? This primer answers those questions with practical examples.

What Are Observables?

The Problem: Modeling Change Over Time

Traditional variables represent a value at a single point in time:

```
let count = 0;

count = 1; count = 2; // Each assignment replaces the previous value
```

But in UI applications, we often need to react to changes:

- When the URL changes, update the display
- When a filter changes, fetch new data
- When data arrives, update the table

Observables model **sequences of values over time**:

```
// Observable of values: 0, then 1, then 2, over time
const count$ = interval(1000); // Emits 0, 1, 2, ... every second


count$.subscribe(value => { console.log(value); // Logs each value as it arrives });
```

The `$` suffix is a naming convention indicating "this is an Observable."

Observables vs Promises

Promises	Observables
Single value	Multiple values over time
Eager (starts immediately)	Lazy (starts on subscribe)
Not cancellable	Cancellable (unsubscribe)
No operators	Rich operator library

Promises are great for one-time async operations. Observables are better for ongoing data streams like user input, WebSocket messages, or state changes.

The Four Essential Operators

Vvroom uses many RxJS operators, but four are critical to understand:

1. `switchMap` — Cancel and Switch

Problem: When the URL changes rapidly (user clicking filters quickly), you don't want to process all intermediate requests — just the latest one.

```
import { switchMap } from 'rxjs/operators';


// Without switchMap: every filter change triggers a request // Results may arrive out
of order, showing stale data



// With switchMap: previous request is cancelled when a new one starts this.results$ =
this.filters$.pipe( switchMap(filters => this.apiAdapter.fetchData(filters)) );
```

How it works:

```
filters$:    --A------B--C-------->

\      \  \ fetchData(A):    ---X   |  |      (cancelled) fetchData(B):          -X
|      (cancelled) fetchData(C):              ----Y--> results$:    ----------------
Y--->



Only the result of the last request (C → Y) is emitted.
```

When to use: API calls triggered by user input where only the latest matters.

2. `combineLatest` — Combine Multiple Streams

Problem: You need to react when *any* of several values change, and you need all current values together.

```
import { combineLatest } from 'rxjs';


// Combine filters and highlights into a single stream const combined$ =
combineLatest([ this.filters$, this.highlights$ ]);



combined$.subscribe(([filters, highlights]) => { // Called whenever either filters OR
highlights change // Always has the latest value of both this.fetchData(filters,
highlights); });
```

How it works:

```
filters$:      --A----B---------->
highlights$:   ----1----2--------> combined$:     ----[A,1][B,1][B,2]>


Emits when either changes, always with latest values.
```

Important: `combineLatest` doesn't emit until *all* source Observables have emitted at least once. If one never emits, combined never emits.

When to use: Combining multiple independent state sources.

3. `distinctUntilChanged` — Skip Duplicates

Problem: The URL might "change" to the same value (user clicks same link twice). You don't want to re-fetch identical data.

```
import { distinctUntilChanged } from 'rxjs/operators';

this.filters$ = this.route.queryParams.pipe( map(params =>
this.urlMapper.fromUrlParams(params)), distinctUntilChanged((a, b) =>
JSON.stringify(a) === JSON.stringify(b)) );
```

How it works:

```
input:    --A--A--B--B--A-->
output:   --A-----B-----A-->


Consecutive duplicates are skipped.
```

The comparator function: For objects, you need a custom comparator because `{a:1} !== {a:1}` (different object references). The example uses `JSON.stringify` for deep comparison. For large objects, consider a more efficient comparison.

When to use: Preventing unnecessary work when values haven't actually changed.

4. `shareReplay(1)` — Share and Cache

Problem: Multiple components subscribe to `filters$`. Without sharing, each subscription triggers a new execution of the Observable chain.

```
import { shareReplay } from 'rxjs/operators';


// Without shareReplay: each subscriber gets a separate stream // URL parsing happens
multiple times



// With shareReplay(1): all subscribers share one stream this.filters$ =
this.route.queryParams.pipe( map(params => this.urlMapper.fromUrlParams(params)),
distinctUntilChanged((a, b) => JSON.stringify(a) === JSON.stringify(b)),
shareReplay(1)  // Share with all subscribers, replay last value );
```

How it works:

```
Without shareReplay:

Subscriber A: queryParams → map → distinctUntilChanged Subscriber B: queryParams → map
→ distinctUntilChanged  (separate execution)



With shareReplay(1): Subscriber A ─┐  ├─ queryParams → map → distinctUntilChanged
(shared) Subscriber B ─┘
```

The `1` parameter: Replay the last 1 value to new subscribers. This means late subscribers immediately get the current value without waiting for the next emission.

When to use: Any Observable that multiple components will subscribe to.

Combining Operators: A Real Example

Here's how these operators work together in `ResourceManagementService`:

```
// Setup: combine filters and highlights

private readonly combined$ = combineLatest([ this.filters$,
this.highlights$ ]).pipe( // Skip if neither actually changed distinctUntilChanged((a,
b) => JSON.stringify(a) === JSON.stringify(b) ) );



// Fetch data when combined state changes this.results$ = this.combined$.pipe( //
Cancel previous request, start new one switchMap(([filters, highlights]) =>
this.apiAdapter.fetchData(filters, highlights) ), // Extract just the results
map(response => response.results), // Share with all subscribers, cache last value
shareReplay(1) );
```

The flow:

- `combineLatest` — Whenever filters OR highlights change, emit both
- `distinctUntilChanged` — Skip if the combined value is the same
- `switchMap` — Cancel any in-flight request, start a new one
- `map` — Extract the results from the response
- `shareReplay(1)` — Share with all components, cache the latest

Error Handling

Observables have built-in error handling with `catchError`:

```
import { catchError, of } from 'rxjs';

this.results$ = this.filters$.pipe( switchMap(filters =>
this.apiAdapter.fetchData(filters).pipe( catchError(error => { // Log the error
console.error('API Error:', error);



// Notify the user this.errorService.notify(error);



// Return empty results (don't break the stream) return of({ results: [], total:
0 }); }) )), map(response => response.results), shareReplay(1) );
```

Important: `catchError` must return an Observable. Use `of()` to create an Observable from a static value.

Where to put catchError: Inside the `switchMap` callback, not after it. This way, errors in one request don't kill the entire stream — future filter changes will still trigger new requests.

Cleanup Patterns

Observables can cause memory leaks if not properly cleaned up. Every `subscribe()` must eventually `unsubscribe()`.

Pattern 1: Store and Unsubscribe

```
import { Subscription } from 'rxjs';


@Component({ / ... / }) export class MyComponent implements OnDestroy { private
subscription: Subscription;



ngOnInit() { this.subscription = this.filters$.subscribe(filters => { // Handle
filters }); }



ngOnDestroy() { this.subscription.unsubscribe(); } }
```

Pattern 2: Use `takeUntil` (Preferred)

```
import { Subject } from 'rxjs';

import { takeUntil } from 'rxjs/operators';



@Component({ / ... / }) export class MyComponent implements OnDestroy { private
destroy$ = new Subject<void>();



ngOnInit() { this.filters$.pipe( takeUntil(this.destroy$) ).subscribe(filters => { //
Handle filters });



// Multiple subscriptions can share the same destroy$
this.results$.pipe( takeUntil(this.destroy$) ).subscribe(results => { // Handle
results }); }



ngOnDestroy() { this.destroy$.next(); this.destroy$.complete(); } }
```

Pattern 3: Use the Async Pipe (Best)

The async pipe in templates automatically subscribes and unsubscribes:

```
@Component({
template: `<div *ngIf="filters$ | async as filters">Current manufacturer: {{ filters.manufacturer }}</div>`
})
export class MyComponent { filters$ = this.urlStateService.filters$; }
```

Why this is best:

- No manual subscription management
- No memory leak risk
- Angular handles everything

Use the async pipe whenever possible. Fall back to `takeUntil` when you need the value in component code.

The Aha Moment

Observables model change over time. That's why they fit state management.

In vvroom's URL-First architecture:

- The URL is state
- The URL changes over time (user navigates, clicks filters)
- Components need to react to those changes

Observables are the perfect fit:

```
// URL changes over time

this.route.queryParams  // Observable<Params>



// Filters derived from URL, changing over time this.filters$  // Observable<TFilters>



// Results derived from filters, changing over time this.results$  //
Observable<TData[]>



// Components react to changes this.results$.subscribe(results =>
this.updateTable(results));
```

The entire data flow is modeled as Observables:

```
URL → queryParams → filters → API request → results → display

(Observable)  (Observable)            (Observable)
```

When the URL changes, the entire chain reacts automatically. You don't manually trigger updates — you describe how data flows, and RxJS handles the rest.

Quick Reference

Operator	Purpose	Use When
`switchMap`	Cancel previous, use latest	API calls from user input
`combineLatest`	Combine multiple streams	Need all current values together
`distinctUntilChanged`	Skip duplicates	Preventing unnecessary work
`shareReplay(1)`	Share and cache	Multiple subscribers
`catchError`	Handle errors	API calls that might fail
`takeUntil`	Auto-unsubscribe	Component cleanup
`map`	Transform values	Converting response formats
`filter`	Skip values	Conditional processing
`tap`	Side effects	Logging, debugging

Practice Exercises

Try these in your editor or the TypeScript Playground.

Exercise 1: Basic Observable

What does this log?

```
import { of } from 'rxjs';
import { map } from 'rxjs/operators';


of(1, 2, 3).pipe( map(x => x * 2) ).subscribe(console.log);
```

<details> <summary>Answer</summary>

```
2
4 6
```

`of(1, 2, 3)` emits three values. `map(x => x * 2)` doubles each. </details>

Exercise 2: switchMap

If `fetchData(id)` returns a Promise, what's the output?

```
import { of } from 'rxjs';
import { switchMap, delay } from 'rxjs/operators';



of('A', 'B', 'C').pipe( switchMap(id => of(`Result: ${id}`).pipe(delay(100))) ).subscribe(console.log);
```

<details> <summary>Answer</summary>

```
Result: C
```

Only the last value (`C`) produces output because `switchMap` cancels previous inner Observables when a new value arrives. </details>

Exercise 3: combineLatest

When does this emit?

```
import { combineLatest, Subject } from 'rxjs';

const a$ = new Subject<string>(); const b$ = new Subject<number>();


combineLatest([a$, b$]).subscribe(console.log);


a$.next('X'); b$.next(1); a$.next('Y');
```

<details> <summary>Answer</summary>

```
['X', 1]
['Y', 1]
```

Nothing emits after `a$.next('X')` because `b$` hasn't emitted yet. After `b$.next(1)`, both have values, so it emits `['X', 1]`. After `a$.next('Y')`, it emits `['Y', 1]` (latest values). </details>

Common Mistakes

Mistake 1: Forgetting to Unsubscribe

```
// Wrong: memory leak
ngOnInit() { this.filters$.subscribe(filters => { / ... / }); }


// Right: cleanup on destroy ngOnDestroy() { this.subscription.unsubscribe(); }
```

Mistake 2: Nested Subscribes

```
// Wrong: callback hell, hard to manage

this.filters$.subscribe(filters =>
{ this.apiAdapter.fetchData(filters).subscribe(response => { this.results =
response.results; }); });



// Right: use operators to flatten this.results$ =
this.filters$.pipe( switchMap(filters => this.apiAdapter.fetchData(filters)),
map(response => response.results) );
```

Mistake 3: Missing shareReplay

```
// Wrong: each subscriber triggers separate API calls

this.results$ = this.filters$.pipe( switchMap(filters =>
this.apiAdapter.fetchData(filters)) );



// Right: share the result this.results$ = this.filters$.pipe( switchMap(filters =>
this.apiAdapter.fetchData(filters)), shareReplay(1) );
```

Key Takeaways

- **Observables model change over time** — Perfect for UI state that updates
- `switchMap` **cancels previous operations** — Use for API calls from user input
- `shareReplay(1)` **prevents duplicate work** — Always use when multiple components subscribe

---

Acceptance Criteria

This is a teaching section with no code changes. Criteria are conceptual:

- [ ] You can explain why Observables fit state management

- [ ] You know when to use `switchMap` vs other flattening operators
- [ ] You understand what `shareReplay(1)` does and why it matters
- [ ] You can implement proper cleanup with `takeUntil` or async pipe

---

Next Step

Proceed to `301-url-state-service.md` to create the first framework service using the RxJS patterns you just learned.

301: URL State Service

Status: Complete **Depends On:** 201-209 (Framework Models), 250-rxjs-patterns-primer **Blocks:** 303-request-coordinator, 306-resource-management-service

---

Learning Objectives

After completing this section, you will:

- Understand why the URL is the single source of truth for application state
- Know how to read and write URL query parameters in Angular
- Recognize the role of BehaviorSubject in providing synchronous access to asynchronous state
- Be able to implement bidirectional synchronization between application state and the URL

Objective

Create the `UrlStateService` that provides bidirectional synchronization between application state and URL query parameters. This service is the foundation of the URL-First State Management architecture.

Why

Every web application manages state. The question is: *where does that state live?*

Traditional approaches store state in:

- Component properties (lost on navigation)
- Services (lost on page refresh)
- LocalStorage (requires manual sync)

The URL-First approach stores state in the URL itself. This provides several benefits:

Feature	URL-First	Traditional State
Shareable	Copy URL to share exact application state	Users see different views
Bookmarkable	Browser bookmarks preserve state	Bookmarks lose context
Deep-linkable	External links open specific views	Links open default views
Back/Forward	Browser history "just works"	Requires manual history management
Refresh-safe	State survives page refresh	State is lost
Debuggable	State visible in browser URL bar	State hidden in memory

The Aha Moment

The URL is the single source of truth. Components react to URL changes, they don't control them.

When a user clicks a filter, the component doesn't update its own state. Instead, it updates the URL. The URL change triggers an observable emission, which the component subscribes to. The component always *reads* state from the URL.

This creates a unidirectional data flow:

```
User Action → URL Update → Observable Emission → Component Update → UI Render
```

Angular Router Integration

The Angular Router provides `ActivatedRoute.queryParams` for reading query parameters. However, there's a subtlety: `ActivatedRoute` is scoped to the route hierarchy. A root-level service doesn't receive query param updates from child routes.

`UrlStateService` solves this by:

- Using `Router.events` to watch all navigation events globally
- Parsing query parameters from the router's URL directly
- Emitting changes through a `BehaviorSubject` that any component can subscribe to

RxJS Patterns Used

This service uses patterns from Interlude B (Section 250):

Pattern	Usage
`BehaviorSubject`	Holds current URL params with synchronous access via `.value`
`filter()`	Only process `NavigationEnd` events
`map()`	Transform navigation events to query param objects
`distinctUntilChanged()`	Prevent duplicate emissions for identical params

What

Step 301.1: Create the Services Directory Index

Before creating individual service files, create a barrel export file that will aggregate all service exports.

Create the file `src/app/framework/services/index.ts`:

```
// src/app/framework/services/index.ts
// VERSION 1 (Section 301) - Barrel file for framework services



// This barrel file exports all framework services. // Import from '@app/framework/
services' instead of individual files.



export * from './url-state.service';
```

Delete the `.gitkeep` file since the directory now has real content:

```
$ rm src/app/framework/services/.gitkeep
```

Step 301.2: Create the URL State Service

Create the file `src/app/framework/services/url-state.service.ts`:

```
// src/app/framework/services/url-state.service.ts
// VERSION 1 (Section 301) - URL-First state management foundation


import { Injectable, NgZone } from '@angular/core'; import { Router, ActivatedRoute,
Params, NavigationEnd } from '@angular/router'; import { Observable, BehaviorSubject }
from 'rxjs'; import { map, distinctUntilChanged, filter } from 'rxjs/operators';



/**


  • Domain-agnostic URL state management service


 *


  • Provides bidirectional synchronization between application state and URL query
    parameters.



  • The URL serves as the single source of truth for application state.


 *


  • Key Design Decisions:


 *


  • 1. Uses Router.events instead of ActivatedRoute.queryParams because this is a



  •    root-level singleton service. ActivatedRoute at root level doesn't receive



  •    query param updates from child routes (like /discover).
```

```
 *
  • 2. Uses Router.url and parseUrl() instead of ActivatedRoute.snapshot.queryParams
  •    for the same reason — to capture the full URL including child route params.
 *
  • 3. Uses BehaviorSubject to provide:
  •    - Synchronous access to current params via getParams()
  •    - Observable stream for reactive updates via watchParams()
  •    - Immediate value emission to new subscribers
 *
  • @example
  •
```

typescript

• // In a component
• constructor(private urlState: UrlStateService) {
• // Watch for changes reactively
• this.urlState.watchParams<MyFilters>().subscribe(filters => {
• console.log('Filters changed:', filters);
• });

*

- // Get current value synchronously
- const current = this.urlState.getParams<MyFilters>();

*

- // Update URL (triggers navigation)
- this.urlState.setParams({ page: 2, search: 'ford' });
- }

```
 /
@Injectable({ providedIn: 'root' }) export class UrlStateService { /**


  • BehaviorSubject holding current URL query parameters

   *

  • BehaviorSubject is used instead of Subject because:


  • 1. It holds the current value, enabling synchronous getParams()


  • 2. New subscribers immediately receive the current value


  • 3. It requires an initial value (empty object)

   */
private paramsSubject = new BehaviorSubject<Params>({});


/**


  • Public observable stream of URL query parameters

   *

  • Components subscribe to this to react to URL changes.

   */
public params$: Observable<Params> = this.paramsSubject.asObservable();
```

```
/**


  • Constructor - sets up URL synchronization

   *

  • @param router - Angular Router for navigation and URL parsing


  • @param route - ActivatedRoute for relative navigation


  • @param ngZone - NgZone for ensuring change detection runs

   */
constructor( private router: Router, private route: ActivatedRoute, private ngZone:
NgZone ) { // Initialize from current URL on service creation
this.initializeFromRoute();


// Watch for all future URL changes this.watchRouteChanges(); }


/**


  • Get current URL query parameters as a typed object

   *

  • Provides synchronous access to the current URL state.


  • Use this when you need the current value without subscribing.
```

```
 *
• @template TParams - The shape of the parameters object


• @returns Current query parameters cast to TParams
 *
• @example


•
```

typescript

• interface MyFilters {

• search: string;

• page: number;

• }

*

• const filters = urlState.getParams<MyFilters>();

• console.log(filters.search); // Type-safe access

```
   /
getParams<TParams = Params>(): TParams { return this.paramsSubject.value as TParams; }


/**


  • Update URL query parameters

   *

  • Performs a shallow merge with existing parameters and navigates to the new URL.


  • Use null or undefined to remove a parameter.

   *

  • @template TParams - The shape of the parameters object


  • @param params - Partial parameters to update


  • @param replaceUrl - If true, replaces current history entry instead of pushing


  • @returns Promise that resolves when navigation completes

   *

  • @example


  •
```

typescript

- // Update specific params
- await urlState.setParams({ page: 2, search: 'test' });

*

- // Remove a param by setting to null
- await urlState.setParams({ search: null });

*

- // Replace history entry (no new back button entry)
- await urlState.setParams({ page: 1 }, true);

```
   /
async setParams<TParams = Params>( params: Partial<TParams>, replaceUrl = false ):
Promise<boolean> { const currentParams = this.paramsSubject.value; const mergedParams
= { ...currentParams };



// Merge new params, removing null/undefined values Object.keys(params).forEach(key =>
{ const value = (params as any)[key]; if (value === null || value === undefined)
{ delete mergedParams[key]; } else { mergedParams[key] = value; } });



return await this.router.navigate([], { relativeTo: this.route, queryParams:
mergedParams, replaceUrl, queryParamsHandling: '' // Use exact params, don't
preserve }); }



/**



  • Watch URL query parameters as an observable stream

   *

  • Returns an observable that emits whenever URL query parameters change.



  • Uses distinctUntilChanged to prevent duplicate emissions.

   *

  • @template TParams - The shape of the parameters object



  • @returns Observable of query parameters

   *

  • @example
```

```
•
```

typescript

• urlState.watchParams<MyFilters>().subscribe(filters => {

• console.log('URL changed:', filters);

• this.loadData(filters);

• });

```
   /
watchParams<TParams = Params>(): Observable<TParams> { return
this.params$.pipe( map(params => params as TParams), distinctUntilChanged((a, b) =>
JSON.stringify(a) === JSON.stringify(b)) ); }



/**



  • Clear all URL query parameters

   *

  • Navigates to the current path with empty query string.

   *

  • @param replaceUrl - If true, replaces current history entry



  • @returns Promise that resolves when navigation completes

   */
async clearParams(replaceUrl = false): Promise<boolean> { return
this.router.navigate([], { relativeTo: this.route, queryParams: {}, replaceUrl }); }



/**



  • Get a specific query parameter value

   *

  • @param key - Parameter key
```

```
   • @returns Parameter value or null if not found

    */
getParam(key: string): any { return this.paramsSubject.value[key] || null; }


/**

   • Set a specific query parameter

    *

   • Convenience method for updating a single parameter.

    *

   • @param key - Parameter key


   • @param value - Parameter value (null to remove)


   • @param replaceUrl - If true, replaces current history entry


   • @returns Promise that resolves when navigation completes

    */
async setParam( key: string, value: any, replaceUrl = false ): Promise<boolean>
{ return this.setParams({ [key]: value } as any, replaceUrl); }


/**
```

```
   • Check if a specific parameter exists in the URL
    *
   • @param key - Parameter key
   • @returns True if parameter exists
    */
hasParam(key: string): boolean { return key in this.paramsSubject.value; }

/**
   • Watch a specific parameter for changes
    *
   • @param key - Parameter key to watch
   • @returns Observable of parameter value
    */
watchParam(key: string): Observable<any> { return this.params$.pipe( map(params =>
params[key] || null), distinctUntilChanged() ); }

/**
   • Serialize parameters to URL query string
```

```
   *

  • Utility method for converting params object to query string format.

   *

  • @param params - Parameters object



  • @returns Query string (without leading '?')

   */
serializeParams(params: Params): string { const queryParams = new URLSearchParams();



Object.keys(params).forEach(key => { const value = params[key];



// Skip null/undefined if (value === null || value === undefined) { return; }



// Handle arrays (comma-separated) if (Array.isArray(value)) { queryParams.set(key,
value.join(',')); return; }



// Convert to string queryParams.set(key, String(value)); });



return queryParams.toString(); }



/**



  • Deserialize URL query string to parameters object
```

```
   *

  • Utility method for parsing query string to params object.



  • Automatically converts numeric and boolean strings.

   *

  • @param queryString - Query string (with or without leading '?')



  • @returns Parameters object

   */
deserializeParams(queryString: string): Params { const params: Params = {}; const
urlParams = new URLSearchParams( queryString.startsWith('?') ? queryString.slice(1) :
queryString );



urlParams.forEach((value, key) => { // Try to parse as number if (!
isNaN(Number(value)) && value !== '') { params[key] = Number(value); return; }



// Try to parse as boolean if (value === 'true' || value === 'false') { params[key] =
value === 'true'; return; }



// Check for comma-separated values (arrays) if (value.includes(',')) { params[key] =
value.split(','); return; }



// Keep as string params[key] = value; });



return params; }
```

```
/**


  • Initialize from current route

   *

  • Called once in constructor to set initial state from URL.

   *

  • IMPORTANT: Uses router.url instead of route.snapshot because UrlStateService


  • is a root singleton. Root-level ActivatedRoute.snapshot may not have query


  • params from child routes (like /discover). router.url contains the full URL.

   */
private initializeFromRoute(): void { const params = this.extractQueryParams();
this.ngZone.run(() => { this.paramsSubject.next(params); }); }


/**


  • Watch for route changes and update internal state

   *

  • Sets up subscription to Router.events for ongoing synchronization.

   *

  • IMPORTANT: Uses Router.events instead of ActivatedRoute.queryParams because
```

```
  • UrlStateService is a root singleton. Root-level ActivatedRoute doesn't receive



  • query param updates from child routes (like /discover). Router.events is global



  • and captures all navigation events including query parameter changes.

   */
private watchRouteChanges(): void { this.router.events .pipe( // Only process
NavigationEnd events (route change complete) filter((event): event is NavigationEnd =>
event instanceof NavigationEnd), // Extract query params from the new URL map(() =>
this.extractQueryParams()), // Only emit if params actually changed
distinctUntilChanged((a, b) => JSON.stringify(a) ===
JSON.stringify(b)) ) .subscribe(params => { // Use NgZone.run to ensure change
detection runs this.ngZone.run(() => { this.paramsSubject.next(params); }); }); }



/**



  • Extract query parameters from current router state

   *

  • Uses router.parseUrl() to reliably extract query params



  • regardless of the route hierarchy.

   */
private extractQueryParams(): Params { const urlTree =
this.router.parseUrl(this.router.url); return urlTree.queryParams; } }
```

Step 301.3: Update the Barrel File

Update `src/app/framework/services/index.ts` to ensure the export is correct:

```
// src/app/framework/services/index.ts
// VERSION 1 (Section 301) - Barrel file for framework services


export * from './url-state.service';
```

Verification

1. Check File Exists

```
$ ls -la src/app/framework/services/
```

Expected output:

```
total 12
drwxr-xr-x 2 user user 4096 Feb  9 12:00 . drwxr-xr-x 5 user user 4096 Feb  9 12:00 ..
-rw-r--r-- 1 user user  123 Feb  9 12:00 index.ts -rw-r--r-- 1 user user 8456 Feb  9
12:00 url-state.service.ts
```

2. TypeScript Compilation Check

```
$ cd ~/projects/vvroom
$ npx tsc --noEmit src/app/framework/services/url-state.service.ts
```

Expected: No output (no compilation errors).

3. Verify Service Injectability

Add a temporary test to `src/app/app.component.ts`:

```
// src/app/app.component.ts

// TEMPORARY TEST - Remove after verification



import { Component, OnInit } from '@angular/core'; import { UrlStateService } from './
framework/services';



@Component({ selector: 'app-root', templateUrl: './app.component.html', styleUrls:
['./app.component.scss'] }) export class AppComponent implements OnInit { title =
'vvroom';



constructor(private urlState: UrlStateService) {}



ngOnInit(): void { // Log URL changes to console
this.urlState.watchParams().subscribe(params => { console.log('[UrlStateService] URL
params:', params); }); } }
```

4. Run the Application

```
$ ng serve
```

Open browser to `http://localhost:4200` and:

- Open browser DevTools (F12) and go to Console tab
- Navigate to `http://localhost:4200?search=ford&page=1`
- You should see in console:
  `[UrlStateService] URL params: {search: 'ford', page: '1'}`
- Modify the URL manually to `http://localhost:4200?search=toyota&page=2`
- Console should show the new params

5. Verify Browser History

- Navigate through several URL param changes
- Click browser Back button
- Console should show params reverting to previous values
- This confirms URL-First state management is working

After verification, remove the temporary test code from AppComponent.

Common Problems

Symptom	Cause	Solution
`Cannot find module '@angular/router'`	RouterModule not imported	Ensure RouterModule is in app.module.ts imports
Console shows empty object `{}`	Route not configured	Ensure routes exist in app-routing.module.ts
Params not updating on child route navigation	Wrong ActivatedRoute	Using Router.events (as implemented) solves this
Change detection not running	Missing NgZone	Ensure ngZone.run() wraps paramsSubject.next()
Duplicate console logs	Multiple emissions	distinctUntilChanged() should prevent this
URL not updating on setParams()	Navigation blocked	Check for route guards or canDeactivate

---

Key Takeaways

- **The URL is the single source of truth** — Components read state from the URL, they don't maintain their own copy
- **BehaviorSubject enables both sync and async access** — `.value` for sync, `.asObservable()` for reactive streams

- **Root services need special URL handling** — Use `Router.events` and `router.parseUrl()` instead of `ActivatedRoute.queryParams`

Acceptance Criteria

- [ ] `src/app/framework/services/url-state.service.ts` exists with complete implementation
- [ ] `src/app/framework/services/index.ts` exports the service
- [ ] Service is `@Injectable({ providedIn: 'root' })` for singleton behavior
- [ ] `getParams<T>()` returns typed URL parameters synchronously
- [ ] `setParams<T>()` updates URL and triggers navigation
- [ ] `watchParams<T>()` returns observable stream of URL changes
- [ ] `distinctUntilChanged()` prevents duplicate emissions
- [ ] Service handles child route query params correctly via Router.events
- [ ] NgZone.run() ensures Angular change detection runs
- [ ] TypeScript compilation succeeds with no errors
- [ ] JSDoc comments document all public methods

Next Step

Proceed to `302-api-service.md` to create the service for making HTTP requests to the API.

302: API Service

Status: Complete **Depends On:** 206-api-response-interface **Blocks:** 303-request-coordinator, 310-filter-options-service, 502-api-adapter

---

Learning Objectives

After completing this section, you will:

- Understand the role of a thin HTTP wrapper service in Angular applications
- Know how to serialize query parameters for HTTP requests
- Recognize when to handle errors at the service level vs. propagate them
- Be able to implement a domain-agnostic API service

Objective

Create the `ApiService` that provides a type-safe, domain-agnostic wrapper around Angular's `HttpClient`. This service handles common patterns like query parameter serialization, error transformation, and response typing.

Why

Every Angular application needs to make HTTP requests. The question is: *how do we structure these requests?*

Option 1: Direct HttpClient Usage

```
// In component

this.http.get<Vehicle[]>('/api/vehicles', { params: new HttpParams() .set('page',
'1') .set('size', '20') .set('manufacturer', 'Ford,Toyota') }).subscribe(data => { //
handle response });
```

This works, but has problems:

- Verbose param building repeated everywhere
- Error handling duplicated in each call
- No consistent typing strategy
- Hard to add cross-cutting concerns (logging, auth)

Option 2: Domain-Specific Services

```
// In VehicleService

getVehicles(filters: VehicleFilters): Observable<Vehicle[]> { return
this.http.get<Vehicle[]>('/api/vehicles', { params: this.buildParams(filters) }); }
```

Better, but now every domain needs its own service with repeated boilerplate.

Option 3: Thin HTTP Wrapper (Our Approach)

```
// ApiService provides thin wrapper

this.api.get<Vehicle>('/api/vehicles', { params: filters })



// Domain adapter uses it this.api.get<Vehicle>(this.baseUrl + '/vehicles', { params:
filters })
```

`ApiService` handles:

- Query parameter serialization (including arrays)
- Error transformation to consistent format
- Response typing

Domain adapters (created in Phase 5) use `ApiService` for domain-specific logic.

Why Not More Abstraction?

You might wonder: *Why not build a full data layer with repositories, unit of work, etc.?*

The answer is **YAGNI (You Aren't Gonna Need It)**. The vvroom application:

- Reads data from a REST API
- Displays it in tables and charts
- Filters via URL parameters

A thin HTTP wrapper is all we need. Adding more abstraction would increase complexity without benefit.

Error Handling Strategy

`ApiService` catches errors and transforms them into consistent Error objects. It does NOT:

- Show UI notifications (that's the component's job)
- Retry failed requests (that's `RequestCoordinator`'s job)
- Log to external services (that's the error handler's job)

Each layer has its responsibility.

What

Step 302.1: Create the API Service

Create the file `src/app/framework/services/api.service.ts`:

```
// src/app/framework/services/api.service.ts

// VERSION 1 (Section 302) - Domain-agnostic HTTP wrapper



import { Injectable } from '@angular/core'; import { HttpClient, HttpParams,
HttpErrorResponse } from '@angular/common/http'; import { Observable, throwError }
from 'rxjs'; import { catchError, map } from 'rxjs/operators'; import { ApiResponse,
StandardApiResponse } from '../models/api-response.interface';



/**



  • Options for API requests


 *


  • Provides a simplified interface for common HTTP options.


 */

export interface ApiRequestOptions { /**



  • Query parameters to append to the URL



  • Arrays are serialized as comma-separated values


   */

params?: Record<string, any>;



/**



  • HTTP headers to include in the request
```

```
   */
headers?: Record<string, string>;


/**


  • Whether to include credentials (cookies) in the request

   */
withCredentials?: boolean;


/**


  • Response type (default: 'json')

   */
responseType?: 'json' | 'text' | 'blob' | 'arraybuffer'; }


/**


  • Domain-agnostic API service for making HTTP requests

 *

  • This service provides a thin, type-safe wrapper around Angular's HttpClient.


  • It handles common patterns like query parameter serialization and error handling.

 *

  • Design Philosophy:
```

```
*
 • 1. Thin wrapper - Doesn't add unnecessary abstraction over HttpClient
 • 2. Type-safe - Generic methods return typed observables
 • 3. Domain-agnostic - No domain-specific logic here
 • 4. Error transformation - Converts HTTP errors to consistent Error objects
*
 • Domain-specific logic (like URL construction, response transformation) belongs
 • in domain adapters (created in Phase 5), not here.
*
 • @example
 •
```

typescript

- // GET request with pagination
- this.api.get<VehicleResult>('/api/vehicles', {
- params: { page: 1, size: 20, manufacturer: ['Ford', 'Toyota'] }
- }).subscribe(response => {
- console.log(response.results);
- });

*

- // POST request
- this.api.post<VehicleResult>('/api/vehicles', {
- manufacturer: 'Ford',
- model: 'F-150'
- }).subscribe(vehicle => {
- console.log('Created:', vehicle);
- });

```
 /
@Injectable({ providedIn: 'root' }) export class ApiService { /**


  • Constructor - injects HttpClient

   *

  • Using readonly to ensure the dependency isn't reassigned.

   */
constructor(private readonly http: HttpClient) {}


/**


  • Execute a GET request

   *

  • Returns the standard paginated API response format.

   *

  • @template TData - The type of items in the response


  • @param endpoint - API endpoint (absolute or relative URL)


  • @param options - Request options (params, headers, etc.)


  • @returns Observable of paginated response
```

```
   *
  • @example
  •
```

typescript

- this.api.get<VehicleResult>('/api/vehicles', {
- params: { page: 1, manufacturer: 'Ford' }
- }).subscribe(response => {
- console.log(`Found ${response.total} vehicles`);
- console.log('Results:', response.results);
- });

```
   /
get<TData>( endpoint: string, options?: ApiRequestOptions ):
Observable<ApiResponse<TData>> { const httpParams =
this.buildHttpParams(options?.params);



return this.http .get<ApiResponse<TData>>(endpoint, { params: httpParams, headers:
options?.headers, withCredentials:
options?.withCredentials }) .pipe(catchError(this.handleError)); }



/**



  • Execute a POST request

   *

  • @template TData - The type of response data



  • @param endpoint - API endpoint (absolute or relative URL)



  • @param body - Request body



  • @param options - Request options (params, headers, etc.)



  • @returns Observable of response data

   */
post<TData>( endpoint: string, body: any, options?: ApiRequestOptions ):
Observable<TData> { const httpParams = this.buildHttpParams(options?.params);
```

```
return this.http .post<TData>(endpoint, body, { params: httpParams, headers:
options?.headers, withCredentials:
options?.withCredentials }) .pipe(catchError(this.handleError)); }



/**



  • Execute a PUT request

   *

  • @template TData - The type of response data



  • @param endpoint - API endpoint (absolute or relative URL)



  • @param body - Request body



  • @param options - Request options (params, headers, etc.)



  • @returns Observable of response data

   */
put<TData>( endpoint: string, body: any, options?: ApiRequestOptions ):
Observable<TData> { const httpParams = this.buildHttpParams(options?.params);



return this.http .put<TData>(endpoint, body, { params: httpParams, headers:
options?.headers, withCredentials:
options?.withCredentials }) .pipe(catchError(this.handleError)); }



/**
```

```
  • Execute a PATCH request
   *
  • @template TData - The type of response data
  • @param endpoint - API endpoint (absolute or relative URL)
  • @param body - Request body
  • @param options - Request options (params, headers, etc.)
  • @returns Observable of response data
   */
patch<TData>( endpoint: string, body: any, options?: ApiRequestOptions ):
Observable<TData> { const httpParams = this.buildHttpParams(options?.params);

return this.http .patch<TData>(endpoint, body, { params: httpParams, headers:
options?.headers, withCredentials:
options?.withCredentials }) .pipe(catchError(this.handleError)); }

/**
  • Execute a DELETE request
   *
  • @template TData - The type of response data
```

```
  • @param endpoint - API endpoint (absolute or relative URL)

  • @param options - Request options (params, headers, etc.)

  • @returns Observable of response data

   */
delete<TData>( endpoint: string, options?: ApiRequestOptions ): Observable<TData>
{ const httpParams = this.buildHttpParams(options?.params);

return this.http .delete<TData>(endpoint, { params: httpParams, headers:
options?.headers, withCredentials:
options?.withCredentials }) .pipe(catchError(this.handleError)); }

/**

  • Execute a GET request that returns a standard success/error response

   *

  • Some APIs wrap responses in a success/error envelope:

  •
```

json

- { "success": true, "data": { ... } }
- { "success": false, "error": { "message": "..." } }

```
   • This method unwraps successful responses and throws on error.

    *

   • @template TData - The type of data in the response



   • @param endpoint - API endpoint (absolute or relative URL)



   • @param options - Request options (params, headers, etc.)



   • @returns Observable of response data (unwrapped from success envelope)

    */
getStandard<TData>( endpoint: string, options?: ApiRequestOptions ): Observable<TData>
{ const httpParams = this.buildHttpParams(options?.params);



return this.http .get<StandardApiResponse<TData>>(endpoint, { params: httpParams,
headers: options?.headers, withCredentials:
options?.withCredentials }) .pipe( map(response => { if (response.success) { return
response.data; } else { throw new Error(response.error.message); } }),
catchError(this.handleError) ); }



/**



   • Build HttpParams from a plain object

    *

   • Handles:
```

```
  • - Null/undefined value filtering (skipped)


  • - Array serialization (comma-separated)


  • - Boolean/number to string conversion
   *
  • @param params - Plain object of query parameters


  • @returns HttpParams instance
   *
  • @example


  •
```

typescript

- // Input
- { page: 1, manufacturer: ['Ford', 'Toyota'], active: true, empty: null }

*

- // Output HttpParams
- page=1&manufacturer=Ford,Toyota&active=true
- // Note: 'empty' is skipped because it's null

```
   /

private buildHttpParams(params?: Record<string, any>): HttpParams { let httpParams =
new HttpParams();



if (!params) { return httpParams; }



Object.keys(params).forEach(key => { const value = params[key];



// Skip null/undefined values if (value === null || value === undefined) { return; }



// Handle arrays (comma-separated) if (Array.isArray(value)) { httpParams =
httpParams.set(key, value.join(',')); return; }



// Convert to string httpParams = httpParams.set(key, String(value)); });



return httpParams; }



/**



  • Handle HTTP errors

   *

  • Transforms HttpErrorResponse into a consistent Error format.



  • Logs the error for debugging, then re-throws for upstream handling.
```

```
   *

  • @param error - HTTP error response



  • @returns Observable that throws a formatted error


   */
private handleError(error: HttpErrorResponse): Observable<never> { let errorMessage =
'An unknown error occurred';



if (error.error instanceof ErrorEvent) { // Client-side or network error errorMessage
= `Network error: ${error.error.message}`; } else { // Backend error if
(error.error?.error?.message) { // Structured error response: { error: { message:
"..." } } errorMessage = error.error.error.message; } else if (error.error?.message)
{ // Simple error response: { message: "..." } errorMessage = error.error.message; }
else if (error.message) { // HttpErrorResponse message errorMessage = error.message; }
else { // Fallback to status errorMessage = `Server error: ${error.status} $
{error.statusText}`; } }



console.error('API Error:', errorMessage, error);



return throwError(() => new Error(errorMessage)); } }
```

Step 302.2: Update the Barrel File

Update `src/app/framework/services/index.ts` to export the new service:

```
// src/app/framework/services/index.ts
// VERSION 2 (Section 302) - Added ApiService


export * from './url-state.service'; export * from './api.service';
```

Step 302.3: Ensure HttpClientModule is Imported

The `ApiService` depends on `HttpClient`, which requires `HttpClientModule` to be imported in the app module.

Verify `src/app/app.module.ts` includes:

```
// src/app/app.module.ts
// Ensure HttpClientModule is imported



import { NgModule } from '@angular/core'; import { BrowserModule } from '@angular/
platform-browser'; import { HttpClientModule } from '@angular/common/http';  // <--
Required



import { AppRoutingModule } from './app-routing.module'; import { AppComponent } from
'./app.component';



@NgModule({ declarations: [ AppComponent ], imports: [ BrowserModule,
AppRoutingModule, HttpClientModule  // <-- Must be here ], providers: [], bootstrap:
[AppComponent] }) export class AppModule { }
```

If `HttpClientModule` is missing, add it as shown.

Verification

1. Check File Exists

```
$ ls -la src/app/framework/services/
```

Expected output includes:

```
-rw-r--r-- 1 user user 7234 Feb  9 12:00 api.service.ts
```

2. TypeScript Compilation Check

```
$ cd ~/projects/vvroom
$ npx tsc --noEmit src/app/framework/services/api.service.ts
```

Expected: No output (no compilation errors).

3. Build the Application

```
$ ng build
```

Expected: Build succeeds with no errors.

4. Verify in Browser (Optional)

If you have a working API endpoint, you can test:

```
// Temporary test in any component
constructor(private api: ApiService) { this.api.get<any>('http://generic-
prime.minilab/api/specs/v1/automobiles', { params: { page: 1, size:
5 } }).subscribe({ next: response => console.log('API Response:', response), error:
err => console.error('API Error:', err) }); }
```

Common Problems

Symptom	Cause	Solution
`NullInjectorError: No provider for HttpClient`	HttpClientModule not imported	Add HttpClientModule to app.module.ts imports
`Cannot find module '../models/api-response.interface'`	Interface file missing	Create it in Section 206 first
CORS errors in browser	API doesn't allow cross-origin	Configure API server or use proxy
`Type 'Observable<Object>' is not assignable to...`	Generic type mismatch	Ensure response type matches expected type
Arrays serialized as `[object Object]`	Nested objects in params	Flatten objects before passing to params

---

Key Takeaways

- **Thin wrappers reduce boilerplate** — ApiService handles serialization and errors so each call site doesn't have to
- **Arrays serialize as comma-separated values** — This matches common REST API conventions
- **Error handling transforms, doesn't swallow** — Errors are logged and re-thrown for upstream handling

---

Acceptance Criteria

- [ ] `src/app/framework/services/api.service.ts` exists with complete implementation
- [ ] Barrel file exports the service
- [ ] Service is `@Injectable({ providedIn: 'root' })`
- [ ] `get<T>()` method returns typed Observable
- [ ] `post<T>()`, `put<T>()`, `patch<T>()`, `delete<T>()` methods implemented
- [ ] `getStandard<T>()` unwraps success/error envelope responses
- [ ] `buildHttpParams()` correctly serializes arrays as comma-separated
- [ ] `handleError()` transforms errors consistently

- [ ] HttpClientModule is imported in app.module.ts
- [ ] TypeScript compilation succeeds
- [ ] JSDoc comments document all public methods

Next Step

Proceed to `303-request-coordinator.md` to create the service that coordinates API requests with caching and deduplication.

303: Request Coordinator Service

Status: Complete **Depends On:** 302-api-service **Blocks:** 306-resource-management-service

---

Learning Objectives

After completing this section, you will:

- Understand the three-layer request processing model (cache, dedup, HTTP)
- Know how to implement TTL-based response caching
- Recognize the in-flight deduplication pattern for preventing duplicate requests
- Be able to implement exponential backoff retry logic

Objective

Create the `RequestCoordinatorService` that provides intelligent request coordination with three processing layers: response caching, in-flight deduplication, and HTTP execution with retry.

Why

When users interact with the vvroom application, they generate many API requests:

- Changing filters → new data fetch
- Navigating between pages → data fetch
- Browser back/forward → URL change → data fetch
- Pop-out windows → parallel data fetch

Without coordination, these requests can:

- **Overwhelm the API** — Same request sent multiple times
- **Waste bandwidth** — Re-fetching data that hasn't changed
- **Create race conditions** — Responses arriving out of order
- **Fail silently** — Transient network errors losing data

`RequestCoordinatorService` solves these problems with a three-layer approach:

Layer 1: Response Cache (TTL-Based)

```
Request comes in → Check cache → If fresh, return cached → Skip HTTP
```

- Stores successful responses with timestamp
- Returns cached response if within TTL
- Reduces API load for repeated queries

Layer 2: In-Flight Deduplication

```
Request comes in → Check if same request in progress → If yes, share observable → One
HTTP call serves many subscribers
```

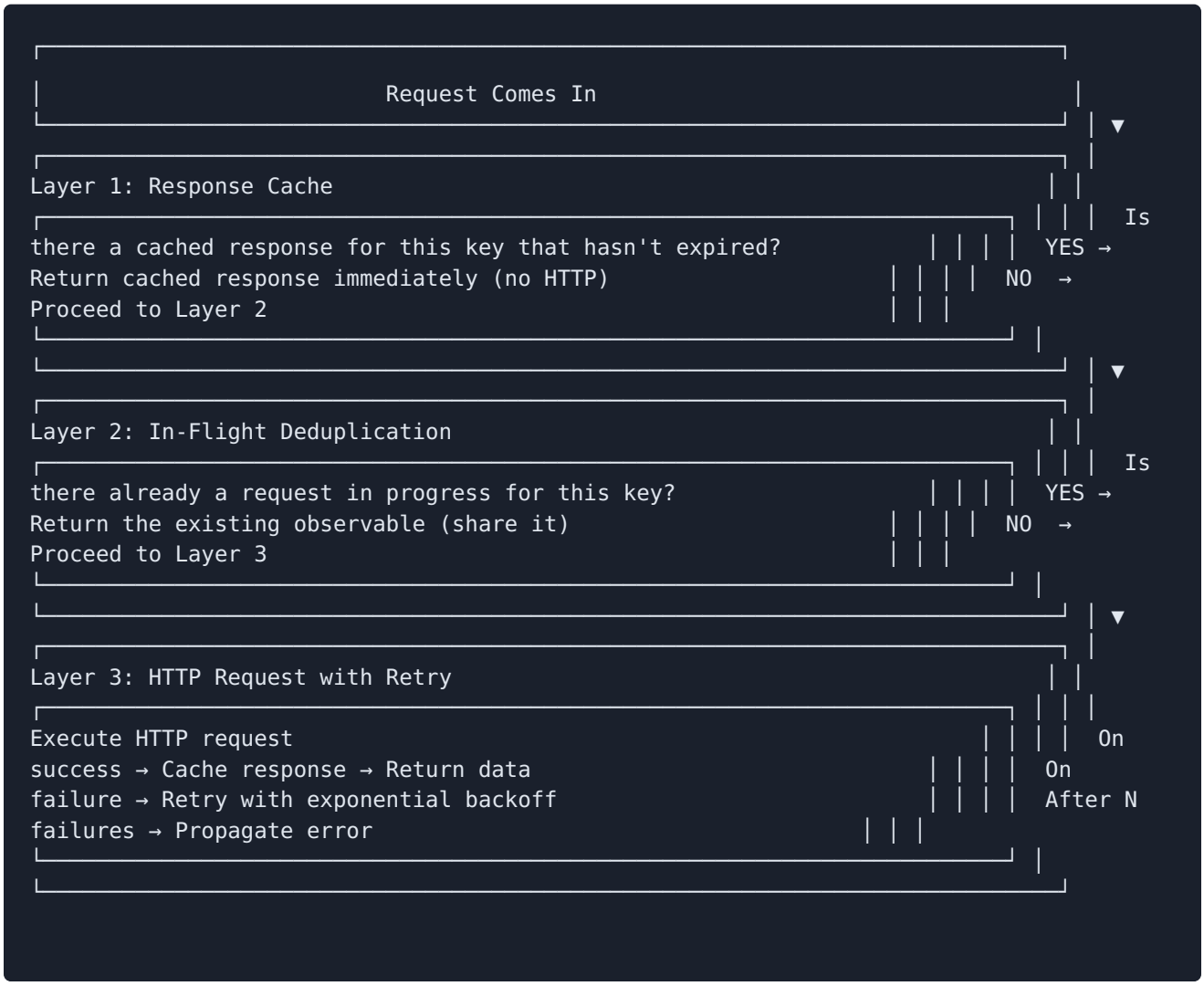
- Tracks ongoing requests by key
- Multiple calls to same endpoint share one observable
- Prevents duplicate concurrent requests

Layer 3: HTTP Request with Retry

```
Execute HTTP → On failure, wait → Retry with exponential backoff → After N failures,
give up
```

- Executes the actual HTTP request
- Retries transient failures with exponential backoff
- Gives up after configurable number of attempts

Visual Flow



RxJS Patterns Used

Pattern	Usage
`BehaviorSubject`	Tracks loading states per request key
`shareReplay(1)`	Shares observable among multiple subscribers
`retry()`	Retries failed requests with custom delay
`timer()`	Creates delay between retries
`finalize()`	Cleanup when observable completes or errors
`tap()`	Side effect to cache successful responses

What

Step 303.1: Create the Request Coordinator Service

Create the file `src/app/framework/services/request-coordinator.service.ts`:

```
// src/app/framework/services/request-coordinator.service.ts

// VERSION 1 (Section 303) - Three-layer request coordination



import { Injectable } from '@angular/core'; import { Observable, BehaviorSubject,
timer } from 'rxjs'; import { tap, finalize, shareReplay, retry, map,
distinctUntilChanged } from 'rxjs/operators';



/**



  • Configuration options for request coordination


 */

export interface RequestConfig { /**



  • Cache time-to-live in milliseconds



  • Responses older than this are considered stale and refetched



  • @default 30000 (30 seconds)


   */

cacheTTL?: number;



/**



  • Number of retry attempts on failure
```

```
  • Set to 0 to disable retries


  • @default 3

   */
retryAttempts?: number;


/**


  • Initial retry delay in milliseconds

    Subsequent retries use exponential backoff (delay  2^attempt)

  • @default 1000 (1 second)

   */
retryDelay?: number;


/**


  • Force bypass cache and execute fresh request


  • Use for "refresh" functionality


  • @default false

   */
skipCache?: boolean; }
```

```
/**
```

- Cache entry structure

```
 */
interface CacheEntry<T> { /* Cached response data */ data: T; /* Timestamp when cached (ms since epoch) */ timestamp: number; /* Time-to-live in milliseconds */ ttl: number; }

/**
```

- Domain-agnostic request coordination service

```
 *
```

- Provides three-layer request processing:

- 1. **Response Cache** (TTL-based) - Return cached response if fresh

- 2. **In-Flight Deduplication** - Share ongoing requests

- 3. **HTTP Request with Retry** - Execute with exponential backoff

```
 *
```

- **Why This Matters:**

```
 *
```

- Without coordination:

```
  • - User changes filter → 5 requests fire → 5 responses arrive → UI flickers

  • - User hits back button → request for data we just had → wasted bandwidth

  • - Network blip → request fails → data lost

*

  • With coordination:

  • - User changes filter → 1 request fires → 1 response → smooth UI

  • - User hits back → cache hit → instant data → no network call

  • - Network blip → retry → data recovered

*

  • @example

  •
```

typescript

- // Basic usage
- this.coordinator.execute(
- 'vehicles-page-1',
- () => this.api.get<Vehicle[]>('/vehicles?page=1')
-).subscribe(vehicles => {
- console.log('Vehicles:', vehicles);
- });

*

- // With custom configuration
- this.coordinator.execute(
- 'stats-latest',
- () => this.api.get<Stats>('/stats'),
- { cacheTTL: 60000, retryAttempts: 5 }
-).subscribe(stats => {
- console.log('Statistics:', stats);
- });

*

- // Force fresh fetch (skip cache)
- this.coordinator.execute(
- 'vehicles-page-1',
- () => this.api.get<Vehicle[]>('/vehicles?page=1'),
- { skipCache: true }
-).subscribe();

```
 /
@Injectable({ providedIn: 'root' }) export class RequestCoordinatorService { /**


  • Response cache storage

   *

  • Maps request keys to CacheEntry objects containing:


  • - Cached data


  • - Timestamp when cached


  • - TTL for this entry

   */
private cache = new Map<string, CacheEntry<any>>();


/**


  • In-flight requests storage

   *

  • Maps request keys to Observable instances of ongoing HTTP requests.


  • If a second request comes in with the same key while the first is
```

```
  • still in progress, we return the same observable (deduplication).
   */
private inFlightRequests = new Map<string, Observable<any>>();

/**
  • Loading state per request key
   *
  • Tracks which requests are currently loading.

  • Components can subscribe to know when specific data is loading.
   */
private loadingStateSubject = new BehaviorSubject<Map<string, boolean>>( new Map() );

/**
  • Observable of loading states
   *
  • Emits Map of request keys to loading booleans.

  • Use getLoadingState$() or getGlobalLoading$() for filtered access.
   */
public loadingState$ = this.loadingStateSubject.asObservable();
```

```
/**
  • Execute HTTP request with caching, deduplication, and retry
   *
  • The three-layer processing:
  • 1. Check cache (return immediately if fresh)
  • 2. Check in-flight (share if same request in progress)
  • 3. Execute HTTP with retry and caching
   *
  • @template T - Response type
  • @param requestKey - Unique identifier for this request
  • @param requestFn - Function that returns Observable (lazy execution)
  • @param config - Optional request configuration
  • @returns Observable of response data (shared, cached, deduplicated)
   */
```

```
execute<T>( requestKey: string, requestFn: () => Observable<T>, config?:
RequestConfig ): Observable<T> { const effectiveConfig =
this.getEffectiveConfig(config);



// Layer 1: Check response cache (unless skipCache) if (!effectiveConfig.skipCache)
{ const cachedResponse = this.getCachedResponse<T>(requestKey); if (cachedResponse !==
null) { // Return cached data wrapped in observable return new Observable(observer =>
{ observer.next(cachedResponse); observer.complete(); }); } }



// Layer 2: Check in-flight requests (deduplication) if
(this.inFlightRequests.has(requestKey)) { // Return existing observable (same request
in progress) return this.inFlightRequests.get(requestKey)!; }



// Layer 3: Execute HTTP request with retry this.setLoadingState(requestKey, true);



const request$ = requestFn().pipe( // Retry with exponential backoff retry({ count:
effectiveConfig.retryAttempts!, delay: (error, retryCount) => { const delay =
effectiveConfig.retryDelay! * Math.pow(2, retryCount - 1); console.log(
[RequestCoordinator] Retry ${retryCount} for ${requestKey} in ${delay}ms );
return timer(delay); } }), // Cache successful response tap(data => { if (!
effectiveConfig.skipCache) { this.setCachedResponse( requestKey, data,
effectiveConfig.cacheTTL! ); } }), // Cleanup on complete/error finalize(() =>
{ this.inFlightRequests.delete(requestKey); this.setLoadingState(requestKey,
false); }), // Share with multiple subscribers (deduplication) shareReplay(1) );



// Store in-flight request for deduplication this.inFlightRequests.set(requestKey,
request$);



return request$; }



/**



  • Get loading state for specific request
```

```
   *
  • @param requestKey - Request identifier
  • @returns Observable of loading state for this request
   */
getLoadingState$(requestKey: string): Observable<boolean> { return
this.loadingState$.pipe( map(stateMap => stateMap.get(requestKey) || false),
distinctUntilChanged() ); }

/**
  • Get global loading state (true if any request is loading)
   *
  • @returns Observable of global loading state
   */
getGlobalLoading$(): Observable<boolean> { return
this.loadingState$.pipe( map(stateMap => Array.from(stateMap.values()).some(loading =>
loading)), distinctUntilChanged() ); }

/**
  • Clear all cached responses or specific request cache
   *
  • @param requestKey - Optional specific request key to clear
   */
```

```
clearCache(requestKey?: string): void { if (requestKey)
{ this.cache.delete(requestKey); } else { this.cache.clear(); } }



/**



  • Invalidate specific cached response

   *

  • Alias for clearCache(requestKey) for semantic clarity.

   *

  • @param requestKey - Request identifier

   */
invalidateCache(requestKey: string): void { this.cache.delete(requestKey); }



/**



  • Invalidate all cached responses matching pattern

   *

  • Useful for invalidating related caches (e.g., all vehicle pages).

   *

  • @param pattern - String pattern to match (simple substring match)

   *

  • @example
```

```
•
```

typescript

• // Invalidate all vehicle-related caches

• coordinator.invalidateCachePattern('vehicles');

• // Clears: 'vehicles-page-1', 'vehicles-page-2', 'vehicles-stats'

```
   /
invalidateCachePattern(pattern: string): void { const keysToDelete: string[] = [];



this.cache.forEach((_, key) => { if (key.includes(pattern))
{ keysToDelete.push(key); } });



keysToDelete.forEach(key => this.cache.delete(key)); }



/**



  • Get current cache size

   *

  • @returns Number of cached entries

   */
getCacheSize(): number { return this.cache.size; }



/**



  • Get number of in-flight requests

   *

  • @returns Number of ongoing requests

   */
getInFlightCount(): number { return this.inFlightRequests.size; }
```

```
/**


  • Get cached response if exists and not expired

   *

  • @returns Cached data or null if not found/expired

   */
private getCachedResponse<T>(key: string): T | null { const entry =
this.cache.get(key); if (!entry) { return null; }



// Check if expired const age = Date.now() - entry.timestamp; if (age > entry.ttl)
{ // Expired - remove and return null this.cache.delete(key); return null; }



return entry.data; }



/**


  • Store response in cache

   */
private setCachedResponse<T>(key: string, data: T, ttl: number): void
{ this.cache.set(key, { data, timestamp: Date.now(), ttl }); }



/**


  • Update loading state for request

   */
```

```
private setLoadingState(requestKey: string, loading: boolean): void { const
currentStates = new Map(this.loadingStateSubject.value);



if (loading) { currentStates.set(requestKey, true); } else
{ currentStates.delete(requestKey); }



this.loadingStateSubject.next(currentStates); }



/**



  • Get effective configuration with defaults


   */
private getEffectiveConfig(config?: RequestConfig): Required<RequestConfig> { return
{ cacheTTL: config?.cacheTTL ?? 30000, retryAttempts: config?.retryAttempts ?? 3,
retryDelay: config?.retryDelay ?? 1000, skipCache: config?.skipCache ?? false }; } }
```

Step 303.2: Update the Barrel File

Update `src/app/framework/services/index.ts`:

```
// src/app/framework/services/index.ts
// VERSION 3 (Section 303) - Added RequestCoordinatorService



export * from './url-state.service'; export * from './api.service'; export * from './
request-coordinator.service';
```

Verification

1. Check File Exists

```
$ ls -la src/app/framework/services/request-coordinator.service.ts
```

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/framework/services/request-coordinator.service.ts
```

Expected: No output (no compilation errors).

3. Build the Application

```
$ ng build
```

Expected: Build succeeds with no errors.

4. Verify Deduplication (Optional)

Add temporary test code:

```
// In any component

constructor( private api: ApiService, private coordinator: RequestCoordinatorService )
{ // Fire same request 5 times simultaneously for (let i = 0; i < 5; i++)
{ this.coordinator.execute( 'test-dedup', () => this.api.get<any>('http://generic-
prime.minilab/api/specs/v1/automobiles') ).subscribe(response =>
{ console.log( Request ${i} received: , response); }); } }
```

In browser DevTools Network tab, you should see only **1** HTTP request, but 5 console logs.

5. Verify Caching (Optional)

```
// First request
this.coordinator.execute('test-cache', () => this.api.get<any>(url)) .subscribe(() =>
console.log('First request complete'));



// Second request after 1 second (should be cached) setTimeout(() =>
{ this.coordinator.execute('test-cache', () => this.api.get<any>(url)) .subscribe(()
=> console.log('Second request complete (from cache)')); }, 1000);



// Third request after 35 seconds (cache expired, new request) setTimeout(() =>
{ this.coordinator.execute('test-cache', () => this.api.get<any>(url)) .subscribe(()
=> console.log('Third request complete (fresh)')); }, 35000);
```

Network tab should show 2 requests total (first and third).

Common Problems

Symptom	Cause	Solution
Multiple HTTP requests despite deduplication	Different request keys	Ensure same key for same request
Cache never expires	TTL too high	Reduce cacheTTL value
Cache always misses	TTL too low	Increase cacheTTL value
Retries not working	retryAttempts set to 0	Use default or positive number
Loading state stuck on true	Request never completes	Check for hung HTTP requests
`shareReplay` memory leak	Observable never completes	finalize() handles cleanup

---

Key Takeaways

- **Three-layer processing prevents redundant requests** — Cache → Dedup → HTTP
- `shareReplay(1)` **enables deduplication** — Multiple subscribers share one HTTP call
- **Exponential backoff handles transient failures** — Each retry waits longer than the last

---

Acceptance Criteria

- [ ] `src/app/framework/services/request-coordinator.service.ts` exists
- [ ] Barrel file exports the service
- [ ] Service is `@Injectable({ providedIn: 'root' })`
- [ ] `execute()` implements three-layer processing
- [ ] Response cache stores with TTL and expires correctly
- [ ] In-flight deduplication shares observable for duplicate requests
- [ ] Retry logic uses exponential backoff
- [ ] `getLoadingState$()` returns per-request loading observable
- [ ] `getGlobalLoading$()` returns any-loading observable
- [ ] Cache invalidation methods work correctly
- [ ] TypeScript compilation succeeds
- [ ] JSDoc comments document all public methods

Next Step

Proceed to `304-domain-config-registry.md` to create the service that manages domain configurations.

304: Domain Config Registry

Status: Complete **Depends On:** 201-domain-config-interface **Blocks:** 305-domain-config-validator, 306-resource-management-service, 310-filter-options-service, 311-picker-config-registry

---

Learning Objectives

After completing this section, you will:

- Understand the Registry pattern and when to use it
- Know how to use Angular InjectionTokens for flexible configuration
- Recognize how runtime registration enables multi-domain applications
- Be able to implement a type-safe configuration registry

Objective

Create the `DomainConfigRegistry` service that provides centralized management of domain configurations. This registry allows domains to be registered, retrieved, and switched at runtime.

Why

The vvroom application is designed to be domain-agnostic. The framework doesn't know about automobiles — it knows about `DomainConfig`. This raises a question: *how does the framework access domain-specific configuration?*

Option 1: Import Directly

```
// In a component
import { AUTOMOBILE_DOMAIN_CONFIG } from './domain-config/automobile';


@Component({...}) export class DiscoverComponent { config =
AUTOMOBILE_DOMAIN_CONFIG; }
```

This works, but:

- Hard-codes the domain in the component
- Can't switch domains at runtime
- Can't add new domains without modifying framework code

Option 2: Injection Token

```
// In a module
providers: [ { provide: DOMAIN_CONFIG, useValue: AUTOMOBILE_DOMAIN_CONFIG } ]


// In a component constructor(@Inject(DOMAIN_CONFIG) private config:
DomainConfig<...>) {}
```

Better, but:

- Only one domain at a time
- No runtime switching
- No validation

Option 3: Registry Pattern (Our Approach)

```
// At app startup

registry.register(AUTOMOBILE_DOMAIN_CONFIG);
registry.register(REAL_ESTATE_DOMAIN_CONFIG);



// In a component const config = registry.getActive();



// Switch domain registry.setActive('real-estate');
```

The registry:

- Stores multiple domain configurations
- Provides runtime switching
- Validates configurations on registration
- Offers type-safe retrieval

Why Both Token AND Registry?

We use BOTH:

- `DOMAIN_CONFIG` **token** — For component injection (simpler DI)
- `DomainConfigRegistry` — For runtime management

The token is *populated from* the registry. Components inject `DOMAIN_CONFIG`, and the module provider gets it from the registry.

```
providers: [

{ provide: DOMAIN_CONFIG, useFactory: (registry: DomainConfigRegistry) =>
registry.getActive(), deps: [DomainConfigRegistry] } ]
```

What

Step 304.1: Create the Domain Config Registry Service

Create the file `src/app/framework/services/domain-config-registry.service.ts`:

```
// src/app/framework/services/domain-config-registry.service.ts
// VERSION 1 (Section 304) - Domain configuration registry


import { Injectable, InjectionToken, Injector, Provider } from '@angular/core'; import
{ DomainConfig } from '../models/domain-config.interface'; import
{ DomainConfigValidator } from './domain-config-validator.service';


/**


  • Injection token for domain configuration

 *

  • Used to provide domain-specific configuration to components.


  • The value is typically retrieved from DomainConfigRegistry.

 *

  • @example


  •
```

typescript

- // In domain module providers
- providers: [
- {
- provide: DOMAIN_CONFIG,
- useFactory: (registry: DomainConfigRegistry) => registry.getActive(),
- deps: [DomainConfigRegistry]
- }
-]

*

- // In component
- constructor(
- @Inject(DOMAIN_CONFIG) private domainConfig: DomainConfig<any, any, any>
-) {}

```
 /
export const DOMAIN_CONFIG = new InjectionToken<DomainConfig<any, any, any>>( 'Domain
Configuration' );



/**



  • Domain configuration registry service

 *

  • Centralized registry for managing multiple domain configurations.



  • Supports registering, retrieving, and switching between domains.

 *

  • Why a Registry?

 *

  • 1. Multi-domain support – Register multiple domains, switch at runtime



  • 2. Validation – Catch configuration errors early



  • 3. Type safety – Generic methods preserve type information



  • 4. Centralized access – Single point for all domain config access

 *

  • Typical Usage Flow:
```

```
 *
  • 1. App initializes → Register domain configs
  • 2. User navigates → Framework gets active config
  • 3. User switches domain → Registry changes active
  • 4. Framework components → Inject DOMAIN_CONFIG token
 *
  • @example
  •
```

typescript

- // Register domains at app initialization
- registry.register(AUTOMOBILE_DOMAIN_CONFIG);
- registry.register(REAL_ESTATE_DOMAIN_CONFIG);

*

- // Get active domain config
- const config = registry.getActive();

*

- // Switch active domain
- registry.setActive('real-estate');

*
    - // List all registered domains
    - const domains = registry.getAllDomainNames(); // ['automobile', 'real-estate']
*
    - // Get specific domain by name
    - const autoConfig = registry.get<AutoFilters, AutoData>('automobile');

```
 /
@Injectable({ providedIn: 'root' }) export class DomainConfigRegistry { /**


  • Storage for registered domain configurations

   *

  • Maps domain name (e.g., 'automobile') to DomainConfig instance.

   */
private configs = new Map<string, DomainConfig<any, any, any>>();


/**


  • Currently active domain name

   *

  • Set to first registered domain by default.


  • Can be changed via setActive().

   */
private activeDomainName?: string;


/**


  • Constructor - injects validator
```

```
   *
  • @param validator - Service for validating domain configurations
   */
constructor(private validator: DomainConfigValidator) {}

/**
  • Register a domain configuration
   *
  • Validates the configuration (unless disabled) and adds it to the registry.
  • First registered domain becomes the active domain.
   *
  • @template TFilters - Filter model type
  • @template TData - Data model type
  • @template TStatistics - Statistics model type
  • @param config - Domain configuration to register
  • @param validate - Whether to validate configuration (default: true)
```

```
  • @throws Error if configuration is invalid (when validate=true)
   *
  • @example
  •
```

typescript

- // Register with validation
- registry.register(AUTOMOBILE_DOMAIN_CONFIG);

*

- // Register without validation (e.g., in tests)
- registry.register(TEST_CONFIG, false);

```
   /
register<TFilters, TData, TStatistics>( config: DomainConfig<TFilters, TData,
TStatistics>, validate: boolean = true ): void { // Validate if requested if
(validate) { const sanitizedConfig = this.validator.validateAndSanitize(config);
config = sanitizedConfig as DomainConfig<TFilters, TData, TStatistics>; }



// Check for duplicate (warn but allow overwrite) if
(this.configs.has(config.domainName)) { console.warn(
Domain '${config.domainName}' already registered. Overwriting. ); }



// Register the config this.configs.set(config.domainName, config);



// Set as active if first domain if (!this.activeDomainName) { this.activeDomainName =
config.domainName; }



console.log( Domain '${config.domainName}' registered successfully ); }



/**



  • Register multiple domain configurations

   *

  • Convenience method for registering several domains at once.

   *

  • @param configs - Array of domain configurations



  • @param validate - Whether to validate configurations (default: true)
```

```
   */
registerMultiple( configs: DomainConfig<any, any, any>[], validate: boolean = true ):
void { configs.forEach((config) => this.register(config, validate)); }



/**



  • Get domain configuration by name

   *

  • @template TFilters - Filter model type



  • @template TData - Data model type



  • @template TStatistics - Statistics model type



  • @param domainName - Domain name to retrieve



  • @returns Domain configuration



  • @throws Error if domain not found

   *

  • @example



  •
```

typescript

- const config = registry.get<AutoFilters, VehicleResult>('automobile');
- // config is typed as DomainConfig<AutoFilters, VehicleResult, any>

```
   /
get<TFilters, TData, TStatistics>( domainName: string ): DomainConfig<TFilters, TData,
TStatistics> { const config = this.configs.get(domainName);



if (!config) { const available = this.getAllDomainNames().join(', '); throw new
Error( Domain '${domainName}' not found. Available domains: ${available} ); }



return config as DomainConfig<TFilters, TData, TStatistics>; }



/**



  • Get active domain configuration

   *

  • Returns the currently active domain's configuration.

   *

  • @template TFilters - Filter model type



  • @template TData - Data model type



  • @template TStatistics - Statistics model type



  • @returns Active domain configuration



  • @throws Error if no domain is active
```

```
   */
getActive<TFilters, TData, TStatistics>(): DomainConfig< TFilters, TData, TStatistics
> { if (!this.activeDomainName) { throw new Error('No active domain. Register a domain
first.'); }



return this.get<TFilters, TData, TStatistics>(this.activeDomainName); }



/**



  • Set active domain

   *

  • Changes which domain is returned by getActive().

   *

  • @param domainName - Domain name to activate



  • @throws Error if domain not found

   */
setActive(domainName: string): void { if (!this.configs.has(domainName)) { const
available = this.getAllDomainNames().join(', '); throw new Error( Cannot activate
domain '${domainName}'. Available domains: ${available} ); }



this.activeDomainName = domainName; console.log( Active domain set to '$
{domainName}' ); }



/**
```

```
   • Get active domain name
    *
   • @returns Active domain name or undefined if none
    */
getActiveDomainName(): string | undefined { return this.activeDomainName; }

/**
   • Check if domain is registered
    *
   • @param domainName - Domain name to check
   • @returns True if domain is registered
    */
has(domainName: string): boolean { return this.configs.has(domainName); }

/**
   • Get all registered domain names
    *
   • @returns Array of domain names
```

```
   */
getAllDomainNames(): string[] { return Array.from(this.configs.keys()); }



/**



  • Get all registered domain configurations

   *

  • @returns Array of domain configurations

   */
getAll(): DomainConfig<any, any, any>[] { return Array.from(this.configs.values()); }



/**



  • Unregister a domain

   *

  • Removes domain from registry. If it was active, activates the next available.

   *

  • @param domainName - Domain name to unregister



  • @returns True if domain was unregistered, false if not found

   */
unregister(domainName: string): boolean { const result =
this.configs.delete(domainName);
```

```
// Handle active domain removal if (this.activeDomainName === domainName)
{ this.activeDomainName = undefined;



// Set first remaining domain as active const remaining = this.getAllDomainNames(); if
(remaining.length > 0) { this.setActive(remaining[0]); } }



if (result) { console.log(`Domain '${domainName}' unregistered`); }



return result; }



/**



  • Clear all registered domains


   */
clear(): void { this.configs.clear(); this.activeDomainName = undefined;
console.log('All domains cleared'); }



/**



  • Get count of registered domains


   *


  • @returns Number of registered domains


   */
getCount(): number { return this.configs.size; }
```

```
/**

  • Validate a domain configuration without registering

   *

  • Useful for pre-validation before registration.

   *

  • @template TFilters - Filter model type

  • @template TData - Data model type

  • @template TStatistics - Statistics model type

  • @param config - Domain configuration to validate

  • @returns Validation result

   */
validate<TFilters, TData, TStatistics>( config: DomainConfig<TFilters, TData,
TStatistics> ) { return this.validator.validate(config); }

/**

  • Get human-readable validation summary for a domain
```

```
   *

  • @param domainName - Domain name



  • @returns Validation summary string


   */
getValidationSummary(domainName: string): string { const config =
this.get(domainName); const result = this.validator.validate(config); return
this.validator.getValidationSummary(result); } }
```

Step 304.2: Update the Barrel File

Update `src/app/framework/services/index.ts`:

```
// src/app/framework/services/index.ts

// VERSION 4 (Section 304) - Added DomainConfigRegistry



export * from './url-state.service'; export * from './api.service'; export * from './
request-coordinator.service'; export * from './domain-config-registry.service';
```

Step 304.3: Understand the Dependency

Note that `DomainConfigRegistry` depends on `DomainConfigValidator` (Section 305). The registry won't compile until the validator exists.

This creates a circular documentation dependency:

- Registry (304) uses Validator (305)
- We document Registry first to show the pattern

In practice, you'll create both files, then build.

Verification

1. Check File Exists

```
$ ls -la src/app/framework/services/domain-config-registry.service.ts
```

2. TypeScript Compilation Check

After creating Section 305 (Validator), run:

```
$ npx tsc --noEmit src/app/framework/services/domain-config-registry.service.ts
```

Expected: No output (no compilation errors).

3. Build the Application

After Section 305:

```
$ ng build
```

Expected: Build succeeds with no errors.

4. Verify Registration (Optional)

Add temporary test code:

```
// In app.component.ts ngOnInit

import { DomainConfigRegistry } from './framework/services';



constructor(private registry: DomainConfigRegistry) {}



ngOnInit(): void { // Mock config for testing (real config comes in Phase 6) const
mockConfig = { domainName: 'test-automobile', domainLabel: 'Test Automobiles',
apiBaseUrl: 'http://example.com/api', filterModel: class {}, dataModel: class {},
apiAdapter: { fetchData: () => {} }, urlMapper: { toUrlParams: () => ({}),
fromUrlParams: () => ({}) }, cacheKeyBuilder: { buildKey: () => '' }, tableConfig:
{ tableId: 'test', dataKey: 'id', columns: [{ field: 'id' }] }, pickers: [], filters:
[], queryControlFilters: [], charts: [], features: { highlights: true, popOuts: true,
rowExpansion: true } };



this.registry.register(mockConfig as any, false); // Skip validation for test
console.log('Registered domains:', this.registry.getAllDomainNames());
console.log('Active domain:', this.registry.getActiveDomainName()); }
```

Console should show:

```
Domain 'test-automobile' registered successfully

Registered domains: ['test-automobile'] Active domain: test-automobile
```

Common Problems

Symptom	Cause	Solution
`Cannot find module './domain-config-validator.service'`	Validator not created yet	Complete Section 305 first
`No active domain` error	No domains registered	Register at least one domain
`Domain 'X' not found`	Domain not registered	Check spelling, ensure registration happened
Validation errors on register	Config missing required fields	Check DomainConfig interface requirements
Console warnings about overwriting	Same domain registered twice	Remove duplicate registration

---

Key Takeaways

- **Registry pattern enables runtime flexibility** — Register, retrieve, switch domains dynamically
- **Injection tokens simplify component DI** — Components inject DOMAIN_CONFIG, not the registry
- **Validation catches errors early** — Invalid configs fail at registration, not at runtime

---

Acceptance Criteria

- [ ] `src/app/framework/services/domain-config-registry.service.ts` exists
- [ ] Barrel file exports the service and DOMAIN_CONFIG token
- [ ] Service is `@Injectable({ providedIn: 'root' })`
- [ ] `register()` validates and stores configurations
- [ ] `get()` retrieves configurations by name
- [ ] `getActive()` returns the active domain configuration
- [ ] `setActive()` changes the active domain
- [ ] First registered domain becomes active automatically
- [ ] `getAllDomainNames()` lists all registered domains
- [ ] `unregister()` removes domains and handles active domain gracefully
- [ ] DOMAIN_CONFIG InjectionToken is exported

- [ ] TypeScript compilation succeeds (after Section 305)
- [ ] JSDoc comments document all public methods

Next Step

Proceed to `305-domain-config-validator.md` to create the validation service that the registry depends on.

305: Domain Config Validator

Status: Complete **Depends On:** 201-domain-config-interface, 304-domain-config-registry **Blocks:** 306-resource-management-service

---

Learning Objectives

After completing this section, you will:

- Understand runtime validation patterns in TypeScript
- Know how to create comprehensive validation with error categorization
- Recognize the value of catching configuration errors early
- Be able to implement a reusable validation service

Objective

Create the `DomainConfigValidator` service that validates domain configurations at runtime, ensuring all required fields are present and correctly typed before the framework attempts to use them.

Why

TypeScript provides compile-time type checking, but it can't catch everything:

What TypeScript Catches

```
const config: DomainConfig<F, D, S> = {

domainName: 123, // Error: Type 'number' is not assignable to type 'string' // Missing
required properties - TypeScript error };
```

What TypeScript Misses

```
const config: DomainConfig<any, any, any> = {

domainName: '',        // Empty string - valid type, invalid value apiBaseUrl: 'not-
url', // Not a valid URL - valid type, invalid value tableConfig: { tableId: 'test',
columns: []          // Empty array - valid type, likely an error } };
```

TypeScript checks types. It doesn't check:

- Empty strings vs meaningful strings
- Valid URLs vs arbitrary strings
- Empty arrays vs populated arrays
- Interface compliance (does `apiAdapter` have `fetchData`?)

Why Not Just Trust TypeScript?

In a perfect world, every `DomainConfig` would be created with correct types and values. In practice:

- **Dynamic data** — Configs may come from JSON files or APIs
- **User error** — Developers make mistakes
- **Refactoring** — Interface changes can break existing configs
- **Third-party configs** — Can't trust external data

Runtime validation catches these issues **at registration time**, with clear error messages, instead of cryptic failures deep in framework code.

Validation vs. Sanitization

The validator does two things:

- **Validate** — Check for errors, return result object
- **Validate and Sanitize** — Check for errors, apply defaults, return clean config

```
// Validation only

const result = validator.validate(config); if (!result.valid)
{ console.error(result.errors); }



// Validation with sanitization (applies defaults) const cleanConfig =
validator.validateAndSanitize(config); // Throws if invalid, returns sanitized config
if valid
```

What

Step 305.1: Create the Domain Config Validator Service

Create the file `src/app/framework/services/domain-config-validator.service.ts`:

```
// src/app/framework/services/domain-config-validator.service.ts

// VERSION 1 (Section 305) - Runtime configuration validation



import { Injectable } from '@angular/core'; import { DomainConfig,
ConfigValidationError, ConfigValidationResult, ConfigErrorType, mergeDomainFeatures }
from '../models/domain-config.interface';



/**



  • Domain configuration validator service

 *

  • Validates domain configurations at runtime to ensure they meet all



  • requirements before being used by the framework. Catches errors that



  • TypeScript's compile-time checks miss.

 *

  • Why Runtime Validation?

 *

  • TypeScript checks types but not values:



  • - Empty strings pass type check but aren't valid domain names



  • - Invalid URLs pass type check but won't work for API calls
```

```
 • - Missing interface methods pass type check with 'any' types
*
 • Error Categories:
*
 • | Category | Description |
 • |----------|-------------|
 • | MISSING_REQUIRED | Required field is null/undefined |
 • | INVALID_TYPE | Field has wrong type |
 • | INVALID_VALUE | Field has valid type but invalid value |
 • | EMPTY_ARRAY | Required array is empty |
 • | DUPLICATE_ID | Duplicate ID in array (pickers, filters, charts) |
*
 • @example
 •
```

typescript

- const validator = inject(DomainConfigValidator);

*

- // Validate without modifying
- const result = validator.validate(config);
- if (!result.valid) {
- console.error('Errors:', result.errors);
- }

*

- // Validate and apply defaults
- try {
- const cleanConfig = validator.validateAndSanitize(config);
- } catch (error) {
- console.error('Invalid config:', error.message);
- }

*

- // Get human-readable summary
- console.log(validator.getValidationSummary(result));

```
 /
@Injectable({ providedIn: 'root' }) export class DomainConfigValidator { /**


  • Validate domain configuration

   *

  • Performs comprehensive validation and returns result object.


  • Does not modify the configuration.

   *

  • @template TFilters - Filter model type


  • @template TData - Data model type


  • @template TStatistics - Statistics model type


  • @param config - Domain configuration to validate


  • @returns Validation result with errors and warnings

   */
validate<TFilters, TData, TStatistics>( config: DomainConfig<TFilters, TData,
TStatistics> ): ConfigValidationResult { const errors: ConfigValidationError[] = [];
const warnings: ConfigValidationError[] = [];
```

```
// Validate required string fields this.validateRequiredString(config, 'domainName',
errors); this.validateRequiredString(config, 'domainLabel', errors);
this.validateRequiredString(config, 'apiBaseUrl', errors);



// Validate domain name format (lowercase, alphanumeric with hyphens)
this.validateDomainNameFormat(config.domainName, errors);



// Validate API base URL format this.validateApiBaseUrl(config.apiBaseUrl, errors);



// Validate type models this.validateRequiredField(config, 'filterModel', errors);
this.validateRequiredField(config, 'dataModel', errors);



// Validate adapters exist this.validateRequiredField(config, 'apiAdapter', errors);
this.validateRequiredField(config, 'urlMapper', errors);
this.validateRequiredField(config, 'cacheKeyBuilder', errors);



// Validate adapter interfaces (have required methods)
this.validateApiAdapter(config.apiAdapter, errors);
this.validateUrlMapper(config.urlMapper, errors);
this.validateCacheKeyBuilder(config.cacheKeyBuilder, errors);



// Validate table configuration this.validateRequiredField(config, 'tableConfig',
errors); this.validateTableConfig(config.tableConfig, errors);



// Validate arrays (may be empty, but should be arrays) this.validateArray(config,
'pickers', errors, warnings); this.validateArray(config, 'filters', errors, warnings);
this.validateArray(config, 'charts', errors, warnings);



// Validate array contents for duplicates if (config.pickers && config.pickers.length
> 0) { this.validatePickers(config.pickers, errors); }
```

```
if (config.filters && config.filters.length > 0)
{ this.validateFilters(config.filters, errors); }



if (config.charts && config.charts.length > 0) { this.validateCharts(config.charts,
errors); }



// Validate features this.validateRequiredField(config, 'features', errors);
this.validateFeatures(config.features, errors);



return { valid: errors.length === 0, errors, warnings }; }



/**



  • Validate and sanitize domain configuration

   *

  • Validates configuration and applies defaults for optional fields.



  • Throws if validation fails.

   *

  • @template TFilters - Filter model type



  • @template TData - Data model type



  • @template TStatistics - Statistics model type
```

```
  * @param config - Domain configuration to validate



  * @returns Sanitized configuration with defaults applied



  * @throws Error if configuration is invalid

   */
validateAndSanitize<TFilters, TData, TStatistics>( config: DomainConfig<TFilters,
TData, TStatistics> ): DomainConfig<TFilters, TData, TStatistics> { const result =
this.validate(config);



if (!result.valid) { const errorMessages = result.errors .map((e) => `${e.field}: ${
e.message}`) .join('\n'); throw new Error( `Domain configuration validation
failed:\n${errorMessages}` ); }



// Apply defaults for optional fields return { ...config, features:
mergeDomainFeatures(config.features), pickers: config.pickers || [], filters:
config.filters || [], charts: config.charts || [] }; }



/**



  * Get human-readable validation summary

   *

  * Formats validation result into readable multi-line string.

   *

  * @param result - Validation result from validate()
```

```
  • @returns Formatted summary string


   */
getValidationSummary(result: ConfigValidationResult): string { if (result.valid)
{ return 'Configuration is valid'; }



const lines: string[] = ['Configuration validation failed:'];



if (result.errors.length > 0) { lines.push(`\nErrors (${result.errors.length}):`);
result.errors.forEach((error, index) => { lines.push(`  ${index + 1}. [$
{error.type}] ${error.field}: ${error.message}`); }); }



if (result.warnings && result.warnings.length > 0) { lines.push(`\nWarnings ($
{result.warnings.length}):`); result.warnings.forEach((warning, index) =>
{ lines.push(`  ${index + 1}. [${warning.type}] ${warning.field}: $
{warning.message}`); }); }



return lines.join('\n'); }



// ========================================================================== //
Private Validation Methods //
============================================================================



/**



  • Validate required string field (non-empty)


   */
private validateRequiredString( config: any, field: string, errors:
ConfigValidationError[] ): void { if (!config[field]) { errors.push({ type:
ConfigErrorType.MISSING_REQUIRED, field, message: `Required field '${field}' is
```

```
missing, expected: 'string' }); } else if (typeof config[field] !== 'string')
{ errors.push({ type: ConfigErrorType.INVALID_TYPE, field, message:
Field '${field}' must be a string, expected: 'string', actual: typeof
config[field] }); } else if (config[field].trim().length === 0) { errors.push({ type:
ConfigErrorType.INVALID_VALUE, field, message: Field '${field}' cannot be empty,
expected: 'non-empty string', actual: config[field] }); } }



/**



  • Validate required field exists


   */

private validateRequiredField( config: any, field: string, errors:
ConfigValidationError[] ): void { if (!config[field]) { errors.push({ type:
ConfigErrorType.MISSING_REQUIRED, field, message: Required field '${field}' is
missing }); } }



/**



  • Validate domain name format


   */

private validateDomainNameFormat( domainName: string, errors:
ConfigValidationError[] ): void { if (!domainName) return;



// Domain name: lowercase, starts with letter, alphanumeric with hyphens const
validPattern = /^[a-z][a-z0-9-]*$/; if (!validPattern.test(domainName))
{ errors.push({ type: ConfigErrorType.INVALID_VALUE, field: 'domainName', message:
'Domain name must be lowercase, start with a letter, and contain only letters,
numbers, and hyphens', expected: 'lowercase alphanumeric with hyphens (e.g.,
"automobile")', actual: domainName }); } }



/**
```

```
  • Validate API base URL format

   */
private validateApiBaseUrl( apiBaseUrl: string, errors: ConfigValidationError[] ):
void { if (!apiBaseUrl) return;



try { new URL(apiBaseUrl); } catch (e) { errors.push({ type:
ConfigErrorType.INVALID_VALUE, field: 'apiBaseUrl', message: 'API base URL is not a
valid URL', expected: 'valid URL (e.g., "http://api.example.com")', actual:
apiBaseUrl }); } }



/**



  • Validate API adapter has required methods

   */
private validateApiAdapter( adapter: any, errors: ConfigValidationError[] ): void { if
(!adapter) return;



if (typeof adapter.fetchData !== 'function') { errors.push({ type:
ConfigErrorType.INVALID_TYPE, field: 'apiAdapter.fetchData', message: 'API adapter
must implement fetchData method', expected: 'function' }); } }



/**



  • Validate URL mapper has required methods

   */
private validateUrlMapper( mapper: any, errors: ConfigValidationError[] ): void { if
(!mapper) return;
```

```
if (typeof mapper.toUrlParams !== 'function') { errors.push({ type:
ConfigErrorType.INVALID_TYPE, field: 'urlMapper.toUrlParams', message: 'URL mapper
must implement toUrlParams method', expected: 'function' }); }



if (typeof mapper.fromUrlParams !== 'function') { errors.push({ type:
ConfigErrorType.INVALID_TYPE, field: 'urlMapper.fromUrlParams', message: 'URL mapper
must implement fromUrlParams method', expected: 'function' }); } }



/**



  • Validate cache key builder has required methods


   */
private validateCacheKeyBuilder( builder: any, errors: ConfigValidationError[] ): void
{ if (!builder) return;



if (typeof builder.buildKey !== 'function') { errors.push({ type:
ConfigErrorType.INVALID_TYPE, field: 'cacheKeyBuilder.buildKey', message: 'Cache key
builder must implement buildKey method', expected: 'function' }); } }



/**



  • Validate table configuration


   */
private validateTableConfig( tableConfig: any, errors: ConfigValidationError[] ): void
{ if (!tableConfig) return;



if (!tableConfig.tableId) { errors.push({ type: ConfigErrorType.MISSING_REQUIRED,
field: 'tableConfig.tableId', message: 'Table config must have tableId' }); }
```

```
if (!tableConfig.dataKey) { errors.push({ type: ConfigErrorType.MISSING_REQUIRED,
field: 'tableConfig.dataKey', message: 'Table config must have dataKey' }); }



if (!tableConfig.columns || !Array.isArray(tableConfig.columns)) { errors.push({ type:
ConfigErrorType.INVALID_TYPE, field: 'tableConfig.columns', message: 'Table config
must have columns array', expected: 'array' }); } else if (tableConfig.columns.length
=== 0) { errors.push({ type: ConfigErrorType.EMPTY_ARRAY, field:
'tableConfig.columns', message: 'Table config must have at least one column' }); } }



/**



  • Validate array field


   */
private validateArray( config: any, field: string, errors: ConfigValidationError[],
warnings: ConfigValidationError[] ): void { if (config[field] === undefined ||
config[field] === null) { // Missing array is a warning (will be defaulted to [])
warnings.push({ type: ConfigErrorType.MISSING_REQUIRED, field, message: `Field '$
{field}' is missing, will default to empty array`, expected: 'array' }); } else
if (!Array.isArray(config[field])) { // Wrong type is an error errors.push({ type:
ConfigErrorType.INVALID_TYPE, field, message: `Field '${field}' must be an array`,
expected: 'array', actual: typeof config[field] }); } }



/**



  • Validate pickers array (check for duplicate IDs)


   */
private validatePickers( pickers: any[], errors: ConfigValidationError[] ): void
{ const ids = new Set<string>();
```

```
pickers.forEach((picker, index) => { if (!picker.id) { errors.push({ type:
ConfigErrorType.MISSING_REQUIRED, field: pickers[${index}].id, message:
Picker at index ${index} must have id }); } else { if (ids.has(picker.id))
{ errors.push({ type: ConfigErrorType.DUPLICATE_ID, field: pickers[${index}].id,
message: Duplicate picker id: ${picker.id}, actual: picker.id }); }
ids.add(picker.id); } }); }



/**



  • Validate filters array (check for duplicate IDs and required fields)


   */
private validateFilters( filters: any[], errors: ConfigValidationError[] ): void
{ const ids = new Set<string>();



filters.forEach((filter, index) => { if (!filter.id) { errors.push({ type:
ConfigErrorType.MISSING_REQUIRED, field: filters[${index}].id, message:
Filter at index ${index} must have id }); } else { if (ids.has(filter.id))
{ errors.push({ type: ConfigErrorType.DUPLICATE_ID, field: filters[${index}].id,
message: Duplicate filter id: ${filter.id}, actual: filter.id }); }
ids.add(filter.id); }



if (!filter.type) { errors.push({ type: ConfigErrorType.MISSING_REQUIRED, field:
filters[${index}].type, message: Filter at index ${index} must have
type }); } }); }



/**



  • Validate charts array (check for duplicate IDs and required fields)


   */
private validateCharts( charts: any[], errors: ConfigValidationError[] ): void { const
ids = new Set<string>();
```

```
charts.forEach((chart, index) => { if (!chart.id) { errors.push({ type:
ConfigErrorType.MISSING_REQUIRED, field: charts[${index}].id, message: Chart at
index ${index} must have id }); } else { if (ids.has(chart.id))
{ errors.push({ type: ConfigErrorType.DUPLICATE_ID, field: charts[${index}].id,
message: Duplicate chart id: ${chart.id}, actual: chart.id }); }
ids.add(chart.id); }



if (!chart.type) { errors.push({ type: ConfigErrorType.MISSING_REQUIRED, field:
charts[${index}].type, message: Chart at index ${index} must have type }); }



if (!chart.dataSourceId) { errors.push({ type: ConfigErrorType.MISSING_REQUIRED,
field: charts[${index}].dataSourceId, message: Chart at index ${index} must
have dataSourceId }); } }); }



/**



  • Validate features object


   */
private validateFeatures( features: any, errors: ConfigValidationError[] ): void { if
(!features) return;



const requiredFeatures = ['highlights', 'popOuts', 'rowExpansion'];



requiredFeatures.forEach((feature) => { if (typeof features[feature] !== 'boolean')
{ errors.push({ type: ConfigErrorType.INVALID_TYPE, field: features.${feature},
message: Feature '${feature}' must be a boolean, expected: 'boolean', actual:
typeof features[feature] }); } }); } }
```

Step 305.2: Update the Barrel File

Update `src/app/framework/services/index.ts`:

```
// src/app/framework/services/index.ts
// VERSION 5 (Section 305) - Added DomainConfigValidator



export * from './url-state.service'; export * from './api.service'; export * from './
request-coordinator.service'; export * from './domain-config-registry.service'; export
* from './domain-config-validator.service';
```

Verification

1. Check File Exists

```
$ ls -la src/app/framework/services/domain-config-validator.service.ts
```

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/framework/services/domain-config-validator.service.ts
```

Expected: No output (no compilation errors).

3. Build the Application

```
$ ng build
```

Expected: Build succeeds with no errors.

4. Verify Validation (Optional)

Add temporary test code:

```
// In any component

constructor(private validator: DomainConfigValidator) { // Test invalid config const
badConfig = { domainName: '', // Empty - invalid domainLabel: 'Test', apiBaseUrl:
'not-a-url', // Invalid URL filterModel: null, // Missing dataModel: class {},
apiAdapter: {}, // Missing fetchData urlMapper: { toUrlParams: () => ({}) }, //
Missing fromUrlParams cacheKeyBuilder: { buildKey: () => '' }, tableConfig: { tableId:
'test', dataKey: 'id', columns: [] }, // Empty columns features: { highlights:
'yes' } // Wrong type } as any;



const result = this.validator.validate(badConfig); console.log('Valid:',
result.valid); console.log('Summary:\n',
this.validator.getValidationSummary(result)); }
```

Console should show multiple validation errors.

Common Problems

Symptom	Cause	Solution
`Cannot find module '../models/domain-config.interface'`	Interface file missing	Complete Section 201 first
`ConfigErrorType is not defined`	Enum not exported from interface	Check domain-config.interface.ts exports
No errors shown for bad config	Validation too lenient	Check validateRequiredField is called
validateAndSanitize doesn't throw	result.valid is true	Check all validation rules are running

---

Key Takeaways

- **Runtime validation catches what TypeScript misses** — Type checks pass for empty strings and invalid URLs

- **Categorized errors enable targeted fixes** — MISSING_REQUIRED vs INVALID_VALUE vs DUPLICATE_ID
- **Validation summary provides actionable feedback** — Human-readable error messages with field paths

---

Acceptance Criteria

- [ ] `src/app/framework/services/domain-config-validator.service.ts` exists
- [ ] Barrel file exports the service
- [ ] Service is `@Injectable({ providedIn: 'root' })`
- [ ] `validate()` returns ConfigValidationResult
- [ ] `validateAndSanitize()` throws on invalid, returns clean config on valid
- [ ] `getValidationSummary()` returns human-readable string
- [ ] Validates required string fields (non-empty)
- [ ] Validates domain name format (lowercase, alphanumeric)
- [ ] Validates API base URL is valid URL
- [ ] Validates adapter interfaces have required methods
- [ ] Validates table config has columns
- [ ] Validates arrays for duplicate IDs
- [ ] Validates features are booleans
- [ ] TypeScript compilation succeeds
- [ ] JSDoc comments document all public methods

Next Step

Proceed to `306-resource-management-service.md` to create the core state orchestration service.

306: Resource Management Service

Status: Complete **Depends On:** 301-url-state-service, 302-api-service, 303-request-coordinator, 304-domain-config-registry, 305-domain-config-validator **Blocks:** Phase 3B (307-311), Phase 8 (Framework Components), Phase 9 (Feature Components)

Learning Objectives

After completing this section, you will:

- Understand the role of a state orchestrator in URL-First architecture
- Know how BehaviorSubject enables reactive state management
- Recognize the pattern of converting URL changes to data fetches
- Be able to implement a generic, domain-agnostic resource management service

---

Objective

Create the `ResourceManagementService` that orchestrates application state with the URL as the single source of truth. This service coordinates filter changes, API calls, state updates, and cross-window synchronization.

Why

We've built the foundation:

- **UrlStateService** — Reads/writes URL parameters
- **ApiService** — Makes HTTP requests
- **RequestCoordinator** — Caches, deduplicates, retries

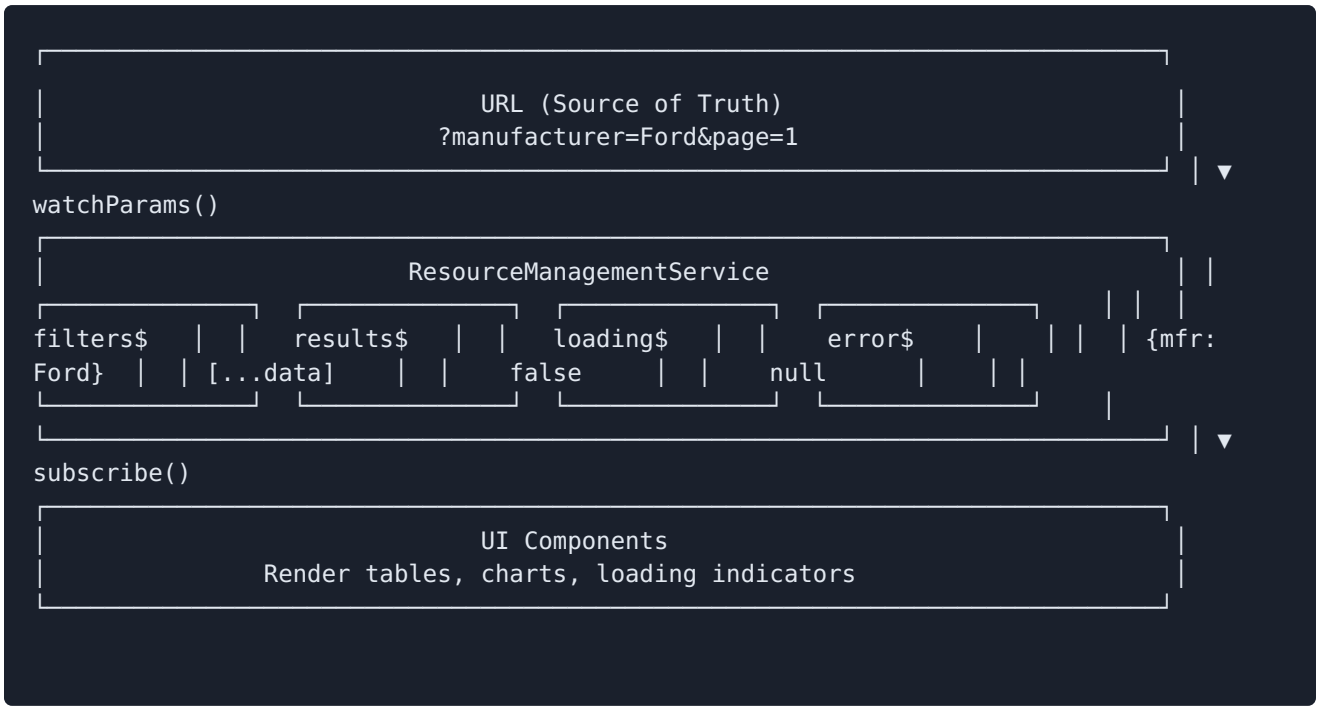
- **DomainConfigRegistry** — Provides domain configuration

Now we need a service that ties them together. When a user changes a filter:

```
User clicks filter → URL updates → Data fetches → UI updates
```

`ResourceManagementService` is the orchestrator that makes this flow work.

The State Flow



Why Not Put This in Components?

You might ask: *Why not just subscribe to URL changes in each component?*

```
// Bad: Every component manages its own state

@Component({...}) export class DiscoverComponent { filters$ =
this.urlState.watchParams().pipe( map(params => this.mapper.fromUrlParams(params)) );



results$ = this.filters$.pipe( switchMap(filters => this.api.get('/vehicles',
{ params: filters })) ); }
```

Problems:

- **Duplication** — Every component repeats the same logic
- **No state sharing** — Pop-outs can't share state with main window
- **No caching** — Every component fetches independently
- **Complex cleanup** — Each component manages subscriptions

`ResourceManagementService` centralizes this:

- **One place** for URL → filters → API → state logic
- **State sharing** via `syncStateFromExternal()`
- **Automatic caching** via RequestCoordinator
- **Proper cleanup** via `ngOnDestroy()`

Pop-Out Awareness

When the service runs in a pop-out window, it behaves differently:

- **Main window**: Watches URL, fetches data, updates state
- **Pop-out window**: Does NOT fetch (receives state from main window)

This prevents duplicate API calls and ensures consistency.

Component-Level Injection

Unlike singleton services, `ResourceManagementService` is **not** `providedIn: 'root'`. Each component that needs resource management provides its own instance:

```
@Component({

providers: [ResourceManagementService] }) export class DiscoverComponent
{ constructor(private resources: ResourceManagementService<F, D, S>) {} }
```

This enables different components to manage different resources independently.

What

Step 306.1: Create the Popout Token (Placeholder)

Before creating `ResourceManagementService`, we need the `IS_POPOUT_TOKEN`. This will be fully implemented in Section 315, but we need a placeholder now.

Create `src/app/framework/tokens/popout.token.ts`:

```
// src/app/framework/tokens/popout.token.ts
// VERSION 1 (Section 306) - Placeholder, fully implemented in Section 315


import { InjectionToken } from '@angular/core';


/**


  • Injection token to indicate if component is in a pop-out window

 *

  • When true, ResourceManagementService disables auto-fetching


  • and waits for state to be synced from the main window.

 */
export const IS_POPOUT_TOKEN = new InjectionToken<boolean>('IS_POPOUT_TOKEN');
```

Create the tokens barrel file `src/app/framework/tokens/index.ts`:

```
// src/app/framework/tokens/index.ts
// VERSION 1 (Section 306)


export * from './popout.token';
```

Step 306.2: Create Placeholder for PopOutContextService

We also need `PopOutContextService`. Create a placeholder:

```
// src/app/framework/services/popout-context.service.ts
// VERSION 1 (Section 306) - Placeholder, fully implemented in Section 307


import { Injectable } from '@angular/core';


/**


  • Pop-out context service (placeholder)

 *

  • Determines if the current window is a pop-out.


  • Full implementation in Section 307.

 */
@Injectable({ providedIn: 'root' }) export class PopOutContextService { /**


  • Check if current window is a pop-out

   *

  • @returns True if in pop-out window

   */
isInPopOut(): boolean { // Placeholder implementation // Real implementation in
Section 307 checks window.opener return false; } }
```

Step 306.3: Create the Resource Management Service

Create the file `src/app/framework/services/resource-management.service.ts`:

```
// src/app/framework/services/resource-management.service.ts

// VERSION 1 (Section 306) - Core state orchestrator



import { Inject, Injectable, NgZone, OnDestroy, Optional } from '@angular/core';
import { BehaviorSubject, Observable, of, Subject } from 'rxjs'; import { catchError,
distinctUntilChanged, finalize, map, takeUntil } from 'rxjs/operators'; import
{ DomainConfig } from '../models/domain-config.interface'; import
{ ResourceManagementConfig, ResourceState } from '../models/resource-
management.interface'; import { DOMAIN_CONFIG } from './domain-config-
registry.service'; import { PopOutContextService } from './popout-context.service';
import { UrlStateService } from './url-state.service'; import { IS_POPOUT_TOKEN } from
'../tokens/popout.token';



/**



  • Deep equality check for filter objects


 *


  • Replaces JSON.stringify comparison with proper structural equality.



  • Handles nested objects and arrays correctly.


 */

function deepEqual(a: unknown, b: unknown): boolean { if (a === b) return true; if (a
== null || b == null) return a === b; if (typeof a !== typeof b) return false; if
(typeof a !== 'object') return a === b;



const aObj = a as Record<string, unknown>; const bObj = b as Record<string, unknown>;
const aKeys = Object.keys(aObj); const bKeys = Object.keys(bObj);



if (aKeys.length !== bKeys.length) return false;
```

```
return aKeys.every(key => deepEqual(aObj[key], bObj[key])); }

/**

  • Generic resource management service - Core state orchestrator
 *
  • Purpose: Manages application state with URL as single source of truth.

  • Coordinates filter changes, API calls, state updates, and cross-window sync.
 *
  • Architecture: URL → Filters → API → Data → Components
 *
  • Key Features:
 *
  • 1. URL-first design — URL parameters are the single source of truth

  • 2. BehaviorSubject state — Current values with Observable streams

  • 3. Domain-agnostic — Works with any domain via DOMAIN_CONFIG injection

  • 4. Component-level injection — New instance per component (not singleton)
```

```
  • 5. Pop-out aware — Disables API calls in pop-out windows
 *
  • Usage Pattern:
 *
  •
```

typescript

- @Component({
- providers: [ResourceManagementService] // New instance for this component
- })
- export class DiscoverComponent {
- filters$ = this.resources.filters$;
- results$ = this.resources.results$;
- loading$ = this.resources.loading$;

*

- constructor(
- private resources: ResourceManagementService<AutoFilters, VehicleResult>
-) {}

*

- onFilterChange(filters: Partial<AutoFilters>): void {
- this.resources.updateFilters(filters);
- }
- }

```
  • @template TFilters - Domain-specific filter model type


  • @template TData - Domain-specific data model type


  • @template TStatistics - Domain-specific statistics model type

 */
@Injectable() // NOT providedIn: 'root' — component-level injection export class
ResourceManagementService<TFilters, TData, TStatistics = any> implements OnDestroy {


// ============================================================================ //
Internal State //
==============================================================================


/* Subject for cleanup on destroy */ private readonly destroy$ = new Subject<void>();


/* Configuration derived from DOMAIN_CONFIG */ private readonly config:
ResourceManagementConfig<TFilters, TData, TStatistics>;


// ============================================================================ //
BehaviorSubject-Based State //
==============================================================================


/* Main state holder — all state in one object */ private readonly stateSubject:
BehaviorSubject<ResourceState<TFilters, TData, TStatistics>>;


// ============================================================================ //
Observable Streams (Public API) //
==============================================================================
```

```
/* Full state observable */ public readonly state$: Observable<ResourceState<TFilters,
TData, TStatistics>>;


/* Current filters (from URL) */ public readonly filters$: Observable<TFilters>;


/* Current results (from API) */ public readonly results$: Observable<TData[]>;


/* Total result count (for pagination) */ public readonly totalResults$:
Observable<number>;


/* Loading state */ public readonly loading$: Observable<boolean>;


/* Current error (null if none) */ public readonly error$: Observable<Error | null>;


/* Statistics data (optional, depends on API) */ public readonly statistics$:
Observable<TStatistics | undefined>;


/* Highlight filters (for chart segmentation) */ public readonly highlights$:
Observable<any>;


// ============================================================================ //
Constructor //
==============================================================================


/**


  • Constructor — wires up state management
```

```
   *

  • @param urlState - Service for URL state management



  • @param domainConfig - Domain configuration (injected via DOMAIN_CONFIG)



  • @param popOutContext - Service to check if in pop-out window



  • @param ngZone - Angular zone for change detection



  • @param isPopOutToken - Optional token indicating pop-out mode

   */
constructor( private readonly urlState: UrlStateService, @Inject(DOMAIN_CONFIG)
private readonly domainConfig: DomainConfig<TFilters, TData, TStatistics>, private
readonly popOutContext: PopOutContextService, private readonly ngZone: NgZone,
@Optional() @Inject(IS_POPOUT_TOKEN) private readonly isPopOutToken: boolean ) { //
Determine if in pop-out (token takes precedence) const isPopOut =
this.isPopOutToken ?? false;



// Build configuration from domain config this.config = { filterMapper:
this.domainConfig.urlMapper, apiAdapter: this.domainConfig.apiAdapter,
cacheKeyBuilder: this.domainConfig.cacheKeyBuilder, defaultFilters:
(this.domainConfig.defaultFilters || {}) as TFilters, supportsHighlights:
this.domainConfig.features?.highlights ?? false, highlightPrefix: 'h_', // Disable
auto-fetch in pop-out windows autoFetch: isPopOut ? false : !
this.popOutContext.isInPopOut() };



// Initialize state with defaults this.stateSubject = new
BehaviorSubject<ResourceState<TFilters, TData, TStatistics>>({ filters:
this.config.defaultFilters, results: [], totalResults: 0, loading: false, error: null,
statistics: undefined });
```

```
// Create derived observables from state this.state$ =
this.stateSubject.asObservable();



this.filters$ = this.state$.pipe( map(s => s.filters), distinctUntilChanged((a, b) =>
deepEqual(a, b)) );



this.results$ = this.state$.pipe( map(s => s.results), distinctUntilChanged() );



this.totalResults$ = this.state$.pipe( map(s => s.totalResults),
distinctUntilChanged() );



this.loading$ = this.state$.pipe( map(s => s.loading), distinctUntilChanged() );



this.error$ = this.state$.pipe( map(s => s.error), distinctUntilChanged() );



this.statistics$ = this.state$.pipe( map(s => s.statistics), distinctUntilChanged() );



this.highlights$ = this.state$.pipe( map(s => s.highlights ?? {}),
distinctUntilChanged((a, b) => deepEqual(a, b)) );



// Initialize from current URL this.initializeFromUrl();



// Watch for URL changes this.watchUrlChanges(); }



// ============================================================================ //
Public API — State Mutation //
==============================================================================
```

```
/**

  • Update filters (triggers URL update → data fetch)

   *

  • Merges partial filters with current filters, updates URL.


  • URL change triggers automatic data fetch (in main window).

   *

  • @param partial - Partial filter object to merge

   *

  • @example


  •
```

typescript

• // Update manufacturer filter

• this.resources.updateFilters({ manufacturer: 'Ford' });

*

• // Update multiple filters

• this.resources.updateFilters({ manufacturer: 'Ford', page: 1 });

*

• // Clear a filter by setting to null/undefined

• this.resources.updateFilters({ manufacturer: null });

```
   /
updateFilters(partial: Partial<TFilters>): void { const currentFilters =
this.stateSubject.value.filters; const merged = { ...currentFilters, ...partial };



// Clean up empty values const newFilters: Record<string, any> = {}; for (const key of
Object.keys(merged)) { const value = (merged as Record<string, any>)[key]; if (value !
== undefined && value !== null && value !== '') { newFilters[key] = value; } }



// Convert to URL parameters const newUrlParams =
this.config.filterMapper.toUrlParams(newFilters as TFilters);



// Get current URL params to identify removals const currentUrlParams =
this.config.filterMapper.toUrlParams(currentFilters);



// Build final params with null for removed params const finalParams: Record<string,
any> = { ...newUrlParams }; for (const key of Object.keys(currentUrlParams)) { if (!
(key in newUrlParams)) { finalParams[key] = null; // Mark for removal } }



// Update URL (triggers watchUrlChanges → fetchData)
this.urlState.setParams(finalParams); }



/**



  • Clear all filters (reset to defaults)

   *

  • @example



  •
```

typescript

- // Reset to initial state
- this.resources.clearFilters();

```
   /
clearFilters(): void { const currentFilters = this.stateSubject.value.filters; const
currentUrlParams = this.config.filterMapper.toUrlParams(currentFilters); const
defaultUrlParams = this.config.filterMapper.toUrlParams(this.config.defaultFilters);



// Build params with null for all current params const finalParams: Record<string,
any> = { ...defaultUrlParams }; for (const key of Object.keys(currentUrlParams)) { if
(!(key in defaultUrlParams)) { finalParams[key] = null; } }



this.urlState.setParams(finalParams, true); // Replace URL }



/**



  • Refresh data with current filters

   *

  • Forces a new API call with current filter state.

   *

  • @example



  •
```

typescript

- // Force refresh after external data change
- this.resources.refresh();

```
   /
refresh(): void { this.fetchData(this.stateSubject.value.filters); }



// ============================================================================ //
Public API — State Access //
==============================================================================



/**



  • Get current state snapshot (synchronous)

   *

  • @returns Current resource state

   */
getCurrentState(): ResourceState<TFilters, TData, TStatistics> { return
this.stateSubject.value; }



/**



  • Get current filters snapshot (synchronous)

   *

  • @returns Current filters

   */
getCurrentFilters(): TFilters { return this.stateSubject.value.filters; }
```

```
// ============================================================================ //
Public API — Cross-Window Sync //
==============================================================================



/**



  • Sync state from external source

   *

  • Used by pop-out windows to receive state from main window.



  • Bypasses URL → fetch flow and directly updates state.

   *

  • @param externalState - State to sync from main window

   *

  • @example



  •
```

typescript

- // In pop-out window, receiving message from main window
- window.addEventListener('message', (event) => {
- if (event.data.type === 'STATE_SYNC') {
- this.resources.syncStateFromExternal(event.data.state);
- }
- });

```
   /
public syncStateFromExternal( externalState: ResourceState<TFilters, TData,
TStatistics> ): void { this.ngZone.run(() =>
{ this.stateSubject.next(externalState); }); }



// ================================================================================ //
Private Methods — Initialization //
================================================================================



/**



  • Initialize filters from current URL


   */
private initializeFromUrl(): void { const urlParams = this.urlState.getParams(); const
filters = this.config.filterMapper.fromUrlParams(urlParams); const highlights =
this.extractHighlights(urlParams);



this.updateState({ filters, highlights }); }



/**



  • Watch for URL changes and update state


   */
private watchUrlChanges(): void
{ this.urlState .watchParams() .pipe(takeUntil(this.destroy$)) .subscribe(urlParams =>
{ const filters = this.config.filterMapper.fromUrlParams(urlParams); const highlights
= this.extractHighlights(urlParams); this.updateState({ filters, highlights });
```

```
// Only fetch in main window (not in pop-out) if (this.config.autoFetch)
{ this.fetchData(filters); } }); }



// ============================================================================ //
Private Methods — Data Fetching //
==============================================================================



/**



  • Fetch data from API


   */
private fetchData(filters: TFilters): void { this.updateState({ loading: true, error:
null });



const highlights = this.stateSubject.value.highlights;



this.config.apiAdapter .fetchData(filters,
highlights) .pipe( takeUntil(this.destroy$), catchError(error =>
{ console.error('[ResourceManagementService] Fetch error:', error);
this.updateState({ loading: false, error: error instanceof Error ? error : new
Error(String(error)), results: [], totalResults: 0 }); return of(null); }),
finalize(() => { // Ensure loading is false even if observable completes without value
if (this.stateSubject.value.loading) { this.updateState({ loading:
false }); } }) ) .subscribe(response => { if (response) { this.updateState({ results:
response.results, totalResults: response.total, statistics: response.statistics,
loading: false, error: null }); } }); }



// ============================================================================ //
Private Methods — Highlights //
==============================================================================



/**
```

```
  • Extract highlight filters from URL parameters


   */
private extractHighlights(urlParams: Record<string, any>): any { // Prefer domain-
specific mapper strategy if (this.config.filterMapper.extractHighlights) { return
this.config.filterMapper.extractHighlights(urlParams); }



// Fallback: prefix-based extraction if (!this.config.supportsHighlights) { return
{}; }



const prefix = this.config.highlightPrefix || 'h_'; const highlights: Record<string,
any> = {};



Object.keys(urlParams).forEach(key => { if (key.startsWith(prefix)) { const
highlightKey = key.substring(prefix.length); let value = urlParams[key];



// Convert pipe-separated to comma-separated if (typeof value === 'string' &&
value.includes('|')) { value = value.replace(/\|/g, ','); }



highlights[highlightKey] = value; } });



return highlights; }



// ============================================================================ //
Private Methods — State Management //
==============================================================================



/**
```

```
  • Update state immutably

   */
private updateState(partial: Partial<ResourceState<TFilters, TData, TStatistics>>):
void { this.stateSubject.next({ ...this.stateSubject.value, ...partial }); }



// ============================================================================ //
Lifecycle //
==============================================================================



/**



  • Clean up subscriptions (public alias)

   */
destroy(): void { this.ngOnDestroy(); }



/**



  • Clean up subscriptions on component destroy

   */
ngOnDestroy(): void { this.destroy$.next(); this.destroy$.complete(); } }
```

Step 306.4: Update the Barrel File

Update `src/app/framework/services/index.ts`:

```
// src/app/framework/services/index.ts

// VERSION 6 (Section 306) - Added ResourceManagementService, PopOutContextService



export * from './url-state.service'; export * from './api.service'; export * from './
request-coordinator.service'; export * from './domain-config-registry.service'; export
* from './domain-config-validator.service'; export * from './popout-context.service';
export * from './resource-management.service';
```

Phase 3A Milestone: Demo Component

After completing Section 306, create a temporary demo component that displays URL state changes.

Create `src/app/features/demo/url-state-demo.component.ts`:

```
// src/app/features/demo/url-state-demo.component.ts

// TEMPORARY - Remove after verification



import { Component, OnInit, OnDestroy } from '@angular/core'; import { Subject } from
'rxjs'; import { takeUntil } from 'rxjs/operators'; import { UrlStateService } from
'../../framework/services';



@Component({ selector: 'app-url-state-demo', template:  <div style="padding: 20px;
font-family: monospace;"> <h2>URL State Demo</h2>


<div style="margin: 20px 0;"> <h3>Current URL Params:</h3>
<pre>{{ currentParams | json }}</pre> </div>


<div style="margin: 20px 0;"> <h3>Test Controls:</h3> <button
(click)="setManufacturer('Ford')">Set manufacturer=Ford</button> <button
(click)="setManufacturer('Toyota')">Set manufacturer=Toyota</button> <button
(click)="setPage(1)">Set page=1</button> <button (click)="setPage(2)">Set
page=2</button> <button (click)="clearAll()">Clear All</button> </div>


<div style="margin: 20px 0;"> <h3>Log:</h3> <div *ngFor="let log of logs"
style="font-size: 12px;">{{ log }}</div> </div> </div>


}) export class UrlStateDemoComponent implements OnInit, OnDestroy { currentParams:
any = {}; logs: string[] = []; private destroy$ = new Subject<void>();



constructor(private urlState: UrlStateService) {}



ngOnInit(): void
{ this.urlState.watchParams() .pipe(takeUntil(this.destroy$)) .subscribe(params =>
```

```
{ this.currentParams = params; this.logs.unshift( ${new Date().toISOString()}: ${JSON.stringify(params)} ); if (this.logs.length > 10) this.logs.pop(); }); }



setManufacturer(value: string): void { this.urlState.setParam('manufacturer', value); }



setPage(value: number): void { this.urlState.setParam('page', value); }



clearAll(): void { this.urlState.clearParams(); }



ngOnDestroy(): void { this.destroy$.next(); this.destroy$.complete(); } }
```

Add route for demo (temporary):

```
// In app-routing.module.ts

{ path: 'demo', component: UrlStateDemoComponent }
```

Navigate to `/demo` and test the URL-First pattern in action.

Verification

1. Check Files Exist

```
$ ls -la src/app/framework/services/resource-management.service.ts

$ ls -la src/app/framework/services/popout-context.service.ts $ ls -la src/app/framework/tokens/popout.token.ts
```

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/framework/services/resource-management.service.ts
```

3. Build the Application

```
$ ng build
```

Expected: Build succeeds with no errors.

4. Run Demo Component

```
$ ng serve
```

Navigate to `http://localhost:4200/demo` and:

- Click "Set manufacturer=Ford"
- Observe URL changes to `?manufacturer=Ford`
- Observe Current URL Params updates
- Use browser back button
- Observe params revert

This is the Phase 3A Aha Moment in action:

"The URL is the single source of truth. Components react to URL changes, they don't control them."

Common Problems

Symptom	Cause	Solution
`Cannot find module '../tokens/popout.token'`	Token file not created	Create the token file in Step 306.1
`No provider for DOMAIN_CONFIG`	Missing provider	Add provider in component or module
State not updating	Missing subscription	Ensure component subscribes to observables
Infinite loop on filter change	Filter update triggers URL, URL triggers fetch	Check distinctUntilChanged is working
Pop-out doesn't receive state	syncStateFromExternal not called	Implement message passing (Phase 3B)

Key Takeaways

- **URL is the single source of truth** — updateFilters() modifies URL, not internal state
- **State flows one direction** — URL → filters → API → results → components
- **Component-level injection enables isolation** — Each component gets its own instance
- **Pop-out awareness prevents duplicate fetches** — Main window fetches, pop-outs sync

---

Acceptance Criteria

- [ ] `src/app/framework/services/resource-management.service.ts` exists
- [ ] `src/app/framework/services/popout-context.service.ts` exists (placeholder)
- [ ] `src/app/framework/tokens/popout.token.ts` exists
- [ ] Service is `@Injectable()` (not providedIn: 'root')
- [ ] Observable streams: filters$, results$, loading$, error$, statistics$
- [ ] `updateFilters()` updates URL, not internal state directly
- [ ] `clearFilters()` resets to default filters
- [ ] `refresh()` forces new data fetch
- [ ] `syncStateFromExternal()` enables pop-out state sync
- [ ] autoFetch disabled in pop-out windows

- [ ] Proper cleanup via ngOnDestroy()
- [ ] TypeScript compilation succeeds
- [ ] Demo component works as described

Phase 3A Complete

Congratulations! You have completed Phase 3A: Core Services.

What you built:

- UrlStateService — URL as source of truth
- ApiService — Thin HTTP wrapper
- RequestCoordinatorService — Cache, dedup, retry
- DomainConfigRegistry — Multi-domain support
- DomainConfigValidator — Runtime validation
- ResourceManagementService — State orchestration

The Aha Moment: "The URL is the single source of truth. Components react to URL changes, they don't control them."

Next Step

Proceed to `307-popout-context-service.md` to begin Phase 3B: Popout & Specialized Services.

307: Popout Context Service

Status: Complete **Depends On:** 208-popout-interface, 306-resource-management-service **Blocks:** 308-popout-manager-service

---

Learning Objectives

After completing this section, you will:

- Understand the BroadcastChannel API for cross-window communication
- Know how to determine if code is running in a pop-out window
- Recognize the messaging patterns for parent-child window coordination
- Be able to implement bidirectional communication between windows

---

Objective

Create the `PopOutContextService` that determines whether the current window is a pop-out and provides messaging infrastructure for communication between the main window and pop-out windows.

Why

Pop-out windows are a core feature of vvroom. Users can detach panels (statistics, charts) into separate browser windows for multi-monitor workflows. This creates a challenge: *how do these windows communicate?*

The Pop-Out Challenge

When you call `window.open()`, the new window is essentially a separate instance:

- Different JavaScript context

- Different Angular application instance
- No shared services or state

We need a way for:

- **Main window** → Detect when pop-out is ready
- **Main window** → Send state updates to pop-out
- **Pop-out** → Receive state updates
- **Pop-out** → Notify main window it's closing

Communication Options

Option	Pros	Cons
`window.opener`	Direct reference	Security restrictions, fragile
`postMessage()`	Standard API	Requires origin checks, complex
`BroadcastChannel`	Simple, bidirectional	Browser support (good since 2017)
`SharedWorker`	Powerful	Complex setup

We use **BroadcastChannel** because:

- Simple API: `channel.postMessage()` and `channel.onmessage`
- Works across same-origin windows
- No need to track window references
- Automatic cleanup when windows close

The Context Service Pattern

`PopOutContextService` serves two roles:

- **In pop-out window**: Determines "I am a pop-out" and sets up messaging
- **In main window**: Initializes as parent for pop-out management

```
// Pop-out window

if (contextService.isInPopOut()) { contextService.initializeAsPopOut('statistics-
panel'); }



// Main window contextService.initializeAsParent();
```

Route-Based Detection

Pop-outs are identified by their URL pattern:

```
/popout/:gridId/:panelId/:panelType
```

Example: `/popout/main-grid/statistics-1/statistics-panel-2`

The service parses this URL to extract:

- `isPopOut`: true (URL starts with `/popout`)
- `gridId`: "main-grid"
- `panelId`: "statistics-1"
- `panelType`: "statistics-panel-2"

What

Step 307.1: Replace the Placeholder Service

Replace the placeholder created in Section 306 with the full implementation.

Update `src/app/framework/services/popout-context.service.ts`:

```
// src/app/framework/services/popout-context.service.ts
// VERSION 2 (Section 307) - Full implementation with BroadcastChannel

import { Injectable, OnDestroy, NgZone } from '@angular/core'; import { Router } from
'@angular/router'; import { Observable, ReplaySubject } from 'rxjs'; import
{ PopOutMessage, PopOutMessageType, PopOutContext, parsePopOutRoute } from '../models/
popout.interface';

/**
  • Pop-out context service
 *
  • Determines whether the current window is a pop-out and provides
  • messaging infrastructure for parent-child window communication.
 *
  • Two Modes:
 *
  • 1. Pop-out mode — Running in a pop-out window
  •    - isInPopOut() returns true
  •    - Sets up BroadcastChannel to receive state from main window
```

```
 •    - Sends PANEL_READY message when initialized
*
 • 2. Parent mode — Running in main window
 •    - `isInPopOut()` returns false
 •    - Manages channels for each pop-out panel
 •    - Broadcasts state updates to all pop-outs
*
 • BroadcastChannel Pattern:
*
 • Each panel gets its own channel named `panel-{panelId}`.
 • This allows targeted messaging to specific panels.
*
 • @example
 •
```

typescript

- // In pop-out window component
- if (this.context.isInPopOut()) {
- const ctx = this.context.getContext();
- this.context.initializeAsPopOut(ctx.panelId);

*

- this.context.getMessages$().subscribe(message => {
- if (message.type === PopOutMessageType.STATE_UPDATE) {
- this.handleStateUpdate(message.payload);
- }
- });
- }

*

- // In main window
- this.context.initializeAsParent();

```
 /
@Injectable({ providedIn: 'root' }) export class PopOutContextService implements
OnDestroy { /**



  • BroadcastChannel for this window

   *

  • In pop-out: channel for receiving messages from main window



  • In parent: may be null (uses per-panel channels via PopOutManager)

   */
private channel: BroadcastChannel | null = null;



/**



  • ReplaySubject for incoming messages

   *

  • ReplaySubject with buffer of 10 ensures late subscribers



  • can receive recent messages they might have missed.

   */
private messagesSubject = new ReplaySubject<PopOutMessage>(10);



/**
```

```
   • Parsed context from URL
    *
   • Contains isPopOut, gridId, panelId, panelType
    */
private context: PopOutContext | null = null;

/**
   • Initialization flag to prevent double-init
    */
private initialized = false;

/**
   • Constructor - parses URL to determine context
    *
   • @param router - Angular Router for URL access
   • @param ngZone - NgZone for ensuring change detection
    */
constructor( private router: Router, private ngZone: NgZone ) { // Parse context from
current URL this.context = parsePopOutRoute(this.router.url); }
```

```
/**


  • Check if current window is a pop-out

   *

  • Determined by URL pattern: /popout/:gridId/:panelId/:panelType

   *

  • @returns True if in pop-out window

   */
isInPopOut(): boolean { // Re-parse if context not set (defensive) if (!this.context)
{ this.context = parsePopOutRoute(this.router.url); } return this.context?.isPopOut ||
false; }



/**


  • Get parsed pop-out context

   *

  • @returns Context object with gridId, panelId, panelType, or null

   */
getContext(): PopOutContext | null { if (!this.context) { this.context =
parsePopOutRoute(this.router.url); } return this.context; }



/**
```

```
  • Initialize as pop-out window

   *

  • Sets up BroadcastChannel for receiving messages from main window.


  • Sends PANEL_READY message to notify main window.

   *

  • @param panelId - Panel identifier for channel naming

   */
initializeAsPopOut(panelId: string): void { if (this.initialized) { return; }


this.initialized = true; this.setupChannel(panelId);


// Notify main window that pop-out is ready this.sendMessage({ type:
PopOutMessageType.PANEL_READY, timestamp: Date.now() }); }


/**


  • Initialize as parent window

   *

  • Called by main window to set initialized flag.


  • Actual channel management is handled by PopOutManagerService.
```

```
   */
initializeAsParent(): void { if (this.initialized) { return; }



this.initialized = true; // Parent doesn't set up a channel here - PopOutManager
handles per-panel channels }



/**



  • Set up BroadcastChannel for this panel

   *

  • @param panelId - Panel identifier for channel naming

   */
private setupChannel(panelId: string): void { const channelName = panel-${panelId};



// Close existing channel if any if (this.channel) { this.channel.close(); }



// Create new channel this.channel = new BroadcastChannel(channelName);



// Handle incoming messages this.channel.onmessage = (event: MessageEvent) => { const
message = event.data as PopOutMessage; // Use NgZone to ensure Angular change
detection runs this.ngZone.run(() => { this.messagesSubject.next(message); }); };



// Handle message errors (rare, but log them) this.channel.onmessageerror = () =>
{ console.warn('[PopOutContextService] Message deserialization error'); }; }
```

```
/**


  • Send message to channel

   *

  • Used by pop-out to send messages to main window.

   *

  • @template T - Payload type



  • @param message - Message to send

   */
sendMessage<T = any>(message: PopOutMessage<T>): void { if (!this.channel)
{ console.warn('[PopOutContextService] No channel available for sending'); return; }



// Add timestamp if not present if (!message.timestamp) { message.timestamp =
Date.now(); }



try { this.channel.postMessage(message); } catch (error)
{ console.error('[PopOutContextService] Failed to send message:', error); } }



/**


  • Get observable of incoming messages

   *

  • @returns Observable of PopOutMessage
```

```
   */
getMessages$(): Observable<PopOutMessage> { return
this.messagesSubject.asObservable(); }



/**



  • Create a BroadcastChannel for a specific panel


   *


  • Used by PopOutManagerService to create channels for pop-out windows.


   *


  • @param panelId - Panel identifier



  • @returns New BroadcastChannel instance


   */
createChannelForPanel(panelId: string): BroadcastChannel { const channelName =
`panel-${panelId}`; return new BroadcastChannel(channelName); }



/**



  • Close the channel


   *


  • Call when pop-out is closing or service is destroyed.


   */
close(): void { if (this.channel) { this.channel.close(); this.channel = null; }
```

```
this.initialized = false; }


/**


  • Cleanup on service destroy


   */
ngOnDestroy(): void { this.close(); this.messagesSubject.complete(); } }
```

Step 307.2: Verify Popout Interface Exists

Ensure `src/app/framework/models/popout.interface.ts` exists (from Section 208).

If the interface doesn't include `parsePopOutRoute`, add it:

```
// Add to src/app/framework/models/popout.interface.ts
// VERSION 2 (Section 307) - Added parsePopOutRoute function


/**


  • Parse pop-out route to extract context

 *

  • @param url - Current router URL


  • @returns PopOutContext with parsed values, or default context if not pop-out

 *

  • @example


  •
```

typescript

• parsePopOutRoute('/popout/main-grid/stats-1/statistics-panel-2')

• // Returns: { isPopOut: true, gridId: 'main-grid', panelId: 'stats-1', panelType: 'statistics-panel-2' }

*

• parsePopOutRoute('/discover')

• // Returns: { isPopOut: false, gridId: '', panelId: '', panelType: '' }

```
 /
export function parsePopOutRoute(url: string): PopOutContext { // Pattern: /
popout/:gridId/:panelId/:panelType const match = url.match(/^\/popout\/([^\/]+)\/
([^\/]+)\/([^\/\?]+)/);



if (match) { return { isPopOut: true, gridId: match[1], panelId: match[2], panelType:
match[3] }; }



return { isPopOut: false, gridId: '', panelId: '', panelType: '' }; }
```

Step 307.3: Update the Barrel File

The barrel file should already export this service from Section 306. Verify:

```
// src/app/framework/services/index.ts

export * from './popout-context.service';
```

Verification

1. Check File Updated

```
$ wc -l src/app/framework/services/popout-context.service.ts
```

Should show ~200+ lines (not the short placeholder).

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/framework/services/popout-context.service.ts
```

Expected: No output (no compilation errors).

3. Build the Application

```
$ ng build
```

Expected: Build succeeds with no errors.

4. Verify Context Detection (Optional)

Add temporary test code:

```
// In app.component.ts

import { PopOutContextService } from './framework/services';



constructor(private popOutContext: PopOutContextService) { console.log('Is pop-out?',
this.popOutContext.isInPopOut()); console.log('Context:',
this.popOutContext.getContext()); }
```

Navigate to `/` — Should log `Is pop-out? false` Navigate to `/popout/grid1/panel1/stats` — Should log `Is pop-out? true`

Common Problems

Symptom	Cause	Solution
`Cannot find module '../models/popout.interface'`	Interface file missing	Complete Section 208 first
`parsePopOutRoute is not a function`	Function not exported	Add to popout.interface.ts and export
isInPopOut always false	URL doesn't match pattern	Check URL pattern is `/popout/...`
Messages not received	Channel name mismatch	Verify both sides use `panel-{panelId}`
Change detection not running	Missing NgZone.run()	Wrap message handling in ngZone.run()

Key Takeaways

- **BroadcastChannel enables simple cross-window messaging** — No window references needed
- **URL pattern determines pop-out status** — Route-based detection is reliable
- **ReplaySubject with buffer catches late subscribers** — Messages aren't lost

---

Acceptance Criteria

- [ ] `src/app/framework/services/popout-context.service.ts` fully implemented
- [ ] `isInPopOut()` correctly detects pop-out windows by URL pattern
- [ ] `getContext()` returns parsed gridId, panelId, panelType
- [ ] `initializeAsPopOut()` sets up BroadcastChannel
- [ ] `initializeAsParent()` sets initialized flag
- [ ] `sendMessage()` sends via BroadcastChannel
- [ ] `getMessages$()` returns observable of incoming messages
- [ ] `createChannelForPanel()` creates channels for parent use
- [ ] `parsePopOutRoute()` function added to popout.interface.ts
- [ ] NgZone.run() wraps message handling
- [ ] TypeScript compilation succeeds

- [ ] JSDoc comments document all public methods

Next Step

Proceed to `308-popout-manager-service.md` to create the service that manages pop-out windows from the main window.

308: Popout Manager Service

Status: Complete **Depends On:** 307-popout-context-service, 208-popout-interface **Blocks:** Phase 9 (Feature Components)

Learning Objectives

After completing this section, you will:

- Understand the window.open() API and its configuration options
- Know how to track pop-out window lifecycle (open, close)
- Recognize the state broadcast pattern for multi-window synchronization
- Be able to implement a comprehensive pop-out window manager

---

Objective

Create the `PopOutManagerService` that opens, tracks, and communicates with pop-out windows from the main window. This service is the counterpart to `PopOutContextService` — one manages from the main window, the other runs inside pop-outs.

Why

`PopOutContextService` handles the pop-out side. Now we need the parent side:

Service	Runs In	Responsibility
PopOutContextService	Pop-out window	Receive state, send events
PopOutManagerService	Main window	Open pop-outs, broadcast state, track lifecycle

The Manager's Responsibilities

- **Open pop-out windows** — Call `window.open()` with correct URL and features
- **Track open pop-outs** — Know which panels are currently popped out
- **Broadcast state** — Send state updates to all pop-out windows
- **Handle closing** — Detect when pop-outs close and clean up
- **Prevent duplicates** — Don't open same panel twice

Why Component-Level Injection?

Unlike `PopOutContextService` (singleton), `PopOutManagerService` is component-level:

```
@Component({
providers: [PopOutManagerService] }) export class DiscoverComponent { ... }
```

Reasons:

- Manager is tied to a specific grid/layout
- Different pages might have different pop-out needs
- Cleanup is easier when tied to component lifecycle

The State Broadcast Pattern

When filters change in the main window:

```
Main Window                              Pop-out Window

|                                        | ├── Filters change                    |
|                                        | ├── ResourceManagementService         | |
updates state$                         | |                                     | ├──
Component subscribes to state$         | |                                     | ├──
Component calls                        | |   popOutManager.broadcastState()    | |
|                          | |           └──── BroadcastChannel ──►|
|                                        | |                                     ├──
PopOutContextService |                                    |   receives message
|                                        | |                                     ├──
ResourceManagementService |
|   .syncStateFromExternal() |                                    |
|                                        └── UI updates
```

Pop-outs never make API calls. They receive state from the main window.

Window Lifecycle Detection

JavaScript can't directly listen for window close events on child windows. We use polling:

```
const checkInterval = setInterval(() => {

if (popoutWindow.closed) { clearInterval(checkInterval);
this.handlePopOutClosed(panelId); } }, 500);
```

What

Step 308.1: Create the Popout Manager Service

Create the file `src/app/framework/services/popout-manager.service.ts`:

```
// src/app/framework/services/popout-manager.service.ts
// VERSION 1 (Section 308) - Pop-out window manager

import { Injectable, NgZone, OnDestroy } from '@angular/core'; import { Subject } from
'rxjs'; import { buildWindowFeatures, PopOutMessage, PopOutMessageType,
PopOutWindowFeatures, PopOutWindowRef } from '../models/popout.interface'; import
{ PopOutContextService } from './popout-context.service';

/**
  • Pop-out manager service
 *
  • Manages pop-out windows from the main window. Opens windows,
  • broadcasts state, and tracks lifecycle.
 *
  • Important: This is a component-level service, not a singleton.
  • Each component that needs pop-out management provides its own instance.
 *
  • Lifecycle:
 *
  • 1. Component creates → Manager injected
```

```
  • 2. Component calls initialize() with grid ID


  • 3. User clicks pop-out button → openPopOut() called


  • 4. Filter changes → broadcastState() called


  • 5. Pop-out closes → Manager detects and cleans up


  • 6. Component destroys → All pop-outs closed
*
  • @example


  •
```

typescript

- @Component({
- providers: [PopOutManagerService]
- })
- export class DiscoverComponent implements OnInit, OnDestroy {
- constructor(
- private popOutManager: PopOutManagerService,
- private resources: ResourceManagementService
-) {}

*

- ngOnInit(): void {
- this.popOutManager.initialize('discover-grid');

*

- // Broadcast state to pop-outs when it changes
- this.resources.state$.subscribe(state => {
- this.popOutManager.broadcastState(state);
- });

*

- // Handle pop-out closure
- this.popOutManager.closed$.subscribe(panelId => {
- console.log(`Pop-out ${panelId} closed`);
- });
- }

*

- onPopOutClick(panelId: string, panelType: string): void {
- this.popOutManager.openPopOut(panelId, panelType);
- }
- }

```
 /
@Injectable() // Component-level, not providedIn: 'root' export class
PopOutManagerService implements OnDestroy { /**



  • Grid ID for URL construction


   */
private gridId = '';



/**



  • Set of currently popped-out panel IDs


   */
private poppedOutPanels = new Set<string>();



/**



  • Map of panel ID to window reference/channel


   */
private popoutWindows = new Map<string, PopOutWindowRef>();



/**



  • Subject for messages from pop-outs


   */
private messagesSubject = new Subject<{ panelId: string; message: PopOutMessage }>();
```

```
/**

  • Subject for pop-out close events

   */
private closedSubject = new Subject<string>();

/**

  • Subject for blocked pop-up events (browser blocked window.open)

   */
private blockedSubject = new Subject<string>();

/**

  • Handler for beforeunload to close all pop-outs

   */
private beforeUnloadHandler = () => this.closeAllPopOuts();

/**

  • Initialization flag

   */
private initialized = false;
```

```
// Public observables readonly messages$ = this.messagesSubject.asObservable();
readonly closed$ = this.closedSubject.asObservable(); readonly blocked$ =
this.blockedSubject.asObservable();



/**



  • Constructor

   *

  • @param popOutContext - Context service for channel creation



  • @param ngZone - NgZone for change detection

   */
constructor( private popOutContext: PopOutContextService, private ngZone: NgZone ) {}



/**



  • Initialize the manager

   *

  • Must be called before any pop-out operations.

   *

  • @param gridId - Grid identifier for URL construction

   */
```

```
initialize(gridId: string): void { if (this.initialized) { return; }



this.gridId = gridId; this.initialized = true;



// Initialize context as parent this.popOutContext.initializeAsParent();



// Close all pop-outs when main window closes window.addEventListener('beforeunload',
this.beforeUnloadHandler);



// Subscribe to messages from pop-outs this.popOutContext.getMessages$
().subscribe(message => { this.messagesSubject.next({ panelId: '', message }); }); }



/**



  • Check if a panel is currently popped out

   *

  • @param panelId - Panel identifier



  • @returns True if panel is popped out

   */
isPoppedOut(panelId: string): boolean { return this.poppedOutPanels.has(panelId); }



/**



  • Get all currently popped-out panel IDs
```

```
   *
  • @returns Array of panel IDs
   */
getPoppedOutPanels(): string[] { return Array.from(this.poppedOutPanels); }

/**
  • Open a panel in a pop-out window
   *
  • @param panelId - Panel identifier
  • @param panelType - Panel type (e.g., 'statistics-panel-2')
  • @param features - Optional window features
  • @returns True if opened successfully, false if blocked or already open
   */
openPopOut( panelId: string, panelType: string, features?:
Partial<PopOutWindowFeatures> ): boolean { // Don't open if already popped out if
(this.poppedOutPanels.has(panelId))
{ console.log( [PopOutManager] Panel ${panelId} already popped out ); return
false; }

// Construct URL: /popout/:gridId/:panelId/:panelType const url = /popout/$
{this.gridId}/${panelId}/${panelType} ;
```

```
// Build window features string const windowFeatures = buildWindowFeatures({ width:
1200, height: 800, left: 100, top: 100, resizable: true, scrollbars:
true, ...features });



// Open the window const popoutWindow = window.open(url, `panel-${panelId}`,
windowFeatures);



// Handle blocked popup if (!popoutWindow) { console.warn(`[PopOutManager] Pop-up
blocked for panel ${panelId}`); this.blockedSubject.next(panelId); return false; }



// Track the pop-out this.poppedOutPanels.add(panelId);



// Create channel for this panel const channel =
this.popOutContext.createChannelForPanel(panelId);



// Listen for messages from this panel channel.onmessage = event =>
{ this.ngZone.run(() => { this.messagesSubject.next({ panelId, message:
event.data }); }); };



// Poll to detect when window closes const checkInterval = window.setInterval(() =>
{ if (popoutWindow.closed) { this.ngZone.run(() => { this.handlePopOutClosed(panelId,
channel, checkInterval); }); } }, 500);



// Store reference this.popoutWindows.set(panelId, { window: popoutWindow, channel,
checkInterval, panelId, panelType });



console.log(`[PopOutManager] Opened pop-out for panel ${panelId}`); return
true; }
```

```
/**


  • Broadcast state to all pop-out windows

   *

  • Call this whenever state changes in the main window.

   *

  • @param state - Application state from ResourceManagementService


  • @param filterOptionsCache - Optional cached filter options

   */
broadcastState(state: any, filterOptionsCache?: any): void { if
(this.popoutWindows.size === 0) { return; // No pop-outs to broadcast to }


const message: PopOutMessage = { type: PopOutMessageType.STATE_UPDATE, payload:
{ state, filterOptionsCache: filterOptionsCache || null }, timestamp: Date.now() };


// Send to all pop-out channels this.popoutWindows.forEach(({ channel }) => { try
{ channel.postMessage(message); } catch { // Silently ignore posting errors (channel
may be closed) } }); }


/**


  • Close a specific pop-out window

   *

  • @param panelId - Panel identifier
```

```
   */
closePopOut(panelId: string): void { const ref = this.popoutWindows.get(panelId); if
(ref) { // Send close message ref.channel.postMessage({ type:
PopOutMessageType.CLOSE_POPOUT, timestamp: Date.now() }); } }



/**



  • Close all pop-out windows

   *

  • Called when main window is closing.

   */
closeAllPopOuts(): void { this.popoutWindows.forEach(({ channel }) =>
{ channel.postMessage({ type: PopOutMessageType.CLOSE_POPOUT, timestamp:
Date.now() }); }); }



/**



  • Handle pop-out window closed

   *

  • @param panelId - Panel that closed



  • @param channel - BroadcastChannel to close



  • @param checkInterval - Interval to clear

   */
```

```
private handlePopOutClosed( panelId: string, channel: BroadcastChannel, checkInterval:
number ): void { // Clear polling interval clearInterval(checkInterval);



// Close channel channel.close();



// Remove from tracking this.popoutWindows.delete(panelId);
this.poppedOutPanels.delete(panelId);



// Emit closed event this.closedSubject.next(panelId);



console.log( [PopOutManager] Pop-out ${panelId} closed ); }



/**



  • Cleanup on service destroy


   */
ngOnDestroy(): void { // Remove beforeunload listener
window.removeEventListener('beforeunload', this.beforeUnloadHandler);



// Close all pop-outs and cleanup this.popoutWindows.forEach(({ window: win, channel,
checkInterval }) => { clearInterval(checkInterval); channel.close(); if (win && !
win.closed) { win.close(); } });



// Complete subjects this.messagesSubject.complete(); this.closedSubject.complete();
this.blockedSubject.complete(); } }
```

Step 308.2: Update Popout Interface

Ensure `src/app/framework/models/popout.interface.ts` includes necessary types:

```
// Add to src/app/framework/models/popout.interface.ts if not present
// VERSION 2 (Section 308) - Added window reference types

/**
  • Pop-out window features configuration
 */
export interface PopOutWindowFeatures { width: number; height: number; left: number; top: number; resizable: boolean; scrollbars: boolean; menubar?: boolean; toolbar?: boolean; location?: boolean; status?: boolean; }

/**
  • Reference to an open pop-out window
 */
export interface PopOutWindowRef { /* Window object */ window: Window; /* BroadcastChannel for communication */ channel: BroadcastChannel; /* Interval ID for close detection */ checkInterval: number; /* Panel identifier */ panelId: string; /* Panel type */ panelType: string; }

/**
  • Build window.open() features string from options
 *
  • @param features - Window features configuration
```

```
  • @returns Features string for window.open()


 */
export function buildWindowFeatures(features: PopOutWindowFeatures): string { const
parts: string[] = [ width=${features.width}, height=${features.height},
left=${features.left}, top=${features.top},
resizable=${features.resizable ? 'yes' : 'no'}, scrollbars=$
{features.scrollbars ? 'yes' : 'no'} ];



if (features.menubar !== undefined) { parts.push( menubar=${features.menubar ?
'yes' : 'no'} ); } if (features.toolbar !== undefined) { parts.push( toolbar=$
{features.toolbar ? 'yes' : 'no'} ); } if (features.location !== undefined)
{ parts.push( location=${features.location ? 'yes' : 'no'} ); } if
(features.status !== undefined) { parts.push( status=${features.status ? 'yes' :
'no'} ); }



return parts.join(','); }
```

Step 308.3: Update the Barrel File

Update `src/app/framework/services/index.ts`:

```
// src/app/framework/services/index.ts

// VERSION 7 (Section 308) - Added PopOutManagerService



export * from './url-state.service'; export * from './api.service'; export * from './
request-coordinator.service'; export * from './domain-config-registry.service'; export
* from './domain-config-validator.service'; export * from './popout-context.service';
export * from './popout-manager.service'; export * from './resource-
management.service';
```

Phase 3B Milestone: Pop-Out Demo

After completing Section 308, demonstrate pop-out communication:

- Add temporary route for pop-out testing
- Create a simple pop-out component
- Verify communication works

```
// Temporary demo component in main window

@Component({ selector: 'app-popout-demo',
template: `
  <h2>Pop-Out Demo</h2>
  <button (click)="openPopOut()">Open Pop-Out</button>
  <p>Popped out: {{ isPoppedOut }}</p>
  <div *ngFor="let log of logs">{{ log }}</div>
`,
providers: [PopOutManagerService] }) export class PopOutDemoComponent implements OnInit { isPoppedOut = false; logs: string[] = [];



constructor(private popOutManager: PopOutManagerService) {}



ngOnInit(): void { this.popOutManager.initialize('demo-grid');



this.popOutManager.closed$.subscribe(panelId => { this.logs.push(`Panel ${panelId} closed`); this.isPoppedOut = false; }); }



openPopOut(): void { this.isPoppedOut = this.popOutManager.openPopOut('demo-panel', 'demo'); if (this.isPoppedOut) { this.logs.push('Pop-out opened'); } } }
```

Verification

1. Check File Exists

```
$ ls -la src/app/framework/services/popout-manager.service.ts
```

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/framework/services/popout-manager.service.ts
```

3. Build the Application

```
$ ng build
```

Common Problems

Symptom	Cause	Solution
Pop-up blocked	Browser blocking window.open()	User must allow pop-ups for site
`blocked$` emits	window.open() returned null	Check browser pop-up settings
Messages not received	Channel name mismatch	Verify `panel-{panelId}` format
Pop-out not detected as closed	Polling interval issue	Check checkInterval is set
State not updating in pop-out	broadcastState not called	Subscribe to state$ and broadcast

---

Key Takeaways

- **Pop-outs are core functionality** — Users need multi-monitor workflows
- **BroadcastChannel simplifies messaging** — No window reference management
- **Polling detects window close** — No direct event available

Acceptance Criteria

- [ ] `src/app/framework/services/popout-manager.service.ts` exists
- [ ] Service is component-level `@Injectable()` (not singleton)
- [ ] `initialize()` sets up grid ID and event listeners
- [ ] `openPopOut()` opens window with correct URL pattern
- [ ] `isPoppedOut()` tracks open panels
- [ ] `broadcastState()` sends state to all pop-outs
- [ ] `closePopOut()` and `closeAllPopOuts()` work
- [ ] Pop-out close detection via polling works
- [ ] `closed$` emits when pop-outs close
- [ ] `blocked$` emits when pop-ups are blocked
- [ ] beforeunload handler closes all pop-outs
- [ ] TypeScript compilation succeeds
- [ ] JSDoc comments document all public methods

The Aha Moment

Pop-out windows share state with the parent through a coordination service.

The main window fetches data. Pop-outs receive state through BroadcastChannel. This ensures:

- No duplicate API calls
- Consistent state across windows
- Clean separation of concerns

Next Step

Proceed to `309-user-preferences-service.md` to create the service for persisting user preferences.

309: User Preferences Service

Status: Complete **Depends On:** None (standalone) **Blocks:** Phase 8 (Framework Components - for panel ordering)

Learning Objectives

After completing this section, you will:

- Understand localStorage patterns for persistent state
- Know how to handle storage failures gracefully
- Recognize domain-aware key namespacing patterns
- Be able to implement a preferences service with reactive state

---

Objective

Create the `UserPreferencesService` that persists user preferences (panel order, collapsed state) using localStorage with domain-aware namespacing and graceful degradation.

Why

Users customize their experience:

- Drag panels to reorder them
- Collapse panels they don't use often
- Their preferences should persist across sessions

Storage Options

Option	Persistence	Sharing	Complexity
Component state	None	None	Low
Service state	Session only	Same tab	Low
localStorage	Permanent	Same origin	Medium
Backend API	Permanent	All devices	High

We use **localStorage** because:

- Works offline
- No backend required
- Suitable for UI preferences
- Falls back gracefully in private browsing

Domain-Aware Namespacing

Vvroom supports multiple domains (automobile, agriculture, etc.). Each domain may have different panels and preferences. Keys are namespaced:

```
prefs:automobile:panelOrder → ['stats-1', 'chart-1', 'query-control']

prefs:automobile:collapsedPanels → ['chart-1'] prefs:agriculture:panelOrder → ['crop-stats', 'yield-chart']
```

Graceful Degradation

localStorage can fail:

- Private browsing mode
- Storage quota exceeded
- SecurityError in iframes

The service:

- Checks storage availability on init
- Uses in-memory fallback if unavailable
- Logs warnings in dev mode

- Never crashes the app

What

Step 309.1: Create the User Preferences Service

Create the file `src/app/framework/services/user-preferences.service.ts`:

```
// src/app/framework/services/user-preferences.service.ts
// VERSION 1 (Section 309) - User preferences with localStorage


import { Injectable, isDevMode } from '@angular/core'; import { BehaviorSubject,
Observable } from 'rxjs';



/**


  • User preferences service

 *

  • Persists user preferences using localStorage with domain-aware namespacing.


  • Handles storage failures gracefully with in-memory fallback.

 *

  • Features:

 *

  • 1. Domain namespacing — Each domain has its own preferences


  • 2. Reactive state — BehaviorSubjects for real-time updates


  • 3. Graceful degradation — Works even if localStorage unavailable


  • 4. Panel merge logic — New panels inserted at correct positions
```

```
 *
  • Storage Keys:
 *
  •
```

• prefs:{domain}:panelOrder → ['panel-1', 'panel-2', ...]

• prefs:{domain}:collapsedPanels → ['panel-1']

```
  • @example
  •
```

typescript

• // In component

• constructor(private prefs: UserPreferencesService) {

• // Subscribe to panel order changes

• this.prefs.getPanelOrder().subscribe(order => {

• this.panelOrder = order;

• });

*

• // Save new order after drag-drop

• this.prefs.savePanelOrder(['panel-2', 'panel-1', 'panel-3']);

• }

```
 /
@Injectable({ providedIn: 'root' }) export class UserPreferencesService { /**



  • Default panel order when no preferences saved


   */
private readonly DEFAULT_PANEL_ORDER = [ 'query-control', 'statistics-panel-2',
'results-table' ];



/**



  • Default collapsed panels (none)


   */
private readonly DEFAULT_COLLAPSED_PANELS: string[] = [];



/**



  • Current domain extracted from URL


   */
private currentDomain = this.extractCurrentDomain();



/**



  • BehaviorSubject for panel order


   */
private panelOrderSubject = new BehaviorSubject<string[]>( this.loadPanelOrder() );
```

```
/**


  • BehaviorSubject for collapsed panels

   */
private collapsedPanelsSubject = new
BehaviorSubject<string[]>( this.loadCollapsedPanels() );


/**


  • Storage availability flag

   */
private storageAvailable = this.checkStorageAvailable();


/**


  • Get panel order as observable

   *

  • Emits current order immediately, then on every change.

   *

  • @returns Observable of panel order array

   */
getPanelOrder(): Observable<string[]> { return
this.panelOrderSubject.asObservable(); }
```

```
/**


  • Get collapsed panels as observable

   *

  • @returns Observable of collapsed panel IDs

   */
getCollapsedPanels(): Observable<string[]> { return
this.collapsedPanelsSubject.asObservable(); }



/**


  • Get current panel order synchronously

   *

  • @returns Current panel order array

   */
getCurrentPanelOrder(): string[] { return this.panelOrderSubject.value; }



/**


  • Get current collapsed panels synchronously

   *

  • @returns Current collapsed panel IDs
```

```
   */
getCurrentCollapsedPanels(): string[] { return this.collapsedPanelsSubject.value; }



/**



  • Save panel order

   *

  • Updates BehaviorSubject and persists to localStorage.

   *

  • @param order - New panel order array

   */
savePanelOrder(order: string[]): void { this.panelOrderSubject.next(order);
this.saveToStorage('panelOrder', order); }



/**



  • Save collapsed panels

   *

  • @param panels - Array of collapsed panel IDs

   */
saveCollapsedPanels(panels: string[]): void
{ this.collapsedPanelsSubject.next(panels); this.saveToStorage('collapsedPanels',
panels); }
```

```
/**


  • Toggle panel collapsed state

   *

  • @param panelId - Panel to toggle



  • @returns New collapsed state

   */
togglePanelCollapsed(panelId: string): boolean { const current =
this.collapsedPanelsSubject.value; let newCollapsed: string[]; let isCollapsed:
boolean;



if (current.includes(panelId)) { newCollapsed = current.filter(id => id !== panelId);
isCollapsed = false; } else { newCollapsed = [...current, panelId]; isCollapsed =
true; }



this.saveCollapsedPanels(newCollapsed); return isCollapsed; }



/**



  • Check if panel is collapsed

   *

  • @param panelId - Panel to check



  • @returns True if collapsed
```

```
   */
isPanelCollapsed(panelId: string): boolean { return
this.collapsedPanelsSubject.value.includes(panelId); }



/**



  • Reset preferences for current domain


   *


  • @param domain - Optional domain (defaults to current)


   */
reset(domain?: string): void { const targetDomain = domain || this.currentDomain;



if (this.storageAvailable) { try { localStorage.removeItem(this.getKey('panelOrder',
targetDomain)); localStorage.removeItem(this.getKey('collapsedPanels',
targetDomain)); } catch (e) { // Ignore errors on reset } }



this.panelOrderSubject.next(this.DEFAULT_PANEL_ORDER);
this.collapsedPanelsSubject.next(this.DEFAULT_COLLAPSED_PANELS); }



/**



  • Switch to a different domain


   *


  • Loads preferences for the new domain.
```

```
   *

  • @param domain - Domain to switch to

   */
switchDomain(domain: string): void { this.currentDomain = domain;
this.panelOrderSubject.next(this.loadPanelOrder());
this.collapsedPanelsSubject.next(this.loadCollapsedPanels()); }



/**



  • Get current domain

   *

  • @returns Current domain name

   */
getCurrentDomain(): string { return this.currentDomain; }



// ========================================================================== //
Private Methods //
============================================================================



/**



  • Load panel order from localStorage

   */
private loadPanelOrder(): string[] { if (!this.storageAvailable) { return
this.DEFAULT_PANEL_ORDER; }
```

```
try { const key = this.getKey('panelOrder'); const stored = localStorage.getItem(key);



if (stored) { const parsed = JSON.parse(stored); if (Array.isArray(parsed)) { // Merge
with defaults to include new panels return this.mergePanelOrder(parsed,
this.DEFAULT_PANEL_ORDER); } } } catch (e) { this.logError('loadPanelOrder', e); }



return this.DEFAULT_PANEL_ORDER; }



/**



  • Load collapsed panels from localStorage


   */
private loadCollapsedPanels(): string[] { if (!this.storageAvailable) { return
this.DEFAULT_COLLAPSED_PANELS; }



try { const key = this.getKey('collapsedPanels'); const stored =
localStorage.getItem(key);



if (stored) { const parsed = JSON.parse(stored); if (Array.isArray(parsed)) { return
parsed; } } } catch (e) { this.logError('loadCollapsedPanels', e); }



return this.DEFAULT_COLLAPSED_PANELS; }



/**



  • Merge stored order with defaults
```

```
   *

  • - Removes panels no longer in defaults



  • - Inserts new panels at their default relative position

   *

  • @param stored - User's stored order



  • @param defaults - Default order (source of truth for valid panels)



  • @returns Merged order

   */
private mergePanelOrder(stored: string[], defaults: string[]): string[] { const
defaultsSet = new Set(defaults);



// Filter out panels that no longer exist const result = stored.filter(id =>
defaultsSet.has(id)); const resultSet = new Set(result);



// Insert new panels at their relative position for (let i = 0; i < defaults.length;
i++) { const panelId = defaults[i]; if (!resultSet.has(panelId)) { // Find insertion
point based on previous panel in defaults let insertIndex = result.length;



for (let j = i - 1; j >= 0; j--) { const prevPanel = defaults[j]; const prevIndex =
result.indexOf(prevPanel); if (prevIndex !== -1) { insertIndex = prevIndex + 1;
break; } }



result.splice(insertIndex, 0, panelId); resultSet.add(panelId); } }
```

```
return result; }



/**



  • Save to localStorage


   */
private saveToStorage(preference: string, value: any): void { if (!
this.storageAvailable) { return; }



try { const key = this.getKey(preference); localStorage.setItem(key,
JSON.stringify(value)); } catch (e) { this.handleStorageError(e); } }



/**



  • Get storage key with domain prefix


   */
private getKey(preference: string, domain?: string): string { const targetDomain =
domain || this.currentDomain; return prefs:${targetDomain}:${preference}; }



/**



  • Check if localStorage is available


   */
private checkStorageAvailable(): boolean { try { const test = '__localStorage_test__';
localStorage.setItem(test, 'test'); localStorage.removeItem(test); return true; }
catch (e) { return false; } }
```

```
/**


  • Handle storage errors


   */
private handleStorageError(error: any): void { this.storageAvailable = false;
this.logError('storage', error); }



/**


  • Log error in dev mode only


   */
private logError(context: string, error: any): void { if (isDevMode())
{ console.debug( [UserPreferencesService] ${context} error: , error); } }



/**


  • Extract current domain from URL


   */
private extractCurrentDomain(): string { const path = window.location.pathname; const
match = path.match(/^\/([a-z-]+)/);



if (match && match[1] && match[1] !== 'popout') { return match[1]; }



return 'automobile'; // Default domain } }
```

Step 309.2: Update the Barrel File

Update `src/app/framework/services/index.ts`:

```
// src/app/framework/services/index.ts
// VERSION 8 (Section 309) - Added UserPreferencesService



export * from './url-state.service'; export * from './api.service'; export * from './
request-coordinator.service'; export * from './domain-config-registry.service'; export
* from './domain-config-validator.service'; export * from './popout-context.service';
export * from './popout-manager.service'; export * from './user-preferences.service';
export * from './resource-management.service';
```

Verification

1. Check File Exists

```
$ ls -la src/app/framework/services/user-preferences.service.ts
```

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/framework/services/user-preferences.service.ts
```

3. Build the Application

```
$ ng build
```

4. Verify Storage (Optional)

```
// In any component

constructor(private prefs: UserPreferencesService) { // Save order
this.prefs.savePanelOrder(['panel-b', 'panel-a', 'panel-c']);



// Check localStorage in DevTools // Key: prefs:automobile:panelOrder // Value:
["panel-b","panel-a","panel-c"]



// Subscribe to changes this.prefs.getPanelOrder().subscribe(order =>
{ console.log('Current order:', order); }); }
```

Common Problems

Symptom	Cause	Solution
Preferences not persisting	localStorage unavailable	Check private browsing mode
Wrong domain detected	URL pattern mismatch	Check extractCurrentDomain() logic
New panels missing	Not in DEFAULT_PANEL_ORDER	Update defaults when adding panels
Quota exceeded	Too much data	Clear old preferences or reduce data
JSON parse error	Corrupted data	Service handles with try/catch

---

Key Takeaways

- **Graceful degradation is essential** — App works even if storage fails
- **Domain namespacing enables multi-domain support** — Preferences don't conflict
- **Panel merge logic handles updates** — New panels appear at correct positions

---

Acceptance Criteria

- [ ] `src/app/framework/services/user-preferences.service.ts` exists
- [ ] Service is `@Injectable({ providedIn: 'root' })`
- [ ] `getPanelOrder()` returns observable
- [ ] `getCollapsedPanels()` returns observable
- [ ] `savePanelOrder()` persists to localStorage
- [ ] `saveCollapsedPanels()` persists to localStorage
- [ ] `togglePanelCollapsed()` toggles and saves
- [ ] `reset()` clears preferences
- [ ] `switchDomain()` loads different domain's preferences
- [ ] Domain-aware key namespacing works
- [ ] Graceful degradation when localStorage unavailable
- [ ] Panel merge logic handles new/removed panels
- [ ] TypeScript compilation succeeds
- [ ] JSDoc comments document all public methods

Next Step

Proceed to `310-filter-options-service.md` to create the service for caching filter dropdown options.

310: Filter Options Service

Status: Complete **Depends On:** 302-api-service, 304-domain-config-registry **Blocks:** Phase 8 (Framework Components - for filter dropdowns)

Learning Objectives

After completing this section, you will:

- Understand why filter options caching matters for pop-outs
- Know how to implement a cache-first data loading pattern
- Recognize the transformer pattern for API response adaptation
- Be able to implement a service that supports cross-window option sharing

---

Objective

Create the `FilterOptionsService` that caches filter dropdown options and supports cross-window sharing for pop-out windows.

Why

Filter dropdowns (manufacturer, body class, year) need options from the API. Without caching:

- **Duplicate requests** — Each dropdown fetches options independently
- **Pop-out overhead** — Pop-out windows make their own API calls
- **Slow UX** — Users wait for options to load

The Pop-Out Challenge

Pop-out windows should NOT make API calls. They receive state from the main window. But what about filter options?

```
Main Window                    Pop-out Window

│                               ├── Load manufacturer options    │ │   (API
call)                  │ │                                  ├── Load body class
options      │ │   (API call)                 │ │                              │
├── Open pop-out                │ │   │                          │ │   └──
Broadcast state ──────────►│ │      (includes filter cache)  │
│                               │ │                                  ├──
FilterOptionsService │                                 │   syncFromExternal()
│                               │ │                                  └── Options
available │                                  (no API calls!)
```

Cache Structure

```
{
'http://api/manufacturers': { field: 'manufacturer', endpoint: 'http://api/
manufacturers', options: [{ value: 'Ford', label: 'Ford' }, ...], rawResponse: { data:
[...] }, cachedAt: 1707500000000 }, 'http://api/body-classes': { field: 'bodyClass',
endpoint: 'http://api/body-classes', options: [{ value: 'Sedan', label:
'Sedan' }, ...], rawResponse: { data: [...] }, cachedAt: 1707500001000 } }
```

Transformer Pattern

Different APIs return options in different formats. The service accepts a transformer function:

```
// API returns: { data: [{ id: 1, name: 'Ford' }, ...] }
// We need: [{ value: 1, label: 'Ford' }, ...]



filterOptions.getOptions( 'http://api/manufacturers', 'manufacturer', (response) =>
response.data.map(item => ({ value: item.id, label: item.name })) );
```

What

Step 310.1: Create the Filter Options Service

Create the file `src/app/framework/services/filter-options.service.ts`:

```
// src/app/framework/services/filter-options.service.ts

// VERSION 1 (Section 310) - Filter options caching



import { Injectable } from '@angular/core'; import { BehaviorSubject, Observable, of }
from 'rxjs'; import { tap, catchError, map } from 'rxjs/operators'; import
{ ApiService } from './api.service';



/**



  • Filter option for dropdowns


 */

export interface FilterOption { /* Option value (sent to API) / value: any; /
* Display label / label: string; /* Optional icon / icon?: string; /* Disabled
state / disabled?: boolean; }



/**



  • Cached options for a single filter


 */

export interface CachedFilterOptions { /* Filter field identifier / field: string; /*
API endpoint used / endpoint: string; /* Transformed options / options:
FilterOption[]; /* Raw API response (for custom transformers) / rawResponse?: any; /*
Cache timestamp / cachedAt: number; }



/**



  • Complete filter options cache


 */
```

```
export interface FilterOptionsCache { [endpoint: string]: CachedFilterOptions; }


/**


  • Filter options service

 *

  • Caches filter dropdown options and supports cross-window sharing.

 *

  • Key Features:

 *

  • 1. Cache-first loading — Returns cached options if available


  • 2. Transformer support — Adapts API responses to FilterOption[]


  • 3. Pop-out support — syncFromExternal() receives cache from main window


  • 4. Reactive cache — Observable of cache changes

 *

  • Usage Pattern:

 *

  • 1. Main window loads options (API call + cache)
```

```
  • 2. Main window broadcasts state (includes filterOptionsCache)


  • 3. Pop-out receives state


  • 4. Pop-out calls syncFromExternal(cache)


  • 5. Pop-out has options without API calls
*
  • @example


  •
```

typescript

- // Load with transformer
- this.filterOptions.getOptions(
- 'http://api/manufacturers',
- 'manufacturer',
- response => response.data.map(m => ({ value: m.id, label: m.name }))
-).subscribe(options => {
- this.manufacturerOptions = options;
- });

*

- // In pop-out, sync from main window
- this.filterOptions.syncFromExternal(message.payload.filterOptionsCache);

*

- // Check if cached
- if (this.filterOptions.isCached('http://api/manufacturers')) {
- // Use cached value
- }

```
 /
@Injectable({ providedIn: 'root' }) export class FilterOptionsService { /**


  • Cache storage

   */
private cache: FilterOptionsCache = {};


/**


  • Observable of cache state

   */
private cache$ = new BehaviorSubject<FilterOptionsCache>({});


/**


  • Constructor

   *

  • @param apiService - API service for fetching options

   */
constructor(private apiService: ApiService) {}


/**


  • Get current cache for state broadcast
```

```
   *
  • @returns Copy of current cache
   */
getCache(): FilterOptionsCache { return { ...this.cache }; }

/**
  • Get observable of cache changes
   *
  • @returns Observable of cache state
   */
getCache$(): Observable<FilterOptionsCache> { return this.cache$.asObservable(); }

/**
  • Get options (from cache or API)
   *
  • @param endpoint - API endpoint URL
  • @param field - Filter field identifier
  • @param transformer - Optional function to transform API response
```

```
   • @returns Observable of filter options

    */
getOptions( endpoint: string, field: string, transformer?: (response: any) =>
FilterOption[] ): Observable<FilterOption[]> { // Check cache first const cached =
this.cache[endpoint]; if (cached) { // If transformer provided and we have raw
response, re-transform if (transformer && cached.rawResponse) { return
of(transformer(cached.rawResponse)); } return of(cached.options); }



// Fetch from API return this.apiService.get(endpoint).pipe( tap(response => { const
options = transformer ? transformer(response) : []; this.cache[endpoint] = { field,
endpoint, options, rawResponse: response, cachedAt: Date.now() };
this.cache$.next({ ...this.cache }); }), map(response => transformer ?
transformer(response) : []), catchError(error =>
{ console.error( [FilterOptionsService] Failed to load from ${endpoint}: ,
error); throw error; }) ); }



/**



   • Get raw API response from cache

    *

   • Used for filters that need the full response (e.g., range filters).

    *

   • @param endpoint - API endpoint URL



   • @returns Cached raw response or null

    */
getRawResponse(endpoint: string): any | null { return
this.cache[endpoint]?.rawResponse ?? null; }
```

```
/**
   • Get raw response asynchronously (from cache or API)
    *
   • @param endpoint - API endpoint URL
   • @param field - Filter field identifier
   • @returns Observable of raw API response
    */
getRawResponseAsync(endpoint: string, field: string): Observable<any> { // Check cache
first const cached = this.cache[endpoint]; if (cached?.rawResponse) { return
of(cached.rawResponse); }

// Fetch from API return this.apiService.get(endpoint).pipe( tap(response =>
{ this.cache[endpoint] = { field, endpoint, options: [], rawResponse: response,
cachedAt: Date.now() }; this.cache$.next({ ...this.cache }); }), catchError(error =>
{ console.error( [FilterOptionsService] Failed to load from ${endpoint}:,
error); throw error; }) ); }

/**
   • Sync cache from external source
    *
   • Used by pop-out windows to receive cache from main window.
```

```
   *
  • @param externalCache - Cache from main window
   */
syncFromExternal(externalCache: FilterOptionsCache): void { if (!externalCache)
{ return; }

// Merge external cache into local cache this.cache =
{ ...this.cache, ...externalCache }; this.cache$.next({ ...this.cache }); }

/**
  • Check if options are cached
   *
  • @param endpoint - API endpoint URL
  • @returns True if cached
   */
isCached(endpoint: string): boolean { return !!this.cache[endpoint]; }

/**
  • Get cached options synchronously
   *
  • @param endpoint - API endpoint URL
```

```
   * @returns Cached options or null
   */
getCachedOptions(endpoint: string): FilterOption[] | null { return
this.cache[endpoint]?.options ?? null; }

/**
   * Invalidate specific cache entry
   * @param endpoint - API endpoint URL
   */
invalidate(endpoint: string): void { delete this.cache[endpoint];
this.cache$.next({ ...this.cache }); }

/**
   * Clear entire cache
   */
clearCache(): void { this.cache = {}; this.cache$.next({}); }

/**
   * Preload options for multiple endpoints
```

```
   *
  • Useful for initializing filters on page load.
   *
  • @param endpoints - Array of { endpoint, field, transformer }
  • @returns Observable that completes when all loaded
   */
preload( endpoints: Array<{ endpoint: string; field: string; transformer?: (response:
any) => FilterOption[]; }> ): Observable<void> { const requests =
endpoints.map(({ endpoint, field, transformer }) => this.getOptions(endpoint, field,
transformer) );

return new Observable(observer => { let completed = 0; const total = requests.length;

if (total === 0) { observer.next(); observer.complete(); return; }

requests.forEach(request => { request.subscribe({ next: () => { completed++; if
(completed === total) { observer.next(); observer.complete(); } }, error: err =>
{ completed++; console.warn('[FilterOptionsService] Preload error:', err); if
(completed === total) { observer.next(); observer.complete(); } } }); }); }); } }
```

Step 310.2: Update the Barrel File

Update `src/app/framework/services/index.ts`:

```
// src/app/framework/services/index.ts

// VERSION 9 (Section 310) - Added FilterOptionsService



export * from './url-state.service'; export * from './api.service'; export * from './
request-coordinator.service'; export * from './domain-config-registry.service'; export
* from './domain-config-validator.service'; export * from './popout-context.service';
export * from './popout-manager.service'; export * from './user-preferences.service';
export * from './filter-options.service'; export * from './resource-
management.service';
```

Verification

1. Check File Exists

```
$ ls -la src/app/framework/services/filter-options.service.ts
```

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/framework/services/filter-options.service.ts
```

3. Build the Application

```
$ ng build
```

4. Verify Caching (Optional)

```
// In any component

constructor(private filterOptions: FilterOptionsService) { // First call - API request
this.filterOptions.getOptions( 'http://example.com/api/options', 'testField', (res:
any) => res.data.map((d: any) => ({ value: d.id, label: d.name })) ).subscribe(options
=> { console.log('First call:', options); });



// Second call (1s later) - from cache setTimeout(() =>
{ this.filterOptions.getOptions( 'http://example.com/api/options',
'testField' ).subscribe(options => { console.log('Second call (cached):',
options); }); }, 1000); }
```

Network tab should show only 1 request.

Common Problems

Symptom	Cause	Solution
Options not cached	Different endpoint URLs	Ensure exact same URL string
Transformer not applied	Raw response missing	Check API returns data
Pop-out missing options	syncFromExternal not called	Call after receiving state
Cache stale	No invalidation	Call invalidate() when needed
Type errors	Transformer return type	Ensure returns FilterOption[]

Key Takeaways

- **Cache-first prevents duplicate requests** — Options loaded once, reused everywhere
- **Transformers adapt API responses** — Decouple option format from API format
- **External sync enables pop-outs** — No API calls in pop-out windows

---

Acceptance Criteria

- [ ] `src/app/framework/services/filter-options.service.ts` exists
- [ ] Service is `@Injectable({ providedIn: 'root' })`
- [ ] `getOptions()` returns from cache or fetches from API
- [ ] `getCache()` returns current cache for broadcasting
- [ ] `syncFromExternal()` merges external cache
- [ ] `isCached()` checks cache presence
- [ ] `getRawResponse()` and `getRawResponseAsync()` work
- [ ] `preload()` loads multiple endpoints
- [ ] `invalidate()` and `clearCache()` work
- [ ] Transformer pattern correctly adapts responses
- [ ] TypeScript compilation succeeds
- [ ] JSDoc comments document all public methods

Next Step

Proceed to `311-picker-config-registry.md` to create the service for managing picker configurations.

311: Picker Config Registry

Status: Complete **Depends On:** 205-picker-config-interface, 304-domain-config-registry **Blocks:** Phase 8 (Framework Components - for picker dialogs)

Learning Objectives

After completing this section, you will:

- Understand the Registry pattern for UI configurations
- Know how to decouple picker configuration from picker components
- Recognize the benefits of ID-based configuration lookup
- Be able to implement a type-safe configuration registry

Objective

Create the `PickerConfigRegistry` that provides centralized management of picker configurations. Pickers are dialogs that let users select items (vehicles, manufacturers, models) from searchable tables.

Why

Vvroom has multiple picker dialogs:

- **Vehicle Picker** — Select vehicles to compare
- **Manufacturer/Model Picker** — Filter by manufacturer and model
- **VIN Picker** — Find vehicles by VIN

Each picker needs configuration:

- Which columns to display
- How to fetch data
- Selection behavior (single vs multi-select)

The Configuration Challenge

Where should picker configuration live?

Option 1: In the picker component

```
@Component({...})
export class VehiclePickerComponent { columns = [ { field: 'manufacturer', header:
'Manufacturer' }, { field: 'model', header: 'Model' }, // ... ]; }
```

Problem: Every picker duplicates configuration logic.

Option 2: In the domain config

```
const AUTOMOBILE_CONFIG = {
pickers: [VEHICLE_PICKER_CONFIG, MODEL_PICKER_CONFIG] };
```

Problem: Components need to search arrays to find their config.

Option 3: In a registry (our approach)

```
// At initialization
registry.register(VEHICLE_PICKER_CONFIG);


// In component const config = registry.get('vehicle-picker');
```

Benefits:

- **O(1) lookup** by ID

- **Type-safe** retrieval with generics
- **Decoupled** from component hierarchy
- **Centralized** validation and management

Usage Pattern

```
// Domain config module registers pickers

@NgModule({...}) export class AutomobileDomainModule { constructor(private registry: PickerConfigRegistry) { registry.registerMultiple([ VEHICLE_PICKER_CONFIG, MANUFACTURER_MODEL_PICKER_CONFIG ]); } }



// Component looks up config by ID @Component({...}) export class VehicleSelectionComponent { pickerConfig = this.registry.get<VehicleResult>('vehicle-picker'); }



// Or use ID in template <app-base-picker [configId]="'vehicle-picker'"></app-base-picker>
```

What

Step 311.1: Create the Picker Config Registry Service

Create the file `src/app/framework/services/picker-config-registry.service.ts`:

```
// src/app/framework/services/picker-config-registry.service.ts
// VERSION 1 (Section 311) - Picker configuration registry

import { Injectable } from '@angular/core'; import { PickerConfig } from '../models/
picker-config.interface';

/**
  • Picker configuration registry service
 *
  • Centralized registry for managing picker configurations.

  • Allows registration and retrieval of picker configs by ID.
 *
  • Why a Registry?
 *
  • 1. O(1) lookup — Get config by ID without searching arrays

  • 2. Type safety — Generic retrieval preserves types

  • 3. Centralized — Single source of truth for picker configs

  • 4. Decoupled — Components don't need to know about domain config
```

```
 *
  • Typical Flow:
 *
  • 1. Domain module registers picker configs at init
  • 2. Components/templates reference pickers by ID
  • 3. BasePicker component looks up config from registry
 *
  • @example
  •
```

typescript

- // Registration (in domain module)
- constructor(private registry: PickerConfigRegistry) {
- registry.registerMultiple(AUTOMOBILE_PICKER_CONFIGS);
- }

*

- // Lookup (in component)
- const config = registry.get<VehicleResult>('vehicle-picker');

*

- // Template usage
- <app-base-picker [configId]="'vehicle-picker'"></app-base-picker>

```
 /
@Injectable({ providedIn: 'root' }) export class PickerConfigRegistry { /**


  • Storage for registered picker configurations

   */
private configs = new Map<string, PickerConfig<any>>();


/**


  • Register a picker configuration

   *

  • @template T - The data model type


  • @param config - Picker configuration to register

   *

  • @example


  •
```

typescript

- const vehicleConfig: PickerConfig<VehicleResult> = {
- id: 'vehicle-picker',
- displayName: 'Vehicle Selection',
- columns: [...],
- // ...
- };

*

- registry.register(vehicleConfig);

```
   /
register<T>(config: PickerConfig<T>): void { if (this.configs.has(config.id))
{ console.warn( [PickerConfigRegistry] Picker '${config.id}' already
registered. Overwriting. ); }



this.configs.set(config.id, config); }



/**



  • Register multiple picker configurations

   *

  • @param configs - Array of picker configurations

   *

  • @example



  •
```

typescript

- registry.registerMultiple([
- VEHICLE_PICKER_CONFIG,
- MANUFACTURER_MODEL_PICKER_CONFIG,
- VIN_PICKER_CONFIG
-]);

```
   /
registerMultiple(configs: PickerConfig<any>[]): void { configs.forEach(config =>
this.register(config)); }



/**



  • Get picker configuration by ID

   *

  • @template T - The data model type



  • @param id - Picker configuration ID



  • @returns Picker configuration



  • @throws Error if picker ID not found

   *

  • @example



  •
```

typescript

- const config = registry.get<VehicleResult>('vehicle-picker');
- // config is typed as PickerConfig<VehicleResult>

```
   /
get<T>(id: string): PickerConfig<T> { const config = this.configs.get(id);



if (!config) { const available = this.getAllIds().join(', ') || '(none)'; throw new
Error( [PickerConfigRegistry] Picker '${id}' not found. Available: $
{available} ); }



return config as PickerConfig<T>; }



/**



  • Get picker configuration by ID (returns null if not found)

   *

  • @template T - The data model type



  • @param id - Picker configuration ID



  • @returns Picker configuration or null

   */
tryGet<T>(id: string): PickerConfig<T> | null { return (this.configs.get(id) as
PickerConfig<T>) ?? null; }



/**



  • Check if picker configuration exists
```

```
   *
  • @param id - Picker configuration ID
  • @returns True if picker exists
   */
has(id: string): boolean { return this.configs.has(id); }

/**
  • Get all registered picker IDs
   *
  • @returns Array of picker IDs
   */
getAllIds(): string[] { return Array.from(this.configs.keys()); }

/**
  • Get all registered picker configurations
   *
  • @returns Array of all picker configurations
   */
getAll(): PickerConfig<any>[] { return Array.from(this.configs.values()); }
```

```
/**
   * Unregister a picker configuration
   *
   * @param id - Picker configuration ID
   *
   * @returns True if removed, false if not found
   */
unregister(id: string): boolean { return this.configs.delete(id); }

/**
   * Clear all registered picker configurations
   */
clear(): void { this.configs.clear(); }

/**
   * Get count of registered pickers
   *
   * @returns Number of registered pickers
   */
```

```
getCount(): number { return this.configs.size; }



/**



  • Get pickers by category (if categories are defined)

   *

  • @param category - Category to filter by



  • @returns Array of picker configurations in that category

   */
getByCategory(category: string): PickerConfig<any>[] { return
this.getAll().filter(config => config.category === category); } }
```

Step 311.2: Update the Barrel File

Update `src/app/framework/services/index.ts`:

```
// src/app/framework/services/index.ts

// VERSION 10 (Section 311) - Added PickerConfigRegistry



export * from './url-state.service'; export * from './api.service'; export * from './
request-coordinator.service'; export * from './domain-config-registry.service'; export
* from './domain-config-validator.service'; export * from './popout-context.service';
export * from './popout-manager.service'; export * from './user-preferences.service';
export * from './filter-options.service'; export * from './picker-config-
registry.service'; export * from './resource-management.service';
```

Step 311.3: Ensure PickerConfig Interface Exists

Verify `src/app/framework/models/picker-config.interface.ts` includes the `category` property:

```
// Add to PickerConfig interface if not present
export interface PickerConfig<T> { /* Unique picker identifier */ id: string;

/* Display name for the picker dialog */ displayName: string;

/* Columns to display in picker table */ columns: ColumnConfig[];

/* Optional category for grouping */ category?: string;

// ... other properties from Section 205 }
```

Verification

1. Check File Exists

```
$ ls -la src/app/framework/services/picker-config-registry.service.ts
```

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/framework/services/picker-config-registry.service.ts
```

3. Build the Application

```
$ ng build
```

4. Verify Registration (Optional)

```
// In any component

constructor(private pickerRegistry: PickerConfigRegistry) { // Register test config
this.pickerRegistry.register({ id: 'test-picker', displayName: 'Test Picker', columns:
[{ field: 'name', header: 'Name' }] } as any);



console.log('Registered pickers:', this.pickerRegistry.getAllIds()); console.log('Test
picker:', this.pickerRegistry.get('test-picker')); }
```

Common Problems

Symptom	Cause	Solution
`Picker 'X' not found`	Not registered	Register before component init
Registration order issues	Component loads before module	Use APP_INITIALIZER
Type errors on get<T>()	Wrong type parameter	Check data model matches config
Config overwritten warning	Same ID registered twice	Use unique IDs

---

Key Takeaways

- **O(1) lookup beats array search** — Registry is faster than filtering arrays
- **ID-based access decouples components** — Components don't know about domain config structure
- **Type-safe retrieval preserves generics** — get<VehicleResult>() returns typed config

---

Acceptance Criteria

- [ ] `src/app/framework/services/picker-config-registry.service.ts` exists
- [ ] Service is `@Injectable({ providedIn: 'root' })`
- [ ] `register()` stores config by ID
- [ ] `registerMultiple()` registers array of configs
- [ ] `get<T>()` returns typed config, throws if not found
- [ ] `tryGet<T>()` returns null instead of throwing
- [ ] `has()` checks if picker exists
- [ ] `getAllIds()` returns all registered IDs
- [ ] `getAll()` returns all configs
- [ ] `unregister()` removes config
- [ ] `clear()` removes all configs
- [ ] `getByCategory()` filters by category
- [ ] TypeScript compilation succeeds
- [ ] JSDoc comments document all public methods

Phase 3B Complete

Congratulations! You have completed Phase 3B: Popout & Specialized Services.

What you built:

- PopOutContextService — Pop-out window detection and messaging
- PopOutManagerService — Pop-out window management from main window
- UserPreferencesService — localStorage-based preferences
- FilterOptionsService — Filter dropdown caching
- PickerConfigRegistry — Picker configuration lookup

The Aha Moment: "Pop-out windows share state with the parent through a coordination service."

Next Step

Proceed to `312-error-notification-service.md` to begin Phase 3C: Error Handling.

312: Error Notification Service

Status: Complete **Depends On:** 201-209 (Framework Models) **Blocks:** 313-http-error-interceptor, 314-global-error-handler

---

Learning Objectives

After completing this section, you will:

- Understand the need for centralized error notification
- Know how to deduplicate error messages to prevent notification spam
- Recognize the role of error categorization in user-facing messaging
- Be able to implement toast notifications using PrimeNG MessageService

---

Objective

Create the `ErrorNotificationService` that provides centralized error notification using PrimeNG Toast. This service handles error deduplication, categorization, and consistent display for all user-facing error messages.

Why

Applications generate errors from multiple sources:

- HTTP requests failing
- Validation errors from form inputs
- Component lifecycle errors
- Unhandled promise rejections

Without centralized error handling, each component handles errors differently. Users see inconsistent messages, and developers repeat the same error-handling logic.

The Problem: Notification Spam

When errors occur, they often repeat rapidly:

- Network failures cause multiple retries
- Polling requests all fail simultaneously
- Component re-renders trigger the same error

Showing every error individually creates **notification spam** that overwhelms users.

The Solution: Deduplication

The ErrorNotificationService maintains a short-term memory of recent errors. If the same error (same category + summary + detail) occurs within a deduplication window (3 seconds), subsequent errors are suppressed.

```
// First error: Shows toast
showError('Connection Error', 'Unable to reach server');



// Same error within 3 seconds: Suppressed showError('Connection Error', 'Unable to
reach server'); // No toast



// After 3 seconds: Shows toast again showError('Connection Error', 'Unable to reach
server'); // Toast shown
```

Error Categories

Errors are categorized by source and type:

Category	Description	Default Severity
`NETWORK`	Connection issues, timeouts	error
`VALIDATION`	Invalid input, business rules	warn
`AUTHORIZATION`	401/403 errors	warn
`SERVER`	5xx status codes	error
`CLIENT`	JavaScript errors	error
`APPLICATION`	Business logic errors	error
`UNKNOWN`	Uncategorized	error

Categories drive two behaviors:

- **Visual severity** — Maps to PrimeNG toast colors (error=red, warn=yellow)
- **User-friendly summaries** — "Connection Error" instead of "NETWORK"

PrimeNG Toast Integration

PrimeNG's `MessageService` provides toast notifications. The ErrorNotificationService wraps it to add:

- Error categorization
- Deduplication
- Consistent formatting
- Debug logging

What

Step 312.1: Create the Error Notification Interface

Create the file `src/app/framework/models/error-notification.interface.ts`:

```
// src/app/framework/models/error-notification.interface.ts
// VERSION 1 (Section 312) - Error notification types


/**


  • Error category enumeration

 *

  • Categorizes errors by their source and nature for appropriate handling



  • and user messaging.

 */
export enum ErrorCategory { /**


  • Network-related errors (connection issues, timeouts, etc.)

   */
NETWORK = 'NETWORK',


/**


  • Validation errors (invalid input, business rule violations)

   */
VALIDATION = 'VALIDATION',


/**
```

```
  • Authorization/authentication errors (401, 403)

   */
AUTHORIZATION = 'AUTHORIZATION',


/**

  • Server-side errors (5xx status codes)

   */
SERVER = 'SERVER',


/**

  • Client-side errors (JavaScript errors, component errors)

   */
CLIENT = 'CLIENT',


/**

  • Application-level errors (business logic, state errors)

   */
APPLICATION = 'APPLICATION',


/**
```

```
  • Unknown or uncategorized errors

   */
UNKNOWN = 'UNKNOWN' }


/**

  • Error severity levels

 *

  • Maps to PrimeNG Toast severity levels for visual feedback

 */
export type ErrorSeverity = 'success' | 'info' | 'warn' | 'error';


/**

  • Error notification data structure

 *

  • Contains all information needed to display an error to the user

 */
export interface ErrorNotification { /**

  • Error category for categorization and routing

   */
```

```
category: ErrorCategory;


/**


  • Display severity (maps to PrimeNG Toast severity)

   */
severity: ErrorSeverity;


/**


  • Brief error summary (shown as toast title)

   */
summary: string;


/**


  • Detailed error message (shown as toast body)

   */
detail: string;


/**


  • Optional error code for debugging

   */
```

```
code?: string;


/**


  • Optional timestamp of when error occurred

   */
timestamp?: string;


/**


  • Optional URL where error occurred

   */
url?: string;


/**


  • Optional HTTP status code (for network errors)

   */
status?: number;


/**


  • Original error object (for logging/debugging)

   */
```

```
originalError?: any; }


/**


  • Error display options

 *

  • Configuration for how errors should be displayed to users

 */
export interface ErrorDisplayOptions { /**


  • Auto-hide duration in milliseconds


  • Set to 0 or null to prevent auto-hide


  • Default: 5000 (5 seconds)

   */
life?: number;


/**


  • Whether to show close button


  • Default: true

   */
```

```
closable?: boolean;


/**


  • Whether to show in sticky mode (no auto-hide)


  • Default: false

   */
sticky?: boolean;


/**


  • Custom CSS class for the toast

   */
styleClass?: string;


/**


  • Toast position key


  • Default: 'app-toast'

   */
key?: string; }
```

```
/**


  • Default error display options


 */
export const DEFAULT_ERROR_DISPLAY_OPTIONS: ErrorDisplayOptions = { life: 5000,
closable: true, sticky: false, key: 'app-toast' };



/**


  • Error severity mapping configuration


 *

  • Maps error categories to default severity levels


 */
export const ERROR_CATEGORY_SEVERITY_MAP: Record<ErrorCategory, ErrorSeverity> =
{ [ErrorCategory.NETWORK]: 'error', [ErrorCategory.VALIDATION]: 'warn',
[ErrorCategory.AUTHORIZATION]: 'warn', [ErrorCategory.SERVER]: 'error',
[ErrorCategory.CLIENT]: 'error', [ErrorCategory.APPLICATION]: 'error',
[ErrorCategory.UNKNOWN]: 'error' };



/**


  • HTTP status code to error category mapping


 *

  • @param status - HTTP status code


  • @returns Appropriate error category
```

```
 */
export function getErrorCategoryFromStatus(status: number): ErrorCategory { if (status
=== 0) { return ErrorCategory.NETWORK; }



if (status === 401 || status === 403) { return ErrorCategory.AUTHORIZATION; }



if (status === 400 || status === 422) { return ErrorCategory.VALIDATION; }



if (status >= 500 && status < 600) { return ErrorCategory.SERVER; }



if (status >= 400 && status < 500) { return ErrorCategory.CLIENT; }



return ErrorCategory.UNKNOWN; }



/**



  • Error code to error category mapping

 *

  • @param code - Error code string



  • @returns Appropriate error category

 */
export function getErrorCategoryFromCode(code: string): ErrorCategory { const
upperCode = code.toUpperCase();
```

```
if (upperCode.includes('NETWORK') || upperCode.includes('TIMEOUT')) { return
ErrorCategory.NETWORK; }

if (upperCode.includes('VALIDATION') || upperCode.includes('INVALID')) { return
ErrorCategory.VALIDATION; }

if ( upperCode.includes('UNAUTHORIZED') || upperCode.includes('FORBIDDEN') ||
upperCode.includes('AUTH') ) { return ErrorCategory.AUTHORIZATION; }

if (upperCode.includes('SERVER') || upperCode.includes('INTERNAL')) { return
ErrorCategory.SERVER; }

if (upperCode.includes('CLIENT')) { return ErrorCategory.CLIENT; }

return ErrorCategory.UNKNOWN; }

/**

  • Get user-friendly summary for error category
 *
  • @param category - Error category

  • @returns Summary text
 */
```

```
export function getSummaryForCategory(category: ErrorCategory): string { switch
(category) { case ErrorCategory.NETWORK: return 'Connection Error'; case
ErrorCategory.VALIDATION: return 'Validation Error'; case ErrorCategory.AUTHORIZATION:
return 'Access Denied'; case ErrorCategory.SERVER: return 'Server Error'; case
ErrorCategory.CLIENT: return 'Application Error'; case ErrorCategory.APPLICATION:
return 'Operation Failed'; case ErrorCategory.UNKNOWN: default: return 'Error'; } }



/**



  • Create error notification from HTTP error


 *


  • @param error - HTTP error object (from interceptor)



  • @returns ErrorNotification object


 */
export function createErrorNotificationFromHttpError(error: any): ErrorNotification
{ const status = error.status || 0; const code = error.code || 'UNKNOWN_ERROR'; const
category = getErrorCategoryFromStatus(status); const severity =
ERROR_CATEGORY_SEVERITY_MAP[category];



return { category, severity, summary: getSummaryForCategory(category), detail:
error.message || 'An unexpected error occurred', code, timestamp: error.timestamp,
url: error.url, status, originalError: error }; }



/**



  • Create error notification from generic error


 *


  • @param error - Generic error object
```

```
  • @returns ErrorNotification object


 */
export function createErrorNotificationFromError(error: Error): ErrorNotification
{ const code = (error as any).code; const category = code ?
getErrorCategoryFromCode(code) : ErrorCategory.CLIENT; const severity =
ERROR_CATEGORY_SEVERITY_MAP[category];



return { category, severity, summary: getSummaryForCategory(category), detail:
error.message || 'An unexpected error occurred', code, timestamp: new
Date().toISOString(), originalError: error }; }
```

Step 312.2: Update the Models Barrel File

Update `src/app/framework/models/index.ts`:

```
// src/app/framework/models/index.ts

// VERSION 9 (Section 312) - Added error notification types



export * from './domain-config.interface'; export * from './column-config.interface';
export * from './filter-config.interface'; export * from './picker-config.interface';
export * from './table-state.interface'; export * from './paginator-state.interface';
export * from './popout.interface'; export * from './user-preferences.interface';
export * from './error-notification.interface';
```

Step 312.3: Create the Error Notification Service

Create the file `src/app/framework/services/error-notification.service.ts`:

```
// src/app/framework/services/error-notification.service.ts
// VERSION 1 (Section 312) - Centralized error notification

import { Injectable, OnDestroy } from '@angular/core'; import { MessageService } from
'primeng/api'; import { ErrorNotification, ErrorCategory, ErrorDisplayOptions,
DEFAULT_ERROR_DISPLAY_OPTIONS, createErrorNotificationFromHttpError,
createErrorNotificationFromError } from '../models/error-notification.interface';

/**
  • Error notification service
 *
  • Centralized error notification system using PrimeNG Toast for user-facing
  • error messages. Provides error deduplication, categorization, and
  • consistent error display.
 *
  • Key Features:
 *
  • 1. Deduplication – Same error within 3 seconds is suppressed
  • 2. Categorization – Errors categorized by type for appropriate messaging
```

```
  • 3. Consistent Display — All errors shown via PrimeNG Toast


  • 4. Debug Logging — All errors logged to console for debugging

 *

  • @example


  •
```

typescript

• // Show HTTP error

• this.errorNotification.showHttpError(httpError);

*

• // Show custom error

• this.errorNotification.showError(

• 'Operation Failed',

• 'Unable to save changes. Please try again.'

•);

*

• // Show warning

• this.errorNotification.showWarning(

• 'Data Modified',

• 'Some fields were automatically corrected.'

•);

```
 /
@Injectable({ providedIn: 'root' }) export class ErrorNotificationService implements
OnDestroy { /**



  • Recent error messages for deduplication

   *

  • Maps error signature (hash of message) to timestamp. Used to suppress



  • duplicate errors shown within DEDUPLICATION_WINDOW milliseconds.

   */
private recentErrors = new Map<string, number>();



/**



  • Deduplication window in milliseconds

   *

  • Errors with the same signature within this window are suppressed



  • to prevent notification spam from repeated failures.

   */
private readonly DEDUPLICATION_WINDOW = 3000; // 3 seconds



/**
```

```
   • Cleanup interval for recent errors map
    *
   • Interval at which expired entries are removed from recentErrors
   • to prevent unbounded memory growth.
    */
private readonly CLEANUP_INTERVAL = 10000; // 10 seconds

/**
   • Cleanup timer reference
    */
private cleanupTimer?: ReturnType<typeof setInterval>;

/**
   • Constructor for dependency injection
    *
   • @param messageService - PrimeNG MessageService for displaying toast notifications
    */
constructor(private messageService: MessageService) { this.startCleanupTimer(); }
```

```
/**


  • Show error notification

   *

  • @param summary - Brief error summary


  • @param detail - Detailed error message


  • @param options - Display options

   */
showError( summary: string, detail: string, options?: ErrorDisplayOptions ): void
{ this.show( { category: ErrorCategory.APPLICATION, severity: 'error', summary,
detail, timestamp: new Date().toISOString() }, options ); }



/**


  • Show warning notification

   *

  • @param summary - Brief warning summary


  • @param detail - Detailed warning message


  • @param options - Display options

   */
```

```
showWarning( summary: string, detail: string, options?: ErrorDisplayOptions ): void
{ this.show( { category: ErrorCategory.APPLICATION, severity: 'warn', summary, detail,
timestamp: new Date().toISOString() }, options ); }



/**



  • Show info notification

   *

  • @param summary - Brief info summary



  • @param detail - Detailed info message



  • @param options - Display options

   */
showInfo( summary: string, detail: string, options?: ErrorDisplayOptions ): void
{ this.show( { category: ErrorCategory.APPLICATION, severity: 'info', summary, detail,
timestamp: new Date().toISOString() }, options ); }



/**



  • Show success notification

   *

  • @param summary - Brief success summary



  • @param detail - Detailed success message
```

```
   * @param options - Display options

    */
showSuccess( summary: string, detail: string, options?: ErrorDisplayOptions ): void
{ this.show( { category: ErrorCategory.APPLICATION, severity: 'success', summary,
detail, timestamp: new Date().toISOString() }, options ); }



/**



   * Show HTTP error notification

    *

   * Automatically categorizes and formats HTTP errors from the interceptor

    *

   * @param error - HTTP error object from interceptor



   * @param options - Display options

    */
showHttpError(error: any, options?: ErrorDisplayOptions): void { const notification =
createErrorNotificationFromHttpError(error); this.show(notification, options); }



/**



   * Show generic error notification

    *

   * Automatically categorizes and formats generic JavaScript errors
```

```
   *

  • @param error - Error object



  • @param options - Display options

   */
showGenericError(error: Error, options?: ErrorDisplayOptions): void { const
notification = createErrorNotificationFromError(error); this.show(notification,
options); }



/**



  • Show custom error notification

   *

  • @param notification - Error notification object



  • @param options - Display options

   */
show( notification: ErrorNotification, options?: ErrorDisplayOptions ): void { //
Check for duplicate if (this.isDuplicate(notification)) { console.debug('Suppressing
duplicate error:', notification.summary); return; }



// Record error for deduplication this.recordError(notification);



// Merge options with defaults const displayOptions: ErrorDisplayOptions =
{ ...DEFAULT_ERROR_DISPLAY_OPTIONS, ...options };
```

```
// Display toast notification this.messageService.add({ severity:
notification.severity, summary: notification.summary, detail: notification.detail,
life: displayOptions.life, closable: displayOptions.closable, sticky:
displayOptions.sticky, styleClass: displayOptions.styleClass, key:
displayOptions.key });



// Log error details this.logError(notification); }



/**



  • Clear all notifications

   */
clearAll(): void { this.messageService.clear(); }



/**



  • Clear notifications by key

   *

  • @param key - Toast key to clear

   */
clear(key?: string): void { this.messageService.clear(key); }



/**



  • Check if error is duplicate
```

```
   *
  • @param notification - Error notification
  • @returns True if duplicate within deduplication window
   */
private isDuplicate(notification: ErrorNotification): boolean { const signature =
this.getErrorSignature(notification); const lastTime =
this.recentErrors.get(signature);

if (!lastTime) { return false; }

const now = Date.now(); return now - lastTime < this.DEDUPLICATION_WINDOW; }

/**
  • Record error for deduplication tracking
   *
  • @param notification - Error notification
   */
private recordError(notification: ErrorNotification): void { const signature =
this.getErrorSignature(notification); this.recentErrors.set(signature, Date.now()); }

/**
  • Generate error signature for deduplication
```

```
   *

  • @param notification - Error notification



  • @returns Unique signature string

   */
private getErrorSignature(notification: ErrorNotification): string { // Combine
category, summary, and detail for signature return `${notification.category}:$
{notification.summary}:${notification.detail}`; }



/**



  • Log error details for debugging

   *

  • @param notification - Error notification

   */
private logError(notification: ErrorNotification): void { const logData: any =
{ category: notification.category, severity: notification.severity, summary:
notification.summary, detail: notification.detail, timestamp: notification.timestamp
|| new Date().toISOString() };



// Add optional fields if present if (notification.code) { logData.code =
notification.code; } if (notification.url) { logData.url = notification.url; } if
(notification.status) { logData.status = notification.status; }



// Log based on severity switch (notification.severity) { case 'error':
console.error('[Error Notification]', logData); if (notification.originalError)
{ console.error('Original error:', notification.originalError); } break; case 'warn':
console.warn('[Warning Notification]', logData); break; case 'info':
```

```
console.info('[Info Notification]', logData); break; case 'success':
console.log('[Success Notification]', logData); break; } }



/**



  • Start cleanup timer for recent errors map


   */

private startCleanupTimer(): void { this.cleanupTimer = setInterval(() =>
{ this.cleanupRecentErrors(); }, this.CLEANUP_INTERVAL); }



/**



  • Clean up expired entries from recent errors map


   */

private cleanupRecentErrors(): void { const now = Date.now(); const expiredKeys:
string[] = [];



// Find expired entries this.recentErrors.forEach((timestamp, key) => { if (now -
timestamp > this.DEDUPLICATION_WINDOW * 2) { expiredKeys.push(key); } });



// Remove expired entries expiredKeys.forEach((key) =>
{ this.recentErrors.delete(key); }); }



/**



  • Stop cleanup timer (for cleanup)


   */
```

```
ngOnDestroy(): void { if (this.cleanupTimer) { clearInterval(this.cleanupTimer); } } }
```

Step 312.4: Update the Services Barrel File

Update `src/app/framework/services/index.ts`:

```
// src/app/framework/services/index.ts
// VERSION 11 (Section 312) - Added ErrorNotificationService



export * from './url-state.service'; export * from './api.service'; export * from './
request-coordinator.service'; export * from './domain-config-registry.service'; export
* from './domain-config-validator.service'; export * from './popout-context.service';
export * from './popout-manager.service'; export * from './user-preferences.service';
export * from './filter-options.service'; export * from './picker-config-
registry.service'; export * from './resource-management.service'; export * from './
error-notification.service';
```

Verification

1. Check Files Exist

```
$ ls -la src/app/framework/models/error-notification.interface.ts
$ ls -la src/app/framework/services/error-notification.service.ts
```

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/framework/services/error-notification.service.ts
```

3. Verify PrimeNG MessageService Provider

Ensure `MessageService` is provided in your `app.module.ts`:

```
import { MessageService } from 'primeng/api';

@NgModule({ providers: [MessageService], // ... }) export class AppModule { }
```

4. Add Toast Component to Template

Add the PrimeNG Toast component to `app.component.html`:

```
<p-toast key="app-toast"></p-toast>

<router-outlet></router-outlet>
```

5. Test Deduplication

```
// In any component

constructor(private errorNotification: ErrorNotificationService) { // First error -
shows toast this.errorNotification.showError('Test', 'This is a test error');



// Same error within 3 seconds - suppressed setTimeout(() =>
{ this.errorNotification.showError('Test', 'This is a test error');
console.log('Second error call (should be suppressed)'); }, 1000);



// Same error after 4 seconds - shows toast setTimeout(() =>
{ this.errorNotification.showError('Test', 'This is a test error'); console.log('Third
error call (should show)'); }, 4000); }
```

Common Problems

Symptom	Cause	Solution
No toast appears	MessageService not provided	Add `MessageService` to app.module.ts providers
Toast appears but no styling	PrimeNG CSS not imported	Import PrimeNG theme in styles.scss
Multiple toasts for same error	Deduplication not working	Check signature generation logic
Memory leak warning	Cleanup timer not stopped	Ensure `ngOnDestroy` clears interval
Wrong toast position	Toast key mismatch	Ensure `key` matches in both service and template

---

Key Takeaways

- **Deduplication prevents spam** — Same error within 3 seconds is suppressed
- **Categorization drives messaging** — Error category determines summary text and severity
- **PrimeNG Toast integration** — Wrapping MessageService adds value through deduplication and categorization

---

Acceptance Criteria

- [ ] `src/app/framework/models/error-notification.interface.ts` exists with all types
- [ ] `src/app/framework/services/error-notification.service.ts` exists
- [ ] Service is `@Injectable({ providedIn: 'root' })`
- [ ] `showError()` displays error toast
- [ ] `showWarning()` displays warning toast
- [ ] `showInfo()` displays info toast
- [ ] `showSuccess()` displays success toast
- [ ] `showHttpError()` creates notification from HTTP error
- [ ] `showGenericError()` creates notification from Error object
- [ ] Deduplication prevents same error within 3 seconds
- [ ] Cleanup timer prevents memory leaks
- [ ] `ngOnDestroy()` clears cleanup timer

- [ ] TypeScript compilation succeeds
- [ ] JSDoc comments document all public methods

Next Step

Proceed to `313-http-error-interceptor.md` to create the HTTP interceptor for automatic error handling.

313: HTTP Error Interceptor

Status: Complete **Depends On:** 302-api-service, 312-error-notification-service **Blocks:** 314-global-error-handler

---

Learning Objectives

After completing this section, you will:

- Understand how Angular HTTP interceptors work
- Know how to implement automatic retry for transient errors
- Recognize the importance of consistent error formatting
- Be able to create user-friendly error messages from HTTP status codes

---

Objective

Create the `HttpErrorInterceptor` that provides global error handling for all HTTP requests. This interceptor automatically retries transient errors, formats error responses consistently, and logs errors for debugging.

Why

Every HTTP request can fail. Without centralized handling, each component must:

- Catch errors in its subscription
- Determine the error type
- Format user-friendly messages
- Log for debugging

- Decide whether to retry

This leads to duplicated logic and inconsistent error handling.

The Interceptor Pattern

Angular's HTTP interceptor pattern allows you to intercept and modify HTTP requests and responses globally. For error handling:

```
Request → Interceptor → HTTP → Server Response
↓ Error? → Retry? → Format → Component
```

Automatic Retry for Transient Errors

Some errors are **transient** — they succeed on retry:

Status Code	Meaning	Retryable?
429	Too Many Requests	Yes — wait and retry
500	Internal Server Error	Maybe — server might recover
502	Bad Gateway	Yes — proxy might recover
503	Service Unavailable	Yes — server restarting
504	Gateway Timeout	Yes — might just be slow
400	Bad Request	No — client error won't change
401	Unauthorized	No — need authentication
404	Not Found	No — resource doesn't exist

The interceptor automatically retries for status codes 429, 500, 502, 503, 504 up to 2 times before giving up.

Consistent Error Formatting

The interceptor transforms Angular's `HttpErrorResponse` into a consistent format:

```
// Input: HttpErrorResponse

{ status: 500, statusText: 'Internal Server Error', error: { message: 'Database
connection failed' }, url: '/api/vehicles' }



// Output: Formatted error object { code: 'INTERNAL_SERVER_ERROR', message: 'Internal
server error. Please try again later.', status: 500, statusText: 'Internal Server
Error', url: '/api/vehicles', timestamp: '2024-02-09T12:00:00.000Z' }
```

This consistent format makes error handling predictable for:

- The ErrorNotificationService
- Component error handlers
- Error logging systems

Error Code Mapping

Each HTTP status maps to a semantic error code:

Status	Error Code
400	`BAD_REQUEST`
401	`UNAUTHORIZED`
403	`FORBIDDEN`
404	`NOT_FOUND`
409	`CONFLICT`
422	`VALIDATION_ERROR`
429	`RATE_LIMIT_EXCEEDED`
500	`INTERNAL_SERVER_ERROR`
502	`BAD_GATEWAY`
503	`SERVICE_UNAVAILABLE`
504	`GATEWAY_TIMEOUT`

What

Step 313.1: Create the HTTP Error Interceptor

Create the file `src/app/framework/services/http-error.interceptor.ts`:

```
// src/app/framework/services/http-error.interceptor.ts

// VERSION 1 (Section 313) - HTTP error interceptor



import { Injectable } from '@angular/core'; import { HttpErrorResponse, HttpEvent,
HttpHandler, HttpInterceptor, HttpRequest } from '@angular/common/http'; import
{ Observable, throwError } from 'rxjs'; import { catchError, retry } from 'rxjs/
operators';



/**



  • Configuration for error interceptor retry behavior


 */

export interface RetryConfig { /* Maximum number of retry attempts / maxRetries:
number; /* HTTP status codes that should trigger retry / retryableStatusCodes:
number[]; }



/**



  • Default retry configuration


 *

  • Retries transient server errors (5xx) and rate limiting (429).



  • Does not retry client errors (4xx) as they won't succeed without changes.


 */

const DEFAULT_RETRY_CONFIG: RetryConfig = { maxRetries: 2, retryableStatusCodes: [429,
500, 502, 503, 504] };
```

```
/**


  • Map HTTP status code to semantic error code


 *


  • @param status - HTTP status code



  • @returns Semantic error code string


 */
function getErrorCode(status: number): string { switch (status) { case 400: return
'BAD_REQUEST'; case 401: return 'UNAUTHORIZED'; case 403: return 'FORBIDDEN'; case
404: return 'NOT_FOUND'; case 409: return 'CONFLICT'; case 422: return
'VALIDATION_ERROR'; case 429: return 'RATE_LIMIT_EXCEEDED'; case 500: return
'INTERNAL_SERVER_ERROR'; case 502: return 'BAD_GATEWAY'; case 503: return
'SERVICE_UNAVAILABLE'; case 504: return 'GATEWAY_TIMEOUT'; default: return status >=
500 ? 'SERVER_ERROR' : 'CLIENT_ERROR'; } }



/**


  • Get user-friendly error message from HTTP error


 *


  • Attempts to extract message from error response body,



  • falls back to standard messages based on status code.


 *


  • @param error - HTTP error response
```

```
  • @returns User-friendly error message

 */
function getErrorMessage(error: HttpErrorResponse): string { // Try to get message
from nested error object if (error.error?.error?.message) { return
error.error.error.message; }



// Try to get message from error object if (error.error?.message) { return
error.error.message; }



// Fall back to status-based messages switch (error.status) { case 0: return 'Unable
to connect to server. Please check your network connection.'; case 400: return
'Invalid request. Please check your input.'; case 401: return 'Authentication
required. Please log in.'; case 403: return 'Access denied. You do not have permission
to access this resource.'; case 404: return 'Resource not found.'; case 409: return
'Conflict. The resource already exists or is in an invalid state.'; case 422: return
'Validation failed. Please check your input.'; case 429: return 'Too many requests.
Please wait and try again.'; case 500: return 'Internal server error. Please try again
later.'; case 502: return 'Bad gateway. The server is temporarily unavailable.'; case
503: return 'Service unavailable. Please try again later.'; case 504: return 'Gateway
timeout. The server took too long to respond.'; default: return error.message ||
HTTP error: ${error.status} ${error.statusText}; } }



/**



  • Log error details for debugging

 *

  • @param error - HTTP error response



  • @param request - Original HTTP request
```

```
  • @param errorCode - Formatted error code



  • @param errorMessage - Formatted error message

 */
function logError( error: HttpErrorResponse, request: HttpRequest<unknown>, errorCode:
string, errorMessage: string ): void { const logDetails = { code: errorCode, message:
errorMessage, status: error.status, statusText: error.statusText, url: request.url,
method: request.method, timestamp: new Date().toISOString() };



console.error('HTTP Error:', logDetails);



// Log response body for additional context if (error.error) { console.error('Error
details:', error.error); } }



/**



  • Handle HTTP error and format response

 *

  • @param error - HTTP error response



  • @param request - Original HTTP request



  • @returns Observable that throws formatted error

 */
function handleError(error: HttpErrorResponse, request: HttpRequest<unknown>) { let
errorMessage: string; let errorCode: string;
```

```
// Check if error is client-side (ErrorEvent) or server-side if (error.error
instanceof ErrorEvent) { // Client-side error (network issue, etc.) errorCode =
'CLIENT_ERROR'; errorMessage = `Network error: ${error.error.message}`; } else { // Server-side error errorCode =
getErrorCode(error.status); errorMessage = getErrorMessage(error); }



// Log for debugging logError(error, request, errorCode, errorMessage);



// Return formatted error object return throwError(() => ({ code: errorCode, message:
errorMessage, status: error.status, statusText: error.statusText, url: request.url,
timestamp: new Date().toISOString() })); }



/**



  • HTTP error interceptor

 *

  • Handles global error processing for all HTTP requests:



  • - Automatic retry for transient errors (5xx, 429)



  • - Consistent error formatting



  • - Error logging

 *

  • Retry Behavior:
```

```
 *
  • The interceptor retries failed requests up to 2 times for specific
  • status codes (429, 500, 502, 503, 504). This handles transient
  • failures that often succeed on retry.
 *
  • Error Format:
 *
  • All errors are transformed to a consistent format:
  •
```

typescript

- {
- code: string; // Semantic error code (e.g., 'UNAUTHORIZED')
- message: string; // User-friendly message
- status: number; // HTTP status code
- statusText: string;
- url: string;
- timestamp: string;
- }

- @example
-

typescript

- // In app.module.ts
- providers: [
- {
- provide: HTTP_INTERCEPTORS,
- useClass: HttpErrorInterceptor,
- multi: true
- }
-]

```
 /
@Injectable() export class HttpErrorInterceptor implements HttpInterceptor { /**


  • Intercept HTTP requests and handle errors

   *

  • @param request - Outgoing HTTP request


  • @param next - HTTP handler for the next interceptor


  • @returns Observable of HTTP events

   */
intercept( request: HttpRequest<unknown>, next: HttpHandler ):
Observable<HttpEvent<unknown>> { const retryConfig = DEFAULT_RETRY_CONFIG;



return next.handle(request).pipe( // Retry transient errors
retry(retryConfig.maxRetries), // Catch and format errors catchError((error:
HttpErrorResponse) => { return handleError(error, request); }) ); } }
```

Step 313.2: Update the Services Barrel File

Update `src/app/framework/services/index.ts`:

```
// src/app/framework/services/index.ts

// VERSION 12 (Section 313) - Added HttpErrorInterceptor



export * from './url-state.service'; export * from './api.service'; export * from './
request-coordinator.service'; export * from './domain-config-registry.service'; export
* from './domain-config-validator.service'; export * from './popout-context.service';
export * from './popout-manager.service'; export * from './user-preferences.service';
export * from './filter-options.service'; export * from './picker-config-
registry.service'; export * from './resource-management.service'; export * from './
error-notification.service'; export * from './http-error.interceptor';
```

Step 313.3: Register the Interceptor

Update `src/app/app.module.ts` to register the interceptor:

```
// src/app/app.module.ts (partial - add to existing)

import { HTTP_INTERCEPTORS } from '@angular/common/http'; import
{ HttpErrorInterceptor } from './framework/services';



@NgModule({ // ... providers: [ { provide: HTTP_INTERCEPTORS, useClass:
HttpErrorInterceptor, multi: true } ], // ... }) export class AppModule { }
```

Verification

1. Check File Exists

```
$ ls -la src/app/framework/services/http-error.interceptor.ts
```

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/framework/services/http-error.interceptor.ts
```

3. Build the Application

```
$ ng build
```

4. Test Error Handling

Create a test that triggers an HTTP error:

```
// In any component

constructor(private api: ApiService) { // Request a non-existent endpoint
this.api.get('/api/non-existent-endpoint').subscribe({ next: (data) =>
console.log('Data:', data), error: (error) => { console.log('Caught error:',
error); // Error should be formatted as: // { code: 'NOT_FOUND', message: 'Resource
not found.', status: 404, ... } } }); }
```

5. Verify Retry Behavior

To test retry, you can temporarily modify your API to return 503:

```
// In browser DevTools Network tab:

// 1. Enable "Offline" mode briefly // 2. Watch for retry attempts in console logs //
3. See that after 2 retries, error is thrown
```

Common Problems

Symptom	Cause	Solution
Interceptor not running	Not registered in providers	Add HTTP_INTERCEPTORS provider to app.module.ts
`multi: true` missing	Single interceptor overwrites others	Always use `multi: true` with HTTP_INTERCEPTORS
Infinite retry loop	All errors trigger retry	Only retry specific status codes (429, 5xx)
Error format wrong	Custom error handler	Ensure components expect the formatted error object
Retry too fast	No exponential backoff	Consider adding delay with `timer()` in retry

---

Key Takeaways

- **Interceptors provide global error handling** — Every HTTP request flows through the interceptor
- **Transient errors deserve retries** — 5xx and 429 errors often succeed on retry
- **Consistent formatting simplifies handling** — Components receive the same error shape regardless of source

---

Acceptance Criteria

- [ ] `src/app/framework/services/http-error.interceptor.ts` exists
- [ ] Interceptor implements `HttpInterceptor` interface
- [ ] `intercept()` method handles requests and errors
- [ ] Retry logic attempts 2 retries for 429, 500, 502, 503, 504
- [ ] Error formatting produces consistent object shape
- [ ] Error codes map correctly (400 → BAD_REQUEST, etc.)
- [ ] User-friendly messages for each status code
- [ ] Client-side errors (ErrorEvent) handled separately
- [ ] Error logging includes request URL and method
- [ ] Interceptor registered with `HTTP_INTERCEPTORS` provider
- [ ] TypeScript compilation succeeds
- [ ] JSDoc comments document all functions

Next Step

Proceed to `314-global-error-handler.md` to create the global error handler for unhandled exceptions.

314: Global Error Handler

Status: Complete **Depends On:** 312-error-notification-service, 313-http-error-interceptor **Blocks:** Phase 8 (Framework Components)

Learning Objectives

After completing this section, you will:

- Understand how Angular's ErrorHandler interface works
- Know how to catch unhandled exceptions globally
- Recognize different error types (HTTP, chunk load, promise rejection)
- Be able to provide user-friendly messages for JavaScript errors

---

Objective

Create the `GlobalErrorHandler` that catches all unhandled exceptions in the Angular application and displays user-friendly error messages using the ErrorNotificationService.

Why

Errors escape individual component handlers for many reasons:

- Developers forget to add `.catch()` to promises
- Observable subscriptions don't include error callbacks
- Third-party libraries throw unexpectedly
- Template expressions throw during rendering

Without global error handling, these errors:

- Crash silently with cryptic console messages
- Leave users staring at broken UIs with no feedback
- Make debugging difficult without context

Angular's ErrorHandler

Angular provides an `ErrorHandler` class that catches all unhandled errors. By default, it just logs to console. We replace it with our `GlobalErrorHandler` that:

- **Categorizes errors** — HTTP vs chunk load vs promise rejection vs generic
- **Shows user feedback** — Toast notifications explain what went wrong
- **Logs context** — URL, route, stack trace for debugging

Error Types

The global handler categorizes errors by type:

Type	Detection	Example
HTTP Error	`instanceof HttpErrorResponse`	API returned 500
Chunk Load Error	Message contains "loading chunk"	Lazy module failed to load
Promise Rejection	Error has `.rejection` property	Unhandled async error
Generic Error	Default	TypeError, ReferenceError

Each type gets appropriate handling:

Type	Display	Duration
HTTP Error	Error toast	5 seconds
Chunk Load Error	Sticky toast (requires action)	Until dismissed
Promise Rejection	Error toast	7 seconds
Generic Error	Sticky toast	Until dismissed

The Lazy-Loading Trick

The GlobalErrorHandler uses Angular's `Injector` to get the `ErrorNotificationService` instead of constructor injection:

```
// Wrong - can cause circular dependency
constructor(private errorNotification: ErrorNotificationService) { }



// Right - lazy loading via injector constructor(private injector: Injector) { }



private get errorNotificationService(): ErrorNotificationService { return
this.injector.get(ErrorNotificationService); }
```

This avoids circular dependency issues because:

- ErrorHandler is created very early in Angular's bootstrap
- ErrorNotificationService may depend on services not yet created
- Lazy injection defers resolution until first use

Chunk Load Errors

A special case is **chunk load errors** — when lazy-loaded modules fail to load:

```
Error: Loading chunk 5 failed.
```

These typically happen when:

- Network connection drops during navigation
- Server deployment changed chunk hashes
- Browser cache serves stale chunk

The handler shows a sticky message prompting the user to refresh.

What

Step 314.1: Create the Global Error Handler

Create the file `src/app/framework/services/global-error.handler.ts`:

```
// src/app/framework/services/global-error.handler.ts

// VERSION 1 (Section 314) - Global error handler



import { ErrorHandler, Injectable, Injector } from '@angular/core'; import
{ HttpErrorResponse } from '@angular/common/http'; import { ErrorNotificationService }
from './error-notification.service'; import { ErrorCategory } from '../models/error-
notification.interface';



/**



  • Global error handler

 *

  • Catches all unhandled errors in the Angular application and displays



  • user-friendly error messages using the ErrorNotificationService.

 *

  • This handler:



  • - Intercepts all unhandled exceptions



  • - Categorizes errors by type (HTTP, Promise rejection, component error, etc.)



  • - Displays appropriate user-facing messages



  • - Logs detailed error information for debugging
```

```
*
 • Why Lazy Injection?
*
 • Uses Injector.get() instead of constructor injection to avoid circular
 • dependency issues. ErrorHandler is created very early in Angular's
 • bootstrap process, before some services are fully initialized.
*
 • @example
 •
```

typescript

- // In app.module.ts
- providers: [
- {
- provide: ErrorHandler,
- useClass: GlobalErrorHandler
- }
-]

```
 /
@Injectable() export class GlobalErrorHandler implements ErrorHandler { /**


  • ErrorNotificationService instance

   *

  • Lazy-loaded via getter to avoid circular dependency issues.


  • Service is requested from injector on first access.

   */
private get errorNotificationService(): ErrorNotificationService { return
this.injector.get(ErrorNotificationService); }


/**


  • Constructor with injector for lazy service loading

   *

  • @param injector - Angular injector for lazy-loading services

   */
constructor(private injector: Injector) {}


/**


  • Handle error
```

```
   *

  • Main error handling method called by Angular for all unhandled errors

   *

  • @param error - The error object

   */
handleError(error: any): void { // Extract actual error from Angular's wrapper const
actualError = this.unwrapError(error);



// Log to console for debugging this.logErrorToConsole(actualError);



// Handle different error types if (this.isHttpError(actualError))
{ this.handleHttpError(actualError); } else if (this.isChunkLoadError(actualError))
{ this.handleChunkLoadError(actualError); } else if (this.isPromiseRejection(error))
{ this.handlePromiseRejection(actualError); } else
{ this.handleGenericError(actualError); } }



/**



  • Unwrap error from Angular's ErrorEvent wrapper

   *

  • Angular sometimes wraps errors in objects with .rejection or .error



  • properties. This extracts the actual error object.

   *

  • @param error - Potentially wrapped error
```

```
  • @returns Actual error object

   */
private unwrapError(error: any): any { // Angular wraps promise rejections in
rejection field if (error && error.rejection) { return error.rejection; }



// Angular wraps errors in error field if (error && error.error) { return
error.error; }



return error; }



/**



  • Check if error is HTTP error

   *

  • @param error - Error object



  • @returns True if HTTP error

   */
private isHttpError(error: any): boolean { return error instanceof
HttpErrorResponse; }



/**



  • Check if error is chunk load error (lazy-loaded module)
```

```
   *
  • Chunk load errors occur when lazy-loaded JavaScript files fail to load.
  • This typically happens due to network issues or server deployments
  • that change chunk hashes.
   *
  • @param error - Error object
  • @returns True if chunk load error
   */
private isChunkLoadError(error: any): boolean { if (!(error instanceof Error))
{ return false; }

const message = error.message.toLowerCase(); return ( message.includes('loading
chunk') || message.includes('failed to fetch') || message.includes('dynamically
imported module') ); }

/**
  • Check if error is promise rejection
   *
  • @param error - Error object
```

```
   • @returns True if promise rejection

    */
private isPromiseRejection(error: any): boolean { return error && error.promise &&
error.rejection; }



/**



   • Handle HTTP errors

    *

   • HTTP errors are already handled by the HTTP interceptor, but may



   • bubble up if not caught in component subscriptions.

    *

   • @param error - HTTP error response

    */
private handleHttpError(error: HttpErrorResponse): void { // Check if error was
already formatted by interceptor if (error.error && error.error.code) { // Already
formatted by interceptor this.errorNotificationService.showHttpError(error.error); }
else { // Not formatted - format now
this.errorNotificationService.showHttpError({ code: 'HTTP_ERROR', message:
error.message || 'An HTTP error occurred', status: error.status, statusText:
error.statusText, url: error.url || undefined, timestamp: new
Date().toISOString() }); } }



/**



   • Handle chunk load errors (lazy-loaded modules)
```

```
   *
  • These occur when lazy-loaded JavaScript chunks fail to load,
  • typically due to network issues or deployments.
   *
  • Shows a sticky toast prompting the user to refresh.
   *
  • @param error - Chunk load error
   */
private handleChunkLoadError(error: Error): void
{ this.errorNotificationService.show( { category: ErrorCategory.NETWORK, severity:
'error', summary: 'Loading Error', detail: 'Failed to load application module. Please
refresh the page. ' + 'If the problem persists, clear your browser cache.', code:
'CHUNK_LOAD_ERROR', timestamp: new Date().toISOString(), originalError: error },
{ sticky: true, // Don't auto-hide - requires user action life: 0 } ); }

/**
  • Handle promise rejections
   *
  • Unhandled promise rejections that weren't caught with .catch()
   *
  • @param error - Rejected promise value
   */
```

```
private handlePromiseRejection(error: any): void { // If it's an HTTP error, handle as
HTTP if (this.isHttpError(error)) { this.handleHttpError(error); return; }



// Generic promise rejection this.errorNotificationService.show( { category:
ErrorCategory.APPLICATION, severity: 'error', summary: 'Operation Failed', detail:
error?.message || 'An asynchronous operation failed. Please try again.', code:
'PROMISE_REJECTION', timestamp: new Date().toISOString(), originalError: error },
{ life: 7000 // Longer display for async errors } ); }



/**



  • Handle generic errors


   *


  • All other unhandled errors (component errors, TypeError, etc.)


   *


  • Shows a sticky toast for serious programming errors.


   *


  • @param error - Error object


   */
private handleGenericError(error: any): void { const errorMessage = error?.message ||
error?.toString() || 'An unexpected error occurred';



// Check for known error patterns const isTypeError = error instanceof TypeError;
const isReferenceError = error instanceof ReferenceError;



let detail = errorMessage; if (isTypeError || isReferenceError) { detail += ' This is
likely a programming error. Please report this issue.'; }
```

```
this.errorNotificationService.show( { category: ErrorCategory.CLIENT, severity:
'error', summary: 'Application Error', detail, code: 'UNHANDLED_ERROR', timestamp: new
Date().toISOString(), originalError: error }, { sticky: true, // Keep visible for
debugging life: 0 } ); }



/**



  • Log error to console for debugging

   *

  • Groups error information for easier debugging in DevTools.

   *

  • @param error - Error object

   */
private logErrorToConsole(error: any): void { const timestamp = new
Date().toISOString();



console.group( [Global Error Handler] ${timestamp} ); console.error('Error caught
by global handler:', error);



// Log stack trace if available if (error instanceof Error && error.stack)
{ console.error('Stack trace:', error.stack); }



// Log additional context console.error('Error type:', error?.constructor?.name ||
typeof error);
```

```
// Log location information console.error('Location:', { url: window.location.href,
route: window.location.pathname });



console.groupEnd(); } }
```

Step 314.2: Update the Services Barrel File

Update `src/app/framework/services/index.ts`:

```
// src/app/framework/services/index.ts

// VERSION 13 (Section 314) - Added GlobalErrorHandler



export * from './url-state.service'; export * from './api.service'; export * from './
request-coordinator.service'; export * from './domain-config-registry.service'; export
* from './domain-config-validator.service'; export * from './popout-context.service';
export * from './popout-manager.service'; export * from './user-preferences.service';
export * from './filter-options.service'; export * from './picker-config-
registry.service'; export * from './resource-management.service'; export * from './
error-notification.service'; export * from './http-error.interceptor'; export * from
'./global-error.handler';
```

Step 314.3: Register the Global Error Handler

Update `src/app/app.module.ts` to register the handler:

```
// src/app/app.module.ts (partial - add to existing)

import { ErrorHandler } from '@angular/core'; import { GlobalErrorHandler } from './
framework/services';



@NgModule({ // ... providers: [ { provide: ErrorHandler, useClass:
GlobalErrorHandler } ], // ... }) export class AppModule { }
```

Verification

1. Check File Exists

```
$ ls -la src/app/framework/services/global-error.handler.ts
```

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/framework/services/global-error.handler.ts
```

3. Build the Application

```
$ ng build
```

4. Test Unhandled Error

Add temporary code to throw an error:

```
// In any component ngOnInit

ngOnInit(): void { // Uncomment to test - should show sticky toast // throw new
Error('Test unhandled error'); }
```

5. Test Promise Rejection

```
// In any component

ngOnInit(): void { // Uncomment to test - should show error toast //
Promise.reject(new Error('Test promise rejection')); }
```

6. Test Chunk Load Error

Simulate by modifying the error check temporarily, or:

- Build the application
- Delete a chunk file from dist
- Navigate to the route that loads that chunk

7. Verify Console Logging

When an error is caught, check the browser console for grouped log output:

```
▼ [Global Error Handler] 2024-02-09T12:00:00.000Z

Error caught by global handler: Error: Test error Stack trace: Error: Test error at
Component.ngOnInit (component.ts:15) ... Error type: Error Location: {url: "http://
localhost:4200/discover", route: "/discover"}
```

Common Problems

Symptom	Cause	Solution
Circular dependency error	ErrorHandler created too early	Use `Injector.get()` instead of constructor injection
No toast appears	ErrorNotificationService not provided	Ensure MessageService is in app.module.ts providers
Error handled twice	HTTP interceptor + global handler	Check if error already formatted (has `.code` property)
Console shows "NullInjectorError"	Service not available	Ensure all dependencies are provided
Chunk load error not detected	Message pattern changed	Update `isChunkLoadError()` detection patterns

---

Key Takeaways

- **Global error handling catches everything** — Even errors developers forgot to handle
- **Lazy injection avoids circular dependencies** — Use `Injector.get()` for early-loaded services
- **Error categorization enables appropriate responses** — Chunk errors need refresh, promise errors need retry

---

Acceptance Criteria

- [ ] `src/app/framework/services/global-error.handler.ts` exists
- [ ] Handler extends Angular's `ErrorHandler` interface
- [ ] `handleError()` method catches all unhandled errors
- [ ] HTTP errors detected and formatted
- [ ] Chunk load errors show sticky refresh prompt
- [ ] Promise rejections handled with 7-second display
- [ ] Generic errors show sticky toast
- [ ] Console logging includes stack trace and location
- [ ] Lazy injection via `Injector.get()` to avoid circular deps
- [ ] Handler registered with `provide: ErrorHandler` in app.module.ts

- [ ] TypeScript compilation succeeds
- [ ] JSDoc comments document all methods

Phase 3C Complete

Congratulations! You have completed Phase 3C: Error Handling.

What you built:

- ErrorNotification interface — Error categorization and display types
- ErrorNotificationService — Centralized toast notifications with deduplication
- HttpErrorInterceptor — Automatic retry and consistent error formatting
- GlobalErrorHandler — Catch-all for unhandled exceptions

The Aha Moment: "Errors are inevitable. User-friendly error messages are not."

Next Step

Proceed to `315-popout-token.md` to create the injection token for pop-out window detection.

315: Pop-Out Token

Status: Complete **Depends On:** 307-popout-context-service **Blocks:** Phase 8 (Framework Components)

Learning Objectives

After completing this section, you will:

- Understand how Angular injection tokens work
- Know when to use tokens vs services for dependency injection
- Recognize the benefits of compile-time constants for environment detection
- Be able to create and provide injection tokens

Objective

Create the `IS_POPOUT_TOKEN` injection token that provides a boolean indicating whether the current window is a pop-out window. This token is used throughout the application to conditionally render components and enable/disable features based on pop-out context.

Why

Pop-out windows need to behave differently from the main window:

Behavior	Main Window	Pop-Out Window
Navigation header	Shown	Hidden
Pop-out buttons	Enabled	Disabled (already popped out)
Close button	Hidden	Shown
State sync	Source	Receiver

Components need to know "Am I in a pop-out?" to adjust their behavior.

Option 1: Call a Service Method

```
// In every component
constructor(private popoutContext: PopOutContextService) {}


get isPopOut(): boolean { return this.popoutContext.isInPopOut(); }
```

Problem: Every component needs to inject the service and call the method.

Option 2: Use an Injection Token (Our Approach)

```
// In any component
constructor(@Inject(IS_POPOUT_TOKEN) public isPopOut: boolean) {}
```

Benefits:

- **Simpler injection** — Just inject the boolean directly
- **Compile-time constant** — Value determined once at bootstrap
- **Template-friendly** — Use directly in templates: `*ngIf="!isPopOut"`
- **Testable** — Easy to provide different values in tests

How It Works

- **At bootstrap**, the router URL is checked for `/popout/` prefix
- **A factory function** returns `true` or `false`
- **The token is provided** at root level with this value
- **Components inject** the token to get the boolean

```
// Factory determines value once

export function isPopOutFactory(router: Router): boolean { return
router.url.includes('/popout/'); }



// Provided at root { provide: IS_POPOUT_TOKEN, useFactory: isPopOutFactory, deps:
[Router] }



// Injected in component constructor(@Inject(IS_POPOUT_TOKEN) private isPopOut:
boolean) { if (this.isPopOut) { // Pop-out specific behavior } }
```

Injection Token vs Service

Feature	Injection Token	Service
Value type	Primitive (boolean, string, etc.)	Object with methods
Computed	Once at creation	Can change over time
Dependencies	Via factory function	Via constructor
Use case	Static configuration	Dynamic behavior

For "is this a pop-out?", an injection token is ideal because:

- The answer never changes during the window's lifetime
- It's a simple boolean, not a complex object
- Components just need to read it, not call methods

What

Step 315.1: Create the Tokens Directory

Create the tokens directory and barrel file.

Create `src/app/framework/tokens/index.ts`:

```
// src/app/framework/tokens/index.ts
// VERSION 1 (Section 315) - Framework injection tokens


export * from './popout.token';
```

Step 315.2: Create the Pop-Out Token

Create the file `src/app/framework/tokens/popout.token.ts`:

```
// src/app/framework/tokens/popout.token.ts
// VERSION 1 (Section 315) - Pop-out window detection token


import { InjectionToken } from '@angular/core';


/**


  • Injection token for pop-out window detection

 *

  • Provides a boolean indicating whether the current window is a pop-out window.


  • This value is determined once at bootstrap based on the URL and never changes.

 *

  • Usage:

 *

  •
```

typescript

- constructor(@Inject(IS_POPOUT_TOKEN) public isPopOut: boolean) {}

```
  • In Templates:

*

  •
```

html <*nav* ngIf="!isPopOut">Main navigation</nav>

<*button* ngIf="isPopOut" (click)="close()">Close</button>

```
  • Providing the Token:

*

  •
```

typescript

- // In app.module.ts
- {
- provide: IS_POPOUT_TOKEN,
- useFactory: () => window.location.pathname.includes('/popout/')
- }

```
 /
export const IS_POPOUT_TOKEN = new InjectionToken<boolean>('IS_POPOUT_TOKEN');
```

Step 315.3: Create the Factory Function

Add a factory function to provide the token value. Update `src/app/framework/tokens/popout.token.ts`:

```
// src/app/framework/tokens/popout.token.ts
// VERSION 2 (Section 315) - Added factory function


import { InjectionToken } from '@angular/core';


/**


  • Injection token for pop-out window detection

 *

  • Provides a boolean indicating whether the current window is a pop-out window.


  • This value is determined once at bootstrap based on the URL and never changes.

 *

  • Usage:

 *

  •
```

typescript

• constructor(@Inject(IS_POPOUT_TOKEN) public isPopOut: boolean) {}

```
  • In Templates:

 *

  •
```

html <*nav* ngIf="!isPopOut">Main navigation</nav>

<*button* ngIf="isPopOut" (click)="close()">Close</button>

```
 / export const IS_POPOUT_TOKEN = new InjectionToken<boolean>('IS_POPOUT_TOKEN');


/**

  • Factory function to determine if current window is a pop-out

 *

  • Checks the current URL pathname for the /popout/ prefix.


  • This is called once during application bootstrap.

 *

  • @returns True if the current window is a pop-out window

 *

  • @example


  •
```

typescript

- // In app.module.ts
- {
- provide: IS_POPOUT_TOKEN,
- useFactory: isPopOutFactory
- }

```
 /
export function isPopOutFactory(): boolean { // Check URL for popout prefix // Pop-out
routes follow pattern: /popout/:panelId/:viewType return
window.location.pathname.includes('/popout/'); }



/**



  • Provider configuration for IS_POPOUT_TOKEN

 *

  • Use this in your app.module.ts providers array.

 *

  • @example



  •
```

typescript

- import { IS_POPOUT_PROVIDER } from './framework/tokens';

*

- @NgModule({
- providers: [IS_POPOUT_PROVIDER]
- })
- export class AppModule { }

```
 /
export const IS_POPOUT_PROVIDER = { provide: IS_POPOUT_TOKEN, useFactory:
isPopOutFactory };
```

Step 315.4: Update the Barrel File

Update `src/app/framework/tokens/index.ts`:

```
// src/app/framework/tokens/index.ts
// VERSION 1 (Section 315) - Framework injection tokens


export * from './popout.token';
```

Step 315.5: Register the Provider

Update `src/app/app.module.ts` to provide the token:

```
// src/app/app.module.ts (partial - add to existing)

import { IS_POPOUT_PROVIDER } from './framework/tokens';



@NgModule({ // ... providers: [ IS_POPOUT_PROVIDER // ... other providers ], // ... })
export class AppModule { }
```

Verification

1. Check Files Exist

```
$ ls -la src/app/framework/tokens/
```

Expected output:

```
total 12

drwxr-xr-x 2 user user 4096 Feb  9 12:00 . drwxr-xr-x 5 user user 4096 Feb  9 12:00 ..
-rw-r--r-- 1 user user  123 Feb  9 12:00 index.ts -rw-r--r-- 1 user user 1456 Feb  9
12:00 popout.token.ts
```

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/framework/tokens/popout.token.ts
```

3. Build the Application

```
$ ng build
```

4. Test Token Injection

Add temporary test code:

```
// In any component

import { Inject } from '@angular/core'; import { IS_POPOUT_TOKEN } from './framework/
tokens';



@Component({...}) export class TestComponent { constructor(@Inject(IS_POPOUT_TOKEN)
private isPopOut: boolean) { console.log('Is pop-out window:', this.isPopOut); } }
```

5. Verify Pop-Out Detection

- Navigate to `http://localhost:4200/discover` → Console shows `false`
- Navigate to `http://localhost:4200/popout/panel-1/chart` → Console shows `true`

Common Problems

Symptom	Cause	Solution
`NullInjectorError: No provider for IS_POPOUT_TOKEN`	Token not provided	Add `IS_POPOUT_PROVIDER` to app.module.ts
Token always `false`	URL checked before navigation	Factory uses `window.location.pathname`, not Router
Token value changes	Token recreated per component	Ensure `providedIn: 'root'` behavior via app.module.ts
TypeScript error on `@Inject`	Missing import	Import `Inject` from `@angular/core`

Key Takeaways

- **Injection tokens provide primitive values** — Use when you need a simple value, not a service
- **Factory functions compute values once** — The pop-out check runs at bootstrap only
- **Tokens simplify component code** — No service method calls, just inject the boolean

Acceptance Criteria

- [ ] `src/app/framework/tokens/popout.token.ts` exists
- [ ] `IS_POPOUT_TOKEN` defined as `InjectionToken<boolean>`
- [ ] `isPopOutFactory()` function returns boolean based on URL
- [ ] `IS_POPOUT_PROVIDER` object ready for app.module.ts
- [ ] Barrel file exports all token-related items
- [ ] Token correctly detects `/popout/` in URL pathname
- [ ] TypeScript compilation succeeds
- [ ] JSDoc comments document usage patterns

Phase 3 Complete

Congratulations! You have completed Phase 3: Framework Services.

What you built:

Phase 3A - Core Services:

- UrlStateService — URL-First state management
- ApiService — HTTP request wrapper
- RequestCoordinatorService — Cache, deduplication, retry
- DomainConfigRegistry — Domain configuration storage
- DomainConfigValidator — Runtime configuration validation
- ResourceManagementService — Core state orchestrator

Phase 3B - Popout & Specialized Services:

- PopOutContextService — Pop-out window detection
- PopOutManagerService — Pop-out window management
- UserPreferencesService — localStorage preferences
- FilterOptionsService — Filter dropdown caching
- PickerConfigRegistry — Picker configuration lookup

Phase 3C - Error Handling:

- ErrorNotificationService — Toast notifications with deduplication

- HttpErrorInterceptor — Automatic retry and error formatting
- GlobalErrorHandler — Catch-all for unhandled exceptions
- IS_POPOUT_TOKEN — Pop-out window detection token

The Aha Moment: "Services are the nervous system of the application. They coordinate state, handle errors, and enable features to work together seamlessly."

Next Step

Proceed to `401-base-model-interface.md` to begin Phase 4: Domain Models.

401: Base Model Interface

Status: Complete **Depends On:** 201-domain-config-interface **Blocks:** 402-domain-data-models, 403-domain-filter-statistics-models

---

Learning Objectives

After completing this section, you will:

- Understand why domain models use classes instead of interfaces
- Know how to implement the class-with-partial-constructor pattern
- Recognize the benefits of `fromApiResponse()` static factory methods
- Be able to create domain-agnostic base patterns for data models

---

Objective

Establish the base patterns for domain data models that all specific domain models will follow. This section defines the conventions for creating type-safe, API-aware data classes.

Why

Domain models represent the data your application works with. In Vvroom, the primary domain is **automobiles** — vehicles, manufacturers, models, body classes, and VIN instances.

The Interface vs Class Decision

TypeScript offers two ways to define data shapes:

Interfaces:

```
interface Vehicle {
vehicle_id: string; manufacturer: string; model: string; year: number; }
```

Classes:

```
class Vehicle {
vehicle_id!: string; manufacturer!: string; model!: string; year!: number; }
```

At first glance, interfaces seem simpler. But classes provide crucial advantages:

Feature	Interface	Class
Runtime existence	No (erased)	Yes
Methods	No	Yes
Computed properties	No	Yes (getters)
Factory methods	No	Yes (static)
Type guard with `instanceof`	No	Yes
Partial initialization	Manual	Constructor pattern
API transformation	External function	`fromApiResponse()`

For domain models that need methods, transformation logic, and computed properties, classes are the better choice.

The Partial Constructor Pattern

Domain models often come from APIs with varied data. The partial constructor pattern handles this:

```
class Vehicle {

vehicle_id!: string; manufacturer!: string; model!: string; year!: number;



constructor(partial?: Partial<Vehicle>) { if (partial) { Object.assign(this,
partial); } } }
```

Benefits:

- **Optional initialization** — Create empty instances or full ones
- **Flexible merging** — `new Vehicle({ ...existing, year: 2024 })`
- **Type safety** — `Partial<Vehicle>` ensures only valid properties
- **IDE support** — Autocomplete works for constructor params

The Static Factory Pattern

APIs often return data with different field naming conventions:

```
{

"vehicle_id": "TOY-CAM-2024", "manufacturer": "Toyota", "model_year": 2024,
"body_class": "Sedan" }
```

The `fromApiResponse()` static method handles transformation:

```
class Vehicle {

// ...



static fromApiResponse(data: any): Vehicle { return new Vehicle({ vehicle_id:
data.vehicle_id || data.id, manufacturer: data.manufacturer, model: data.model, year:
Number(data.model_year || data.year), body_class: data.body_class ||
data.bodyClass }); } }
```

Benefits:

- **Single transformation point** — All API normalization in one place
- **Handles both formats** — `model_year` and `year` both work
- **Type coercion** — `Number()` ensures numeric types
- **Testable** — Easy to unit test transformation logic

Instance Methods for Computed Values

Classes can include methods for computed values:

```
class Vehicle {
// ...


getDisplayName(): string { return ${this.manufacturer} ${this.model} ${this.year}; }


getAge(): number { return new Date().getFullYear() - this.year; }


isCurrentYear(): boolean { return this.year === new Date().getFullYear(); } }
```

This encapsulates business logic in the model itself, making it reusable across components.

What

Step 401.1: Establish Model Conventions

Before creating specific models, establish the patterns all domain models will follow.

Convention 1: Class with Non-Null Assertion

Use `!` for required properties to indicate they will be set:

```
class Model {

required_field!: string;  // Will be set by constructor optional_field?: number;  //
May or may not be set }
```

Convention 2: Partial Constructor

Every model class has a partial constructor:

```
constructor(partial?: Partial<ModelClass>) {

if (partial) { Object.assign(this, partial); } }
```

Convention 3: Static Factory Method

Models that come from APIs have `fromApiResponse()`:

```
static fromApiResponse(data: any): ModelClass {

return new ModelClass({ // Map API fields to class properties }); }
```

Convention 4: Display Methods

Models with display requirements have getter methods:

```
getDisplayName(): string { ... }

getFullDescription(): string { ... }
```

Convention 5: JSDoc Documentation

Every property and method has JSDoc:

```
/**

  • Vehicle manufacturer name

 *

  • @example 'Toyota', 'Honda', 'Ford'

 */
manufacturer!: string;
```

Step 401.2: Create the Domain Models Directory

Create the directory structure for automobile domain models.

```
$ mkdir -p src/app/domains/automobile/models
$ touch src/app/domains/automobile/models/index.ts
```

Create the barrel file `src/app/domains/automobile/models/index.ts`:

```
// src/app/domains/automobile/models/index.ts
// VERSION 1 (Section 401) - Automobile domain models barrel


// Data models (Section 402) // export * from './automobile.data';


// Filter and statistics models (Section 403) // export * from './
automobile.filters'; // export * from './automobile.statistics';
```

Step 401.3: Document the Model Pattern

Create a reference document showing the complete model pattern.

The complete model pattern looks like this:

```
/**
  • [Model Name]
 *
  • [Description of what this model represents]
 *
  • Domain: [Domain Name]
 *
  • @example

  •
```

typescript

- const instance: ModelClass = {
- field1: 'value1',
- field2: 123
- };

```
 /
export class ModelClass { /**


  • [Field description]

   *

  • @example 'example value'

   */
field1!: string;


/**


  • [Optional field description]

   *

  • @example 123

   */
field2?: number;


/**


  • Constructor with partial data

   *

  • @param partial - Partial ModelClass object
```

```
   */
constructor(partial?: Partial<ModelClass>) { if (partial) { Object.assign(this,
partial); } }



/**



  • Create ModelClass from API response


   *


  • @param data - Raw API response data



  • @returns ModelClass instance


   */
static fromApiResponse(data: any): ModelClass { return new ModelClass({ field1:
data.field1 || data.field_1, field2: data.field2 !== undefined ? Number(data.field2) :
undefined }); }



/**



  • Get display name


   *


  • @returns Formatted display string


   */
getDisplayName(): string { return this.field1; } }
```

Verification

1. Check Directory Exists

```
$ ls -la src/app/domains/automobile/models/
```

Expected output:

```
total 8
drwxr-xr-x 2 user user 4096 Feb  9 12:00 . drwxr-xr-x 3 user user 4096 Feb  9 12:00 ..
-rw-r--r-- 1 user user  200 Feb  9 12:00 index.ts
```

2. TypeScript Check

```
$ npx tsc --noEmit src/app/domains/automobile/models/index.ts
```

Common Problems

Symptom	Cause	Solution
`Property has no initializer`	Missing `!` on required field	Add `!` or make optional with `?`
`Partial<T>` not accepting field	Wrong field name	Ensure field names match exactly
API fields not mapping	Missing transformation	Add case in `fromApiResponse()`
Methods not available	Using interface instead of class	Convert to class with methods

---

Key Takeaways

- **Classes over interfaces for domain models** — Enable methods, factories, and instanceof checks
- **Partial constructor pattern** — Flexible initialization with type safety
- **Static factory for API data** — Single point for API field normalization
- **Instance methods for computed values** — Encapsulate business logic in the model

---

Acceptance Criteria

- [ ] Domain models directory structure created
- [ ] `src/app/domains/automobile/models/index.ts` exists
- [ ] Model conventions documented and understood:
- [ ] Non-null assertion for required fields
- [ ] Optional marker for optional fields
- [ ] Partial constructor pattern
- [ ] Static `fromApiResponse()` factory
- [ ] Instance methods for computed values
- [ ] JSDoc documentation on all members

Next Step

Proceed to `402-domain-data-models.md` to create the vehicle and VIN instance data models.

402: Domain Data Models

Status: Complete **Depends On:** 401-base-model-interface **Blocks:** 403-domain-filter-statistics-models, Phase 5 (Domain Adapters)

Learning Objectives

After completing this section, you will:

- Understand how to model domain entities as TypeScript classes
- Know how to handle API response transformation for different naming conventions
- Recognize the relationship between aggregate data (VehicleResult) and detail data (VinInstance)
- Be able to implement utility methods that encapsulate business logic

---

Objective

Create the core data models for the automobile domain: `VehicleResult` for vehicle configurations and `VinInstance` for individual VIN records.

Why

The automobile domain centers on two core entities:

VehicleResult — The Aggregate

A `VehicleResult` represents a **unique vehicle configuration** — a specific combination of:

- Manufacturer (Toyota, Honda, Ford)
- Model (Camry, Accord, F-150)

- Year (2024, 2023, 2022)
- Body Class (Sedan, SUV, Truck)

Each configuration has an `instance_count` showing how many individual vehicles (VINs) exist for that configuration.

```
Toyota Camry 2024 Sedan — 156 instances

Honda Accord 2023 Sedan — 89 instances Ford F-150 2024 Truck — 234 instances
```

VinInstance — The Detail

A `VinInstance` represents a **single vehicle** with a unique VIN (Vehicle Identification Number). When users expand a row in the data table, they see individual VINs for that vehicle configuration.

```
1HGCM82633A123456 — Toyota Camry 2024

1HGCM82633A123457 — Toyota Camry 2024 1HGCM82633A123458 — Toyota Camry 2024
```

The Relationship

```
VehicleResult (1) ———— (many) VinInstance

↓ manufacturer: Toyota model: Camry year: 2024 body_class: Sedan instance_count: 156 ↓
Links to 156 VinInstance records
```

API Response Handling

The API may return data in different formats:

```
// snake_case from backend

{ "vehicle_id": "TOY-CAM-2024-SED", "body_class": "Sedan", "instance_count": 156,
"first_seen": "2024-01-15T10:30:00Z" }



// camelCase from some endpoints { "vehicleId": "TOY-CAM-2024-SED", "bodyClass":
"Sedan", "instanceCount": 156, "firstSeen": "2024-01-15T10:30:00Z" }
```

The `fromApiResponse()` method handles both:

```
static fromApiResponse(data: any): VehicleResult {

return new VehicleResult({ vehicle_id: data.vehicle_id || data.vehicleId || data.id,
body_class: data.body_class || data.bodyClass, instance_count:
Number(data.instance_count || data.instanceCount || 0), first_seen: data.first_seen ||
data.firstSeen }); }
```

What

Step 402.1: Create the Vehicle Result Model

Create the file `src/app/domains/automobile/models/automobile.data.ts`:

```
// src/app/domains/automobile/models/automobile.data.ts
// VERSION 1 (Section 402) - Automobile domain data models


/**


  • Vehicle result data


 *


  • Represents a unique vehicle configuration with aggregated VIN instance count.


  • Each record represents a distinct combination of manufacturer, model, year, and
    body class.


 *


  • Domain: Automobile Discovery


 *


  • @example


  •
```

typescript

```
• const vehicle: VehicleResult = {
• vehicle_id: 'TOY-CAM-2024-SED',
• manufacturer: 'Toyota',
• model: 'Camry',
• year: 2024,
• body_class: 'Sedan',
• instance_count: 156,
• first_seen: '2024-01-15T10:30:00Z',
• last_seen: '2024-11-20T14:22:00Z'
• };
```

```
 /
export class VehicleResult { /**


  • Unique vehicle identifier


  • Composite key: manufacturer-model-year-bodyclass

   *

  • @example 'TOY-CAM-2024-SED', 'HON-ACC-2023-SED'

   */
vehicle_id!: string;


/**


  • Vehicle manufacturer name

   *

  • @example 'Toyota', 'Honda', 'Ford', 'Chevrolet'

   */
manufacturer!: string;


/**


  • Vehicle model name
```

```
   *
  • @example 'Camry', 'Accord', 'F-150', 'Silverado'
   */
model!: string;

/**
  • Vehicle model year
   *
  • @example 2024, 2023, 2022
   */
year!: number;

/**
  • Vehicle body class/type
   *
  • @example 'Sedan', 'SUV', 'Truck', 'Coupe', 'Wagon', 'Van'
   */
body_class!: string;

/**
```

```
  • Number of VIN instances for this vehicle configuration


  • Represents how many individual vehicles (VINs) exist for this configuration

   *

  • @example 156 (means 156 unique VINs for this vehicle config)

   */
instance_count!: number;


/**


  • Date/time this vehicle configuration was first seen in the system


  • ISO 8601 format

   *

  • @example '2024-01-15T10:30:00Z'

   */
first_seen?: string;


/**


  • Date/time this vehicle configuration was last updated
```

```
   • ISO 8601 format
    *
   • @example '2024-11-20T14:22:00Z'
    */
last_seen?: string;

/**
   • Drive type
   • @example 'FWD', 'RWD', 'AWD', '4WD'
    */
drive_type?: string;

/**
   • Engine configuration
   • @example 'V6', 'I4', 'V8', 'Electric'
    */
engine?: string;

/**
```

```
   • Transmission type


   • @example 'Automatic', 'Manual', 'CVT'

    */
transmission?: string;


/**

   • Fuel type


   • @example 'Gasoline', 'Diesel', 'Electric', 'Hybrid'

    */
fuel_type?: string;


/**

   • Vehicle class/category


   • @example 'Passenger Car', 'Light Truck', 'Commercial Vehicle'

    */
vehicle_class?: string;


/**
```

```
  • Constructor with partial data

   *

  • @param partial - Partial VehicleResult object

   */
constructor(partial?: Partial<VehicleResult>) { if (partial) { Object.assign(this,
partial); } }



/**

  • Create VehicleResult from API response

   *

  • Handles both snake_case and camelCase field names from API.

   *

  • @param data - Raw API response data

  • @returns VehicleResult instance

   */
static fromApiResponse(data: any): VehicleResult { return new
VehicleResult({ vehicle_id: data.vehicle_id || data.vehicleId || data.id,
manufacturer: data.manufacturer, model: data.model, year: Number(data.year),
body_class: data.body_class || data.bodyClass, instance_count:
Number(data.instance_count || data.instanceCount || 0), first_seen: data.first_seen ||
data.firstSeen, last_seen: data.last_seen || data.lastSeen, drive_type:
data.drive_type || data.driveType, engine: data.engine, transmission:
data.transmission, fuel_type: data.fuel_type || data.fuelType, vehicle_class:
data.vehicle_class || data.vehicleClass }); }
```

```
/**


  • Get display name for vehicle

   *

  • @returns Formatted display string


  • @example 'Toyota Camry 2024'

   */
getDisplayName(): string { return ${this.manufacturer} ${this.model} $
{this.year}; }


/**


  • Get full description for vehicle

   *

  • @returns Detailed description string


  • @example 'Toyota Camry 2024 Sedan (156 instances)'

   */
getFullDescription(): string { return ${this.manufacturer} ${this.model} $
{this.year} ${this.body_class} (${this.instance_count} instances); }


/**
```

```
   • Check if vehicle has VIN instances
    *
   • @returns True if instance_count > 0
    */
hasInstances(): boolean { return this.instance_count > 0; }

/**
   • Get age of vehicle in years
    *
   • @returns Age in years from current year
    */
getAge(): number { const currentYear = new Date().getFullYear(); return currentYear -
this.year; }

/**
   • Check if vehicle is current year
    *
   • @returns True if year matches current year
    */
isCurrentYear(): boolean { return this.year === new Date().getFullYear(); } }
```

```
/**

  • VIN instance detail

 *

  • Represents a single VIN instance for a vehicle configuration.

  • Used in row expansion to show individual VINs.

 *

  • @example

  •
```

typescript

- const vin: VinInstance = {
- vin: '1HGCM82633A123456',
- vehicle_id: 'HON-ACC-2023-SED',
- registration_date: '2023-05-10',
- registration_state: 'CA',
- odometer_reading: 15234
- };

```
 /
export class VinInstance { /**


  • Vehicle Identification Number (17 characters)

   *

  • @example '1HGCM82633A123456'

   */
vin!: string;


/**


  • Associated vehicle configuration ID


  • Links to VehicleResult.vehicle_id

   *

  • @example 'HON-ACC-2023-SED'

   */
vehicle_id!: string;


/**


  • Registration date
```

```
   * @example '2023-05-10'
   */
  registration_date?: string;

  /**
   * Registration state/province
   * @example 'CA', 'TX', 'NY'
   */
  registration_state?: string;

  /**
   * Odometer reading (miles)
   * @example 15234
   */
  odometer_reading?: number;

  /**
   * Vehicle status
```

```
  • @example 'Active', 'Salvage', 'Totaled', 'Stolen'

   */
status?: string;


/**

  • Color


  • @example 'White', 'Black', 'Silver'

   */
color?: string;


/**

  • Current owner/registrant (anonymized)


  • @example 'Owner-12345'

   */
owner_id?: string;


/**

  • Last update timestamp
```

```
   • @example '2024-11-20T10:30:00Z'

    */
last_updated?: string;



/**



   • Constructor with partial data

    *

   • @param partial - Partial VinInstance object

    */
constructor(partial?: Partial<VinInstance>) { if (partial) { Object.assign(this,
partial); } }



/**



   • Create VinInstance from API response

    *

   • @param data - Raw API response data



   • @returns VinInstance instance

    */
static fromApiResponse(data: any): VinInstance { return new VinInstance({ vin:
data.vin, vehicle_id: data.vehicle_id || data.vehicleId, registration_date:
data.registration_date || data.registrationDate, registration_state:
data.registration_state || data.registrationState, odometer_reading:
```

```
data.odometer_reading || data.odometerReading, status: data.status, color: data.color,
owner_id: data.owner_id || data.ownerId, last_updated: data.last_updated ||
data.lastUpdated }); }



/**



  • Get formatted VIN (groups of 4 characters)

   *

  • @returns Formatted VIN string



  • @example '1HGC M826 33A1 2345 6'

   */
getFormattedVin(): string { return this.vin.match(/.{1,4}/g)?.join(' ') || this.vin; }



/**



  • Check if VIN is valid length

   *

  • @returns True if VIN is 17 characters

   */
isValidLength(): boolean { return this.vin?.length === 17; } }
```

Step 402.2: Update the Barrel File

Update `src/app/domains/automobile/models/index.ts`:

```
// src/app/domains/automobile/models/index.ts

// VERSION 2 (Section 402) - Added data models



export * from './automobile.data';



// Filter and statistics models (Section 403) // export * from './
automobile.filters'; // export * from './automobile.statistics';
```

Verification

1. Check File Exists

```
$ ls -la src/app/domains/automobile/models/automobile.data.ts
```

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/domains/automobile/models/automobile.data.ts
```

3. Test Model Usage

Create a temporary test:

```
// In any component or test file

import { VehicleResult, VinInstance } from '@app/domains/automobile/models';



// Test VehicleResult const vehicle = VehicleResult.fromApiResponse({ vehicle_id:
'TOY-CAM-2024-SED', manufacturer: 'Toyota', model: 'Camry', year: '2024', // String
from API body_class: 'Sedan', instance_count: '156' // String from API });



console.log('Display name:', vehicle.getDisplayName()); // Output: 'Toyota Camry 2024'



console.log('Full description:', vehicle.getFullDescription()); // Output: 'Toyota
Camry 2024 Sedan (156 instances)'



console.log('Age:', vehicle.getAge()); // Output: 0 (if current year is 2024)



// Test VinInstance const vin = new VinInstance({ vin: '1HGCM82633A123456',
vehicle_id: 'HON-ACC-2023-SED' });



console.log('Formatted VIN:', vin.getFormattedVin()); // Output: '1HGC M826 33A1 2345
6'



console.log('Valid length:', vin.isValidLength()); // Output: true
```

Common Problems

Symptom	Cause	Solution
`year` is string not number	API returns string	Use `Number()` in `fromApiResponse()`
`instance_count` is `NaN`	Missing or null from API	Default to 0: `Number(data.instance_count`
`body_class` undefined	Different API field name	Check for both `body_class` and `bodyClass`
Methods not available on object	Plain object, not class instance	Use `fromApiResponse()` or `new VehicleResult()`

0)

Key Takeaways

- **Classes enable methods and transformation** — `getDisplayName()`, `getAge()`, etc.
- **Handle both API naming conventions** — Check for `body_class` and `bodyClass`
- **Type coercion is essential** — API sends strings, models need numbers
- **Instance methods encapsulate logic** — Business rules live in the model

---

Acceptance Criteria

- [ ] `src/app/domains/automobile/models/automobile.data.ts` exists
- [ ] `VehicleResult` class with all required fields
- [ ] `VinInstance` class with all required fields
- [ ] Both classes have partial constructor
- [ ] Both classes have `fromApiResponse()` static method
- [ ] `VehicleResult` has utility methods:

- [ ] `getDisplayName()`
- [ ] `getFullDescription()`
- [ ] `hasInstances()`
- [ ] `getAge()`
- [ ] `isCurrentYear()`
- [ ] `VinInstance` has utility methods:
- [ ] `getFormattedVin()`
- [ ] `isValidLength()`
- [ ] Barrel file exports both classes
- [ ] TypeScript compilation succeeds
- [ ] JSDoc comments on all properties and methods

Next Step

Proceed to `403-domain-filter-statistics-models.md` to create the filter and statistics models.

403: Domain Filter and Statistics Models

Status: Complete **Depends On:** 401-base-model-interface, 402-domain-data-models **Blocks:** Phase 5 (Domain Adapters), Phase 6 (Charts)

Learning Objectives

After completing this section, you will:

- Understand the difference between filter models and data models
- Know how to model search filters with pagination and sorting
- Recognize the structure of aggregated statistics for chart data
- Be able to implement highlight filters for segmented chart data

---

Objective

Create the filter model (`AutoSearchFilters`) for search parameters and the statistics models (`VehicleStatistics`, `ManufacturerStat`, etc.) for chart and analytics data.

Why

Filter Models

When users search for vehicles, they apply filters:

- Manufacturer (Toyota)
- Year range (2020-2024)
- Body class (Sedan, SUV)

- Page number (1)
- Sort field (manufacturer)

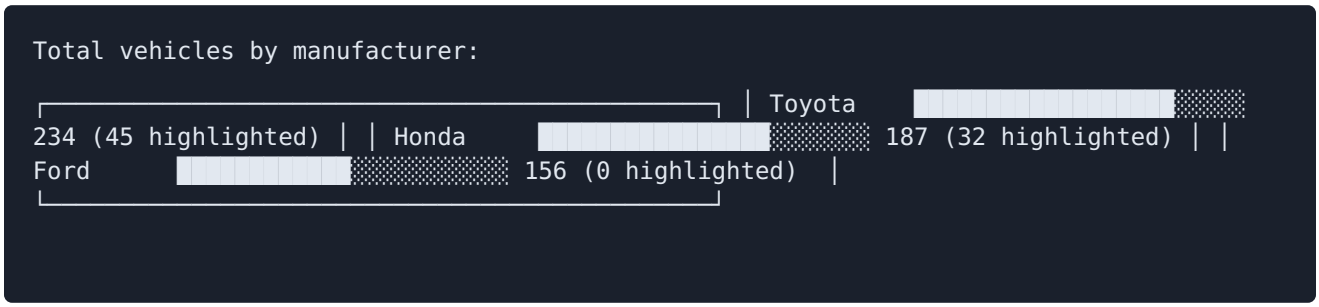
These filters:

- **Map to URL parameters** — Filters persist in the URL
- **Map to API query strings** — Sent to backend for filtering
- **Support multiple data types** — Strings, numbers, arrays

The `AutoSearchFilters` class models all filter options with utility methods for checking if filters are active.

Highlight Filters

Vvroom supports **chart highlighting** — showing a subset of data emphasized within the total:



Highlight filters are sent to the API with `h_` prefix:

- `?h_manufacturer=Toyota` — Highlight Toyota vehicles
- `?h_yearMin=2020&h_yearMax=2024` — Highlight 2020-2024 vehicles

The API returns segmented data: `{ total: 234, highlighted: 45 }`

Statistics Models

Statistics provide aggregated data for:

- **Summary cards** — Total vehicles, total VINs, manufacturer count
- **Charts** — Top manufacturers, body class distribution, year distribution
- **Filters** — Year range for slider min/max

Statistics models handle two API response formats:

Format 1: Array-based (simpler endpoints)

```
{
"total_vehicles": 1247, "top_manufacturers": [ { "name": "Toyota", "count": 234 } ] }
```

Format 2: Segmented (chart-ready)

```
{
"byManufacturer": { "Toyota": { "total": 234, "highlighted": 45 }, "Honda": { "total":
187, "highlighted": 32 } } }
```

The `VehicleStatistics.fromApiResponse()` method detects and handles both formats.

What

Step 403.1: Create the Filter Models

Create the file `src/app/domains/automobile/models/automobile.filters.ts`:

```
// src/app/domains/automobile/models/automobile.filters.ts
// VERSION 1 (Section 403) - Automobile domain filter models


/**


  • Highlight Filters

 *

  • Parameters for segmented statistics computation.

  Sent to backend API as h_ query parameters.
*


  • Purpose: Request segmented statistics with {total, highlighted} format


  • for chart visualization showing highlighted vs total data.

 *

  • @example


  •
```

typescript

- const highlights: HighlightFilters = {
- manufacturer: 'Ford',
- yearMin: 2020,
- yearMax: 2024
- };
- // URL: ?h_manufacturer=Ford&h_yearMin=2020&h_yearMax=2024

```
 /
export interface HighlightFilters { /**


  • Year range highlighting (minimum)


  • URL parameter: h_yearMin

   */
yearMin?: number;


/**


  • Year range highlighting (maximum)


  • URL parameter: h_yearMax

   */
yearMax?: number;


/**


  • Manufacturer highlighting


  • URL parameter: h_manufacturer

   */
manufacturer?: string;
```

```
/**

  • Model combinations highlighting

  • Format: Manufacturer:Model,Manufacturer:Model

  • URL parameter: h_modelCombos

   */
modelCombos?: string;

/**

  • Body class highlighting

  • URL parameter: h_bodyClass

   */
bodyClass?: string; }

/**

  • Automobile search filters

 *

  • Comprehensive filter model for searching and filtering vehicle data.
```

```
  • All fields are optional to support partial filtering.

*

  • @example


  •
```

typescript

• const filters: AutoSearchFilters = {

• manufacturer: 'Toyota',

• yearMin: 2020,

• yearMax: 2024,

• bodyClass: 'SUV',

• page: 1,

• size: 20,

• sort: 'year',

• sortDirection: 'desc'

• };

```
 /
export class AutoSearchFilters { /**


  • Vehicle manufacturer name


  • Case-insensitive partial match

   *

  • @example 'Toyota', 'Honda', 'Ford'

   */
manufacturer?: string;


/**


  • Vehicle model name


  • Case-insensitive partial match

   *

  • @example 'Camry', 'Accord', 'F-150'

   */
model?: string;


/**
```

```
   • Minimum year (inclusive)

    *

   • @example 2020

    */
yearMin?: number;


/**

   • Maximum year (inclusive)

    *

   • @example 2024

    */
yearMax?: number;


/**

   • Vehicle body class/type (supports multiple selections)


   • Case-insensitive partial match


   • Can be a single value or array for multi-select
```

```
   *
  • @example 'Sedan', ['SUV', 'Truck'], 'Coupe'
   */
bodyClass?: string | string[];

/**
  • Minimum VIN instance count
  • Filter vehicles with at least this many VIN instances
   *
  • @example 10
   */
instanceCountMin?: number;

/**
  • Maximum VIN instance count
  • Filter vehicles with at most this many VIN instances
   *
  • @example 1000
   */
```

```
instanceCountMax?: number;


/**


  • Page number (1-indexed)


  • Used for pagination

   *

  • @default 1

   */
page?: number;


/**


  • Page size (number of results per page)


  • Used for pagination

   *

  • @default 20

   */
size?: number;


/**
```

```
   • Sort field

   • Field name to sort by
    *
   • @example 'manufacturer', 'model', 'year', 'instance_count'
    */
sort?: string;

/**
   • Sort direction

   • Ascending or descending order
    *
   • @default 'asc'
    */
sortDirection?: 'asc' | 'desc';

/**
   • Search query (global search)
```

```
   • Searches across multiple fields (manufacturer, model, body class)
    *
   • @example 'Toyota Camry'
    */
search?: string;

/**
   • Model combinations (from picker)
   • Comma-separated manufacturer:model pairs
    *
   • @example 'Ford:F-150,Toyota:Camry,Honda:Accord'
    */
modelCombos?: string;

/**
   • Constructor with default values
    *
   • @param partial - Partial AutoSearchFilters object
    */
```

```
constructor(partial?: Partial<AutoSearchFilters>) { Object.assign(this, partial); }



/**



  • Create filters from partial object

   *

  • @param partial - Partial filter object



  • @returns AutoSearchFilters instance

   */
static fromPartial(partial: Partial<AutoSearchFilters>): AutoSearchFilters { return
new AutoSearchFilters(partial); }



/**



  • Get default filters

   *

  • @returns Default filter values

   */
static getDefaults(): AutoSearchFilters { return new AutoSearchFilters({ page: 1,
size: 20, sort: 'manufacturer', sortDirection: 'asc' }); }



/**



  • Check if filters are empty (no active filters except pagination/sort)
```

```
   *
  • @returns True if no search filters are active
   */
isEmpty(): boolean { const hasBodyClass = Array.isArray(this.bodyClass) ?
this.bodyClass.length > 0 : !!this.bodyClass;



return ( !this.manufacturer && !this.model && !this.yearMin && !this.yearMax && !
hasBodyClass && !this.instanceCountMin && !this.instanceCountMax && !this.search && !
this.modelCombos ); }



/**



  • Clone filters
   *
  • @returns New AutoSearchFilters instance with same values
   */
clone(): AutoSearchFilters { return new AutoSearchFilters({ ...this }); }



/**



  • Merge with other filters
   *
  • @param other - Filters to merge
```

```
   • @returns New AutoSearchFilters with merged values

    */
merge(other: Partial<AutoSearchFilters>): AutoSearchFilters { return new
AutoSearchFilters({ ...this, ...other }); }



/**

   • Clear all filters except pagination and sort

    *
   • @returns New AutoSearchFilters with only pagination/sort

    */
clearSearch(): AutoSearchFilters { return new AutoSearchFilters({ page: this.page,
size: this.size, sort: this.sort, sortDirection: this.sortDirection }); }



/**

   • Get active filter count

    *
   • @returns Number of active search filters

    */
getActiveFilterCount(): number { let count = 0; if (this.manufacturer) count++; if
(this.model) count++; if (this.yearMin) count++; if (this.yearMax) count++; if
(this.bodyClass) count++; if (this.instanceCountMin) count++; if
(this.instanceCountMax) count++; if (this.search) count++; if (this.modelCombos)
count++; return count; } }
```

Step 403.2: Create the Statistics Models

Create the file `src/app/domains/automobile/models/automobile.statistics.ts`:

```
// src/app/domains/automobile/models/automobile.statistics.ts
// VERSION 1 (Section 403) - Automobile domain statistics models


/**


  • Manufacturer statistic

 *

  • Aggregated data for a single manufacturer

 */
export class ManufacturerStat { /**


  • Manufacturer name


  • @example 'Toyota'

   */
name!: string;


/**


  • Number of vehicle configurations


  • @example 234

   */
count!: number;
```

```
/**

  • Total VIN instances

  • @example 8456

   */
instanceCount?: number;

/**

  • Percentage of total

  • @example 18.8

   */
percentage!: number;

/**

  • Number of unique models

  • @example 42

   */
modelCount?: number;
```

```
constructor(partial?: Partial<ManufacturerStat>) { if (partial) { Object.assign(this,
partial); } }



static fromApiResponse(data: any): ManufacturerStat { return new
ManufacturerStat({ name: data.name || data.manufacturer, count: Number(data.count ||
data.vehicle_count || 0), instanceCount: data.instance_count || data.instanceCount,
percentage: Number(data.percentage || 0), modelCount: data.model_count ||
data.modelCount }); } }



/**



  • Model statistic


 *


  • Aggregated data for a single vehicle model


 */
export class ModelStat { /**



  • Model name



  • @example 'Camry'


   */
name!: string;



/**
```

```
   * Manufacturer name


   * @example 'Toyota'

    */
manufacturer!: string;


/**


   * Number of vehicle configurations


   * @example 15

    */
count!: number;


/**


   * Total VIN instances


   * @example 3456

    */
instanceCount!: number;


/**
```

```
  • Percentage of total instances



  • @example 7.6

   */
percentage!: number;



constructor(partial?: Partial<ModelStat>) { if (partial) { Object.assign(this,
partial); } }



static fromApiResponse(data: any): ModelStat { return new ModelStat({ name: data.name
|| data.model, manufacturer: data.manufacturer, count: Number(data.count ||
data.vehicle_count || 0), instanceCount: Number(data.instance_count ||
data.instanceCount || 0), percentage: Number(data.percentage || 0) }); }



/**



  • Get full model name including manufacturer



  • @returns Full name (e.g., "Toyota Camry")

   */
getFullName(): string { return `${this.manufacturer} ${this.name}`; } }



/**



  • Body class statistic
```

```
 *

  • Aggregated data for a single body class

 */
export class BodyClassStat { /**


  • Body class name


  • @example 'Sedan'

   */
name!: string;


/**


  • Number of vehicle configurations


  • @example 456

   */
count!: number;


/**


  • Total VIN instances


  • @example 16789
```

```
   */
instanceCount?: number;



/**



  • Percentage of total



  • @example 36.6

   */
percentage!: number;



constructor(partial?: Partial<BodyClassStat>) { if (partial) { Object.assign(this,
partial); } }



static fromApiResponse(data: any): BodyClassStat { return new BodyClassStat({ name:
data.name || data.body_class || data.bodyClass, count: Number(data.count ||
data.vehicle_count || 0), instanceCount: data.instance_count || data.instanceCount,
percentage: Number(data.percentage || 0) }); } }



/**



  • Year statistic

 *

  • Aggregated data for a single vehicle year

 */
export class YearStat { /**
```

```
   • Vehicle year


   • @example 2024

    */
year!: number;


/**

   • Number of vehicle configurations


   • @example 89

    */
count!: number;


/**

   • Total VIN instances


   • @example 3245

    */
instanceCount?: number;


/**
```

```
  • Percentage of total



  • @example 7.1

   */
percentage!: number;



constructor(partial?: Partial<YearStat>) { if (partial) { Object.assign(this,
partial); } }



static fromApiResponse(data: any): YearStat { return new YearStat({ year:
Number(data.year), count: Number(data.count || data.vehicle_count || 0),
instanceCount: data.instance_count || data.instanceCount, percentage:
Number(data.percentage || 0) }); }



/**



  • Check if this year is current year

   */
isCurrentYear(): boolean { return this.year === new Date().getFullYear(); }



/**



  • Get age of vehicles from this year

   */
getAge(): number { return new Date().getFullYear() - this.year; } }
```

```
/**

  • Vehicle statistics

 *

  • Aggregated statistics across all filtered vehicles.


  • Provides high-level metrics and distributions for analysis.

 *

  • @example


  •
```

typescript

• const stats: VehicleStatistics = {

• totalVehicles: 1247,

• totalInstances: 45623,

• manufacturerCount: 23,

• modelCount: 412,

• yearRange: { min: 2010, max: 2024 },

• averageInstancesPerVehicle: 36.6

• };

```
 /
export class VehicleStatistics { /**


  • Total number of unique vehicle configurations


  • @example 1247

   */
totalVehicles!: number;


/**


  • Total number of VIN instances across all vehicles


  • @example 45623

   */
totalInstances!: number;


/**


  • Number of unique manufacturers


  • @example 23

   */
manufacturerCount!: number;
```

```
/**

  • Number of unique models

  • @example 412

   */
modelCount!: number;

/**

  • Number of unique body classes

  • @example 8

   */
bodyClassCount?: number;

/**

  • Year range (min and max years in dataset)

   */
yearRange!: { min: number; max: number; };

/**
```

```
  • Average VIN instances per vehicle configuration

  • @example 36.6

   */
averageInstancesPerVehicle!: number;

/**

  • Median VIN instances per vehicle configuration

  • @example 28

   */
medianInstancesPerVehicle?: number;

/**

  • Top manufacturers by vehicle count (top 20)

   */
topManufacturers?: ManufacturerStat[];

/**

  • Top models by instance count (top 20)
```

```
   */
topModels?: ModelStat[];

/**

  • Distribution by body class

   */
bodyClassDistribution?: BodyClassStat[];

/**

  • Distribution by year

   */
yearDistribution?: YearStat[];

/**

  • Complete manufacturer distribution

   */
manufacturerDistribution?: ManufacturerStat[];

/**

  • Raw segmented statistics by manufacturer
```

```
  • Format: { "Toyota": { total: 234, highlighted: 45 } }

   */
byManufacturer?: Record<string, { total: number; highlighted: number }>;

/**

  • Raw segmented statistics by body class

   */
byBodyClass?: Record<string, { total: number; highlighted: number }>;

/**

  • Raw segmented statistics by year

   */
byYearRange?: Record<string, { total: number; highlighted: number }>;

/**

  • Raw segmented statistics by model per manufacturer

   */
modelsByManufacturer?: Record<string, Record<string, { total: number; highlighted:
number }>>;
```

```
constructor(partial?: Partial<VehicleStatistics>) { if (partial) { Object.assign(this,
partial); } }



/**



  • Create VehicleStatistics from API response

   *

  • Handles two formats:



  • 1. Array-based statistics (topManufacturers, etc.)



  • 2. Segmented statistics (byManufacturer, modelsByManufacturer, etc.)

   *

  • @param data - Raw API response data



  • @returns VehicleStatistics instance

   */
static fromApiResponse(data: any): VehicleStatistics { // Check if segmented format
const byYearData = data.byYearRange || data.byYear; if (data.byManufacturer ||
data.modelsByManufacturer || data.byBodyClass || byYearData) { return
VehicleStatistics.fromSegmentedStats(data); }



// Array-based format return new VehicleStatistics({ totalVehicles:
Number(data.total_vehicles || data.totalVehicles || 0), totalInstances:
Number(data.total_instances || data.totalInstances || 0), manufacturerCount:
Number(data.manufacturer_count || data.manufacturerCount || 0), modelCount:
Number(data.model_count || data.modelCount || 0), bodyClassCount:
data.body_class_count || data.bodyClassCount, yearRange: { min:
Number(data.year_range?.min || data.yearRange?.min || 0), max:
```

```
Number(data.year_range?.max || data.yearRange?.max || 0) },
averageInstancesPerVehicle: Number( data.average_instances_per_vehicle ||
data.averageInstancesPerVehicle || 0 ), medianInstancesPerVehicle:
data.median_instances_per_vehicle || data.medianInstancesPerVehicle, topManufacturers:
data.top_manufacturers?.map((m: any) => ManufacturerStat.fromApiResponse(m) ) ||
data.topManufacturers?.map((m: any) => ManufacturerStat.fromApiResponse(m) ),
topModels: data.top_models?.map((m: any) => ModelStat.fromApiResponse(m) ) ||
data.topModels?.map((m: any) => ModelStat.fromApiResponse(m) ), bodyClassDistribution:
data.body_class_distribution?.map((b: any) => BodyClassStat.fromApiResponse(b) ) ||
data.bodyClassDistribution?.map((b: any) => BodyClassStat.fromApiResponse(b) ),
yearDistribution: data.year_distribution?.map((y: any) =>
YearStat.fromApiResponse(y) ) || data.yearDistribution?.map((y: any) =>
YearStat.fromApiResponse(y) ) }); }



/**



  • Create from segmented statistics format


   */
private static fromSegmentedStats(data: any): VehicleStatistics { const byYearData =
data.byYearRange || data.byYear;



const topManufacturers =
VehicleStatistics.transformByManufacturer(data.byManufacturer); const topModels =
VehicleStatistics.transformModelsByManufacturer(data.modelsByManufacturer); const
bodyClassDistribution = VehicleStatistics.transformByBodyClass(data.byBodyClass);
const yearDistribution = VehicleStatistics.transformByYearRange(byYearData);



const totalVehicles = data.totalCount || 0; const manufacturerCount =
topManufacturers?.length || 0; const modelCount = topModels?.length || 0; const
bodyClassCount = bodyClassDistribution?.length || 0;



const years = yearDistribution?.map(y => y.year) || []; const yearRange = years.length
> 0 ? { min: Math.min(...years), max: Math.max(...years) } : { min: 0, max: 0 };



return new VehicleStatistics({ totalVehicles, totalInstances: totalVehicles,
manufacturerCount, modelCount, bodyClassCount, yearRange, averageInstancesPerVehicle:
```

```
0, topManufacturers, topModels, bodyClassDistribution, yearDistribution,
manufacturerDistribution: topManufacturers, byManufacturer: data.byManufacturer,
byBodyClass: data.byBodyClass, byYearRange: byYearData, modelsByManufacturer:
data.modelsByManufacturer }); }



/**



  • Transform byManufacturer object to ManufacturerStat array


   */
private static transformByManufacturer( byManufacturer: Record<string, any> |
undefined ): ManufacturerStat[] | undefined { if (!byManufacturer) return undefined;



const stats = Object.entries(byManufacturer).map(([name, countOrStats]) => { const
count = typeof countOrStats === 'object' ? (countOrStats.total || 0) : (countOrStats
|| 0);



return new ManufacturerStat({ name, count, instanceCount: count, percentage: 0,
modelCount: 0 }); });



stats.sort((a, b) => b.count - a.count);



const totalCount = stats.reduce((sum, s) => sum + s.count, 0); stats.forEach(s =>
{ s.percentage = totalCount > 0 ? (s.count / totalCount) * 100 : 0; });



return stats.slice(0, 20); }



/**



  • Transform modelsByManufacturer to ModelStat array
```

```
   */
private static transformModelsByManufacturer( modelsByManufacturer: Record<string,
Record<string, any>> | undefined ): ModelStat[] | undefined { if (!
modelsByManufacturer) return undefined;



const stats: ModelStat[] = []; let totalCount = 0;



Object.entries(modelsByManufacturer).forEach(([manufacturer, models]) =>
{ Object.entries(models).forEach(([modelName, countOrStats]) => { const instanceCount
= typeof countOrStats === 'object' ? (countOrStats.total || 0) : (countOrStats || 0);
totalCount += instanceCount; stats.push(new ModelStat({ name: modelName, manufacturer,
count: 1, instanceCount, percentage: 0 })); }); });



stats.sort((a, b) => b.instanceCount - a.instanceCount);



stats.forEach(s => { s.percentage = totalCount > 0 ? (s.instanceCount / totalCount) *
100 : 0; });



return stats.slice(0, 20); }



/**



  • Transform byBodyClass to BodyClassStat array


   */
private static transformByBodyClass( byBodyClass: Record<string, any> | undefined ):
BodyClassStat[] | undefined { if (!byBodyClass) return undefined;



const stats = Object.entries(byBodyClass).map(([name, countOrStats]) => { const count
= typeof countOrStats === 'object' ? (countOrStats.total || 0) : (countOrStats || 0);
```

```
return new BodyClassStat({ name, count, instanceCount: count, percentage: 0 }); });


const totalCount = stats.reduce((sum, s) => sum + s.count, 0); stats.forEach(s =>
{ s.percentage = totalCount > 0 ? (s.count / totalCount) * 100 : 0; });


stats.sort((a, b) => b.count - a.count);


return stats; }


/**


  • Transform byYearRange to YearStat array


   */
private static transformByYearRange( byYearRange: Record<string, any> | undefined ):
YearStat[] | undefined { if (!byYearRange) return undefined;


const stats = Object.entries(byYearRange).map(([yearStr, countOrStats]) => { const
count = typeof countOrStats === 'object' ? (countOrStats.total || 0) : (countOrStats
|| 0);


return new YearStat({ year: parseInt(yearStr, 10), count, instanceCount: count,
percentage: 0 }); });


const totalCount = stats.reduce((sum, s) => sum + s.count, 0); stats.forEach(s =>
{ s.percentage = totalCount > 0 ? (s.count / totalCount) * 100 : 0; });
```

```
stats.sort((a, b) => a.year - b.year);



return stats; }



/**



  • Get year span (number of years covered)

   */
getYearSpan(): number { return this.yearRange.max - this.yearRange.min + 1; }



/**



  • Get average vehicles per manufacturer

   */
getAverageVehiclesPerManufacturer(): number { return this.manufacturerCount > 0 ?
this.totalVehicles / this.manufacturerCount : 0; } }
```

Step 403.3: Update the Barrel File

Update `src/app/domains/automobile/models/index.ts`:

```
// src/app/domains/automobile/models/index.ts

// VERSION 3 (Section 403) - Complete domain models



export * from './automobile.data'; export * from './automobile.filters'; export * from
'./automobile.statistics';
```

Verification

1. Check Files Exist

```
$ ls -la src/app/domains/automobile/models/
```

Expected output:

```
total 20

drwxr-xr-x 2 user user 4096 Feb  9 12:00 . drwxr-xr-x 3 user user 4096 Feb  9 12:00 ..
-rw-r--r-- 1 user user 4567 Feb  9 12:00 automobile.data.ts -rw-r--r-- 1 user user
3456 Feb  9 12:00 automobile.filters.ts -rw-r--r-- 1 user user 8901 Feb  9 12:00
automobile.statistics.ts -rw-r--r-- 1 user user  200 Feb  9 12:00 index.ts
```

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/domains/automobile/models/index.ts
```

3. Test Filter Usage

```
import { AutoSearchFilters } from '@app/domains/automobile/models';

// Create filters const filters = new AutoSearchFilters({ manufacturer: 'Toyota',
yearMin: 2020, yearMax: 2024, page: 1, size: 20 });



console.log('Is empty:', filters.isEmpty()); // Output: false



console.log('Active count:', filters.getActiveFilterCount()); // Output: 3
(manufacturer, yearMin, yearMax)



// Clone and modify const newFilters = filters.merge({ bodyClass: 'SUV' });
console.log('Body class:', newFilters.bodyClass); // Output: 'SUV'
```

Common Problems

Symptom	Cause	Solution
Statistics empty	Wrong API format detection	Check for `byManufacturer` vs `topManufacturers`
Percentages all zero	Total count is zero	Check totalCount calculation
Year range wrong	Years not parsed as numbers	Use `parseInt()` for year keys
Highlight data missing	Not preserved in transformation	Keep raw `byManufacturer` etc.

Key Takeaways

- **Filter models map to URL and API** — Same structure for URL params and query strings

- **Statistics handle two formats** — Array-based and segmented for charts
- **Highlight filters enable chart segmentation** — `h_*` prefix for API parameters
- **Transformation preserves raw data** — Keep segmented data for chart highlighting

---

Acceptance Criteria

- [ ] `src/app/domains/automobile/models/automobile.filters.ts` exists
- [ ] `HighlightFilters` interface defined
- [ ] `AutoSearchFilters` class with all filter fields
- [ ] Filter utility methods: `isEmpty()`, `clone()`, `merge()`, `clearSearch()`
- [ ] `src/app/domains/automobile/models/automobile.statistics.ts` exists
- [ ] Statistic classes: `ManufacturerStat`, `ModelStat`, `BodyClassStat`, `YearStat`
- [ ] `VehicleStatistics` class with all aggregation fields
- [ ] `fromApiResponse()` handles both array and segmented formats
- [ ] Segmented data preserved for chart highlighting
- [ ] Barrel file exports all models
- [ ] TypeScript compilation succeeds
- [ ] JSDoc comments on all properties and methods

Phase 4 Complete

Congratulations! You have completed Phase 4: Domain Models.

What you built:

- Base model patterns with partial constructors
- `VehicleResult` and `VinInstance` data models
- `AutoSearchFilters` filter model
- `VehicleStatistics` and distribution stat models
- API response transformation for multiple formats

The Aha Moment: "Domain models are more than data containers — they encapsulate transformation logic, business rules, and computed values."

Next Step

Proceed to `501-domain-adapter-pattern.md` to begin Phase 5: Domain Adapters.

501: Domain Adapter Pattern

Status: Complete **Depends On:** 306-resource-management-service, 403-domain-filter-statistics-models **Blocks:** 502-url-mapper-adapter, 503-api-adapter

---

Learning Objectives

After completing this section, you will:

- Understand the Adapter pattern and why it enables domain-agnostic framework code
- Know the three adapter interfaces: IFilterUrlMapper, IApiAdapter, ICacheKeyBuilder
- Recognize how adapters bridge domain-specific logic to framework services
- Be able to design domain adapters for new business domains

Objective

Establish the adapter pattern that allows the framework's `ResourceManagementService` to work with any business domain. Define the interfaces that domain-specific adapters must implement.

Why

The `ResourceManagementService` (Section 306) provides:

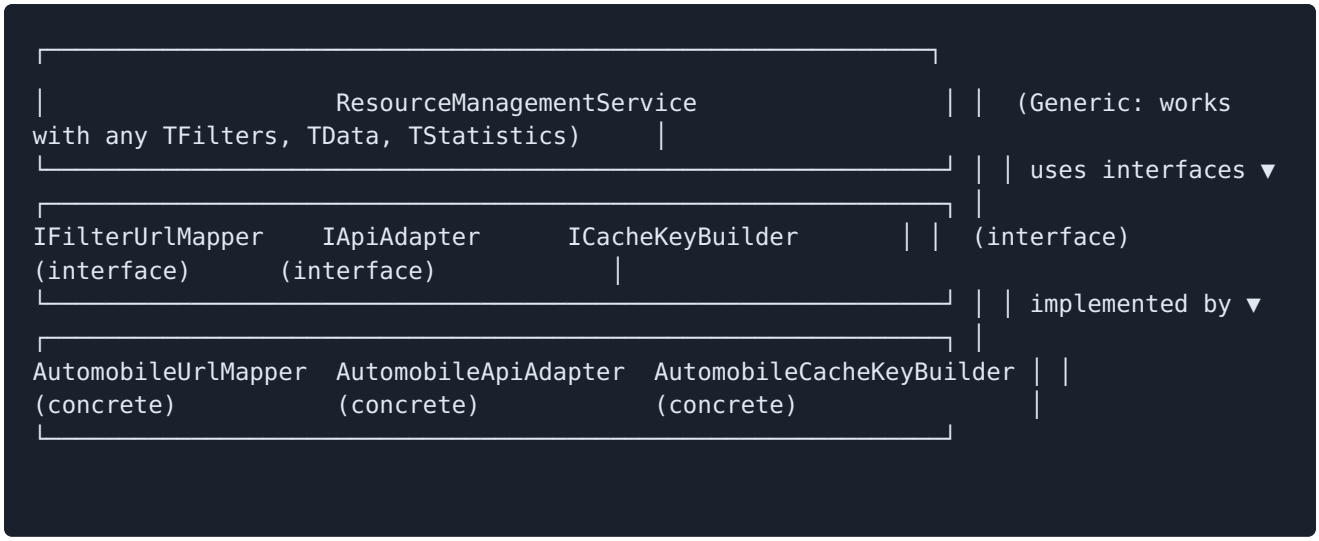
- URL-First state management
- Automatic data fetching on filter changes
- Response caching and deduplication
- Loading/error state management

But this service doesn't know about vehicles, manufacturers, or body classes. It doesn't know how to:

- Convert filters to URL parameters
- Make API calls for vehicle data
- Build cache keys for specific queries

The Adapter Pattern

Adapters bridge the gap between generic framework code and domain-specific logic:



The framework depends on **interfaces** (abstractions), not concrete implementations. Domain modules provide concrete implementations.

Why Three Adapters?

Each adapter handles one responsibility:

Adapter	Responsibility	Interface
URL Mapper	Convert filters ↔ URL params	`IFilterUrlMapper<TFilters>`
API Adapter	Fetch data from backend	`IApiAdapter<TFilters, TData, TStats>`
Cache Key Builder	Create unique cache keys	`ICacheKeyBuilder<TFilters>`

Separation of concerns: Each adapter does one thing well. This makes them:

- Easy to test individually

- Easy to replace or mock
- Easy to understand

Interface-Based Polymorphism

The framework uses TypeScript generics to work with any domain:

```
// Framework service (domain-agnostic)

class ResourceManagementService<TFilters, TData, TStatistics> { private urlMapper:
IFilterUrlMapper<TFilters>; private apiAdapter: IApiAdapter<TFilters, TData,
TStatistics>;



// Works with any domain that provides adapters updateFilters(partial:
Partial<TFilters>): void { ... } fetchData(): void { ... } }



// Automobile domain provides concrete types type AutoFilters = AutoSearchFilters;
type AutoData = VehicleResult; type AutoStats = VehicleStatistics;



// ResourceManagementService<AutoFilters, AutoData, AutoStats>
```

What

Step 501.1: Review the Adapter Interfaces

The adapter interfaces are defined in `src/app/framework/models/resource-management.interface.ts`. Review them to understand the contract each adapter must fulfill.

IFilterUrlMapper Interface:

```
/**

  • Adapter for mapping filters to/from URL parameters

 */
export interface IFilterUrlMapper<TFilters> { /**


  • Convert filters to URL query parameters

   */
toUrlParams(filters: TFilters): Params;


/**


  • Convert URL query parameters to filters

   */
fromUrlParams(params: Params): TFilters;


/**


  • Extract highlight filters from URL parameters (optional)

   */
extractHighlights?(params: Params): any; }
```

IApiAdapter Interface:

```
/**

  • Adapter for fetching data from API

 */
export interface IApiAdapter<TFilters, TData, TStatistics = any> { /**



  • Fetch data from API based on filters

   */
fetchData( filters: TFilters, highlights?: any ): Observable<ApiAdapterResponse<TData,
TStatistics>>; }



/**



  • Response from API adapter

 */
export interface ApiAdapterResponse<TData, TStatistics = any> { results: TData[];
total: number; statistics?: TStatistics; }
```

ICacheKeyBuilder Interface:

```
/**

  • Adapter for building cache keys from filters

 */
export interface ICacheKeyBuilder<TFilters> { /**


  • Build a unique cache key from filters

   */
buildKey(filters: TFilters, highlights?: any): string; }
```

Step 501.2: Create the Adapters Directory

Create the directory structure for automobile domain adapters:

```
$ mkdir -p src/app/domains/automobile/adapters
$ touch src/app/domains/automobile/adapters/index.ts
```

Create the barrel file `src/app/domains/automobile/adapters/index.ts`:

```
// src/app/domains/automobile/adapters/index.ts
// VERSION 1 (Section 501) - Automobile domain adapters barrel


// URL Mapper (Section 502) // export * from './automobile-url-mapper';


// API Adapter (Section 503) // export * from './automobile-api.adapter';


// Cache Key Builder (Section 503) // export * from './automobile-cache-key.builder';
```

Step 501.3: Understand Adapter Usage

The adapters are provided to `ResourceManagementService` via a configuration object:

```
// In domain config factory
const resourceConfig: ResourceManagementConfig< AutoSearchFilters, VehicleResult,
VehicleStatistics > = { filterMapper: new AutomobileUrlMapper(), apiAdapter: new
AutomobileApiAdapter(apiService, baseUrl), cacheKeyBuilder: new
AutomobileCacheKeyBuilder(), defaultFilters: AutoSearchFilters.getDefaults(),
autoFetch: true, cacheTTL: 30000 };


// ResourceManagementService uses these adapters const resourceService = new
ResourceManagementService( urlStateService, requestCoordinator, resourceConfig );
```

The adapters form the bridge between:

- **URL** (user-facing state)
- **Filters** (domain-specific parameters)
- **API** (backend data source)
- **Cache** (request deduplication)

Verification

1. Check Directory Exists

```
$ ls -la src/app/domains/automobile/adapters/
```

2. Review Interface File

```
$ cat src/app/framework/models/resource-management.interface.ts
```

Ensure the interface exports include:

- `IFilterUrlMapper<TFilters>`
- `IApiAdapter<TFilters, TData, TStatistics>`
- `ICacheKeyBuilder<TFilters>`
- `ApiAdapterResponse<TData, TStatistics>`
- `ResourceManagementConfig<TFilters, TData, TStatistics>`

Common Problems

Symptom	Cause	Solution
"Generic type requires arguments"	Missing type parameters	Provide `<TFilters, TData>` when implementing
Method signature mismatch	Wrong parameter/return types	Match interface signature exactly
Cannot find interface	Import path wrong	Import from `@app/framework/models`
"Property is missing"	Interface not fully implemented	Implement all required methods

---

Key Takeaways

- **Adapters enable domain-agnostic frameworks** — Framework code works with interfaces, not implementations
- **Three adapters, three responsibilities** — URL mapping, API fetching, cache key building
- **TypeScript generics preserve type safety** — `IApiAdapter<AutoFilters, VehicleResult>` is type-safe
- **Separation enables testing** — Each adapter can be unit tested in isolation

---

Acceptance Criteria

- [ ] Adapter interfaces understood: IFilterUrlMapper, IApiAdapter, ICacheKeyBuilder
- [ ] Adapters directory created: `src/app/domains/automobile/adapters/`
- [ ] Barrel file created with placeholder exports
- [ ] Adapter usage pattern understood (configuration object)
- [ ] Type parameters understood: TFilters, TData, TStatistics

---

Next Step

Proceed to `502-url-mapper-adapter.md` to implement the automobile URL mapper.

502: URL Mapper Adapter

Status: Complete **Depends On:** 501-domain-adapter-pattern, 403-domain-filter-statistics-models **Blocks:** 503-api-adapter, Phase 8 (Framework Components)

Learning Objectives

After completing this section, you will:

- Understand how to map domain filters to URL query parameters
- Know how to handle bidirectional type conversion (strings ↔ numbers)
- Recognize the importance of parameter naming consistency (URL ↔ API)
- Be able to implement highlight filter extraction for chart segmentation

---

Objective

Create the `AutomobileUrlMapper` that provides bidirectional conversion between `AutoSearchFilters` objects and URL query parameters. This adapter enables URL-First state management for the automobile domain.

Why

The URL-First architecture stores application state in the URL. When a user applies filters:

- **Filter object created:** `{ manufacturer: 'Toyota', yearMin: 2020, page: 1 }`
- **URL updated:** `?manufacturer=Toyota&yearMin=2020&page=1`
- **URL is shareable:** Copy/paste the URL to share the exact filter state

When a user navigates to a URL with parameters:

- **URL parsed:** `?manufacturer=Toyota&yearMin=2020`
- **Filter object created:** `{ manufacturer: 'Toyota', yearMin: 2020 }`
- **Data fetched:** API called with filter values

Bidirectional Mapping

The URL mapper handles two directions:

Direction	Method	Purpose
Filters → URL	`toUrlParams()`	Generate URL from filter state
URL → Filters	`fromUrlParams()`	Parse URL into filter state

Type Conversion

URL parameters are always strings. Filter objects have typed fields:

```
// URL: ?yearMin=2020&page=1
// Params: { yearMin: '2020', page: '1' }


// Filter object needs: // { yearMin: 2020, page: 1 } (numbers, not strings)
```

The mapper handles:

- **String → Number:** `'2020'` → `2020`
- **Number → String:** `2020` → `'2020'`
- **Array → String:** `['Sedan', 'SUV']` → `'Sedan,SUV'`
- **String → Array:** `'Sedan,SUV'` → `['Sedan', 'SUV']`

Parameter Naming

URL parameters should match API parameters for consistency:

```
URL:     ?yearMin=2020&yearMax=2024&sortBy=manufacturer
API:     GET /vehicles?yearMin=2020&yearMax=2024&sortBy=manufacturer
```

This allows the URL to be directly usable as API query string (with minor adjustments).

Highlight Filters

For chart highlighting, the mapper extracts `h_` prefixed parameters:

```
URL: ?manufacturer=Toyota&h_yearMin=2022&h_yearMax=2024
```

Extracted highlights: `{ yearMin: 2022, yearMax: 2024 }`

These are sent to the API for segmented statistics: `{ total: 234, highlighted: 45 }`

What

Step 502.1: Create the URL Mapper

Create the file `src/app/domains/automobile/adapters/automobile-url-mapper.ts`:

```
// src/app/domains/automobile/adapters/automobile-url-mapper.ts
// VERSION 1 (Section 502) - Automobile URL mapper adapter


import { Injectable } from '@angular/core'; import { Params } from '@angular/router';
import { IFilterUrlMapper } from '../../../framework/models/resource-
management.interface'; import { AutoSearchFilters } from '../models/
automobile.filters';


/**


  • Automobile filter URL mapper

 *

  • Bidirectional conversion between filter objects and URL query parameters.


  • URL parameter names match backend API parameter names (camelCase).

 *

  • URL Parameter Mapping:


  • - manufacturer → manufacturer


  • - model → model


  • - yearMin → yearMin


  • - yearMax → yearMax
```

```
 • - bodyClass → bodyClass

 • - page → page

 • - size → size

 • - sortBy → sort (filter property)

 • - sortOrder → sortDirection (filter property)
*
 • Highlight Parameters:

 • - h_yearMin, h_yearMax, h_manufacturer, etc.
*
 • @example

 •
```

typescript

- const mapper = new AutomobileUrlMapper();

*

- // To URL
- const filters = new AutoSearchFilters({
- manufacturer: 'Toyota',
- yearMin: 2020,
- page: 1
- });
- const params = mapper.toUrlParams(filters);
- // { manufacturer: 'Toyota', yearMin: '2020', page: '1' }

*

- // From URL
- const urlParams = { manufacturer: 'Toyota', yearMin: '2020', page: '1' };
- const filters = mapper.fromUrlParams(urlParams);
- // AutoSearchFilters { manufacturer: 'Toyota', yearMin: 2020, page: 1 }

```
 /
@Injectable({ providedIn: 'root' }) export class AutomobileUrlMapper implements
IFilterUrlMapper<AutoSearchFilters> { /**



  • URL parameter names (match backend API parameter names)


   */
private readonly PARAM_NAMES = { manufacturer: 'manufacturer', model: 'model',
yearMin: 'yearMin', yearMax: 'yearMax', bodyClass: 'bodyClass', instanceCountMin:
'instanceCountMin', instanceCountMax: 'instanceCountMax', search: 'search',
modelCombos: 'modelCombos', page: 'page', size: 'size', sort: 'sortBy', sortDirection:
'sortOrder' };



/**



  • Convert filters to URL query parameters


   *

  • Maps filter object fields to URL parameter names.



  • Only includes non-null/undefined values.



  • Converts all values to strings for URL compatibility.


   *

  • @param filters - Filter object



  • @returns URL query parameters
```

```
   */
toUrlParams(filters: AutoSearchFilters): Params { const params: Params = {};



// Search filters if (filters.manufacturer !== undefined && filters.manufacturer !==
null) { params[this.PARAM_NAMES.manufacturer] = filters.manufacturer; }



if (filters.model !== undefined && filters.model !== null)
{ params[this.PARAM_NAMES.model] = filters.model; }



if (filters.yearMin !== undefined && filters.yearMin !== null)
{ params[this.PARAM_NAMES.yearMin] = String(filters.yearMin); }



if (filters.yearMax !== undefined && filters.yearMax !== null)
{ params[this.PARAM_NAMES.yearMax] = String(filters.yearMax); }



if (filters.bodyClass !== undefined && filters.bodyClass !== null) { // Handle array
values (multiselect) - join with comma if (Array.isArray(filters.bodyClass)) { if
(filters.bodyClass.length > 0) { params[this.PARAM_NAMES.bodyClass] =
filters.bodyClass.join(','); } } else { params[this.PARAM_NAMES.bodyClass] =
filters.bodyClass; } }



if (filters.instanceCountMin !== undefined && filters.instanceCountMin !== null)
{ params[this.PARAM_NAMES.instanceCountMin] = String(filters.instanceCountMin); }



if (filters.instanceCountMax !== undefined && filters.instanceCountMax !== null)
{ params[this.PARAM_NAMES.instanceCountMax] = String(filters.instanceCountMax); }



if (filters.search !== undefined && filters.search !== null)
{ params[this.PARAM_NAMES.search] = filters.search; }
```

```
if (filters.modelCombos !== undefined && filters.modelCombos !== null)
{ params[this.PARAM_NAMES.modelCombos] = filters.modelCombos; }



// Pagination if (filters.page !== undefined && filters.page !== null)
{ params[this.PARAM_NAMES.page] = String(filters.page); }



if (filters.size !== undefined && filters.size !== null)
{ params[this.PARAM_NAMES.size] = String(filters.size); }



// Sorting if (filters.sort !== undefined && filters.sort !== null)
{ params[this.PARAM_NAMES.sort] = filters.sort; }



if (filters.sortDirection !== undefined && filters.sortDirection !== null)
{ params[this.PARAM_NAMES.sortDirection] = filters.sortDirection; }



return params; }



/**



  • Convert URL query parameters to filters

   *

  • Maps URL parameter names back to filter object fields.



  • Performs type conversion (strings to numbers).



  • Returns filter object with only defined values.
```

```
   *

  • @param params - URL query parameters



  • @returns Filter object


   */
fromUrlParams(params: Params): AutoSearchFilters { const filters = new
AutoSearchFilters();



// Search filters if (params[this.PARAM_NAMES.manufacturer]) { filters.manufacturer =
String(params[this.PARAM_NAMES.manufacturer]); }



if (params[this.PARAM_NAMES.model]) { filters.model =
String(params[this.PARAM_NAMES.model]); }



if (params[this.PARAM_NAMES.yearMin]) { const value =
this.parseNumber(params[this.PARAM_NAMES.yearMin]); if (value !== null)
{ filters.yearMin = value; } }



if (params[this.PARAM_NAMES.yearMax]) { const value =
this.parseNumber(params[this.PARAM_NAMES.yearMax]); if (value !== null)
{ filters.yearMax = value; } }



if (params[this.PARAM_NAMES.bodyClass]) { const bodyClassParam =
String(params[this.PARAM_NAMES.bodyClass]); // Check if it contains comma (multiple
values) if (bodyClassParam.includes(',')) { filters.bodyClass =
bodyClassParam.split(',').map(v => v.trim()); } else { // Single value - return as
array for consistency with multiselect filters.bodyClass = [bodyClassParam]; } }



if (params[this.PARAM_NAMES.instanceCountMin]) { const value =
this.parseNumber(params[this.PARAM_NAMES.instanceCountMin]); if (value !== null)
{ filters.instanceCountMin = value; } }
```

```
if (params[this.PARAM_NAMES.instanceCountMax]) { const value =
this.parseNumber(params[this.PARAM_NAMES.instanceCountMax]); if (value !== null)
{ filters.instanceCountMax = value; } }



if (params[this.PARAM_NAMES.search]) { filters.search =
String(params[this.PARAM_NAMES.search]); }



if (params[this.PARAM_NAMES.modelCombos]) { filters.modelCombos =
String(params[this.PARAM_NAMES.modelCombos]); }



// Pagination if (params[this.PARAM_NAMES.page]) { const value =
this.parseNumber(params[this.PARAM_NAMES.page]); if (value !== null) { filters.page =
value; } }



if (params[this.PARAM_NAMES.size]) { const value =
this.parseNumber(params[this.PARAM_NAMES.size]); if (value !== null) { filters.size =
value; } }



// Sorting if (params[this.PARAM_NAMES.sort]) { filters.sort =
String(params[this.PARAM_NAMES.sort]); }



if (params[this.PARAM_NAMES.sortDirection]) { const direction =
String(params[this.PARAM_NAMES.sortDirection]); if (direction === 'asc' || direction
=== 'desc') { filters.sortDirection = direction; } }



return filters; }



/**
```

```
  • Extract highlight filters from URL parameters

   *

  • Looks for 'h_' prefixed parameters for chart highlighting.



  • These enable segmented statistics (total vs highlighted).

   *

  • @param params - URL query parameters



  • @returns Highlight filters object

   */
extractHighlights(params: Params): Record<string, any> { const highlights:
Record<string, any> = {}; const prefix = 'h_';



Object.keys(params).forEach(key => { if (key.startsWith(prefix)) { const highlightKey
= key.substring(prefix.length); let value = params[key];



// Normalize separators: Convert pipes to commas if (typeof value === 'string' &&
value.includes('|')) { value = value.replace(/\|/g, ','); }



highlights[highlightKey] = value; } });



return highlights; }



/**
```

```
   • Parse number from URL parameter value
    *
   • @param value - URL parameter value
   • @returns Parsed number or null
    */
private parseNumber(value: any): number | null { if (value === null || value ===
undefined || value === '') { return null; }

const num = Number(value); return isNaN(num) ? null : num; }

/**
   • Get parameter name mapping
    *
   • @returns Parameter name mapping object
    */
getParameterMapping(): Record<string, string> { return { ...this.PARAM_NAMES }; }

/**
   • Build shareable URL from filters
```

```
   *
  • @param baseUrl - Base URL (e.g., '/discover')
  • @param filters - Filter object
  • @returns Complete URL with query parameters
   */
buildShareableUrl(baseUrl: string, filters: AutoSearchFilters): string { const params
= this.toUrlParams(filters); const queryString = Object.entries(params) .map(([key,
value]) => ${key}=${encodeURIComponent(value)}) .join('&');
return queryString ? ${baseUrl}?${queryString} : baseUrl; }
/**
  • Validate URL parameters
   *
  • @param params - URL query parameters
  • @returns Validation result with errors
   */
validateUrlParams(params: Params): { valid: boolean; errors: string[] } { const
errors: string[] = [];
```

```
// Check numeric fields const numericFields = [ this.PARAM_NAMES.yearMin,
this.PARAM_NAMES.yearMax, this.PARAM_NAMES.instanceCountMin,
this.PARAM_NAMES.instanceCountMax, this.PARAM_NAMES.page, this.PARAM_NAMES.size ];



numericFields.forEach(field => { if (params[field] !== undefined && params[field] !==
null) { const value = this.parseNumber(params[field]); if (value === null)
{ errors.push(`Invalid numeric value for ${field}: ${params[field]}`); } } });



// Check sort direction if (params[this.PARAM_NAMES.sortDirection]) { const direction
= String(params[this.PARAM_NAMES.sortDirection]); if (direction !== 'asc' &&
direction !== 'desc')
{ errors.push(`Invalid sort direction: ${direction}. Must be 'asc' or
'desc'.`); } }



// Check year range if (params[this.PARAM_NAMES.yearMin] &&
params[this.PARAM_NAMES.yearMax]) { const min =
this.parseNumber(params[this.PARAM_NAMES.yearMin]); const max =
this.parseNumber(params[this.PARAM_NAMES.yearMax]); if (min !== null && max !== null
&& min > max)
{ errors.push(`Year minimum (${min}) cannot be greater than maximum ($
{max})`); } }



return { valid: errors.length === 0, errors }; } }
```

Step 502.2: Update the Barrel File

Update `src/app/domains/automobile/adapters/index.ts`:

```
// src/app/domains/automobile/adapters/index.ts
// VERSION 2 (Section 502) - Added URL mapper


export * from './automobile-url-mapper';


// API Adapter (Section 503) // export * from './automobile-api.adapter';


// Cache Key Builder (Section 503) // export * from './automobile-cache-key.builder';
```

Verification

1. Check File Exists

```
$ ls -la src/app/domains/automobile/adapters/automobile-url-mapper.ts
```

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/domains/automobile/adapters/automobile-url-mapper.ts
```

3. Test Mapping

```
import { AutomobileUrlMapper } from '@app/domains/automobile/adapters';

import { AutoSearchFilters } from '@app/domains/automobile/models';



const mapper = new AutomobileUrlMapper();



// Test toUrlParams const filters = new AutoSearchFilters({ manufacturer: 'Toyota',
yearMin: 2020, yearMax: 2024, bodyClass: ['Sedan', 'SUV'], page: 1, size: 20 });



const params = mapper.toUrlParams(filters); console.log('URL params:', params); //
{ manufacturer: 'Toyota', yearMin: '2020', yearMax: '2024', //   bodyClass:
'Sedan,SUV', page: '1', size: '20' }



// Test fromUrlParams const urlParams = { manufacturer: 'Honda', yearMin: '2022',
page: '2' };



const parsedFilters = mapper.fromUrlParams(urlParams); console.log('Parsed filters:',
parsedFilters); // AutoSearchFilters { manufacturer: 'Honda', yearMin: 2022, page: 2 }



// Test highlight extraction const highlightParams = { manufacturer: 'Ford',
h_yearMin: '2023', h_yearMax: '2024' };



const highlights = mapper.extractHighlights(highlightParams);
console.log('Highlights:', highlights); // { yearMin: '2023', yearMax: '2024' }
```

Common Problems

Symptom	Cause	Solution
Numbers come back as strings	Missing `parseNumber()` call	Use `parseNumber()` for numeric fields
Array comes back as string	Missing comma split	Check for comma and split
Highlight params not extracted	Wrong prefix check	Ensure `h_` prefix matching
Empty string in URL	Null/undefined not filtered	Check for null/undefined before adding

---

Key Takeaways

- **Bidirectional mapping is essential** — Filters ↔ URL must work both ways
- **Type conversion is critical** — URL strings must become proper types
- **Parameter naming matters** — Consistency between URL and API simplifies debugging
- **Highlight extraction enables charts** — `h_` prefix separates highlight from filter params

---

Acceptance Criteria

- [ ] `src/app/domains/automobile/adapters/automobile-url-mapper.ts` exists
- [ ] Implements `IFilterUrlMapper<AutoSearchFilters>` interface
- [ ] `toUrlParams()` converts all filter fields to URL params
- [ ] `fromUrlParams()` parses all URL params to filter fields
- [ ] Numeric fields properly converted (string ↔ number)
- [ ] Array fields properly handled (bodyClass multiselect)
- [ ] `extractHighlights()` finds `h_` prefixed params
- [ ] Validation method checks for invalid values
- [ ] Barrel file exports the mapper
- [ ] TypeScript compilation succeeds
- [ ] JSDoc comments on all public methods

Next Step

Proceed to `503-api-adapter.md` to implement the automobile API adapter.

503: API Adapter

Status: Complete **Depends On:** 501-domain-adapter-pattern, 502-url-mapper-adapter, 302-api-service **Blocks:** Phase 8 (Framework Components), Phase 9 (Feature Components)

Learning Objectives

After completing this section, you will:

- Understand how to implement domain-specific API adapters
- Know how to transform API responses to domain models
- Recognize the difference between services and adapter classes
- Be able to implement cache key builders for request deduplication

---

Objective

Create the `AutomobileApiAdapter` for fetching vehicle data and the `AutomobileCacheKeyBuilder` for generating unique cache keys. These adapters complete the domain adapter layer.

Why

API Adapter: Domain-Specific Fetching

The framework's `ResourceManagementService` knows *when* to fetch data (on filter changes), but not *how* to fetch automobile data. The API adapter provides:

- **Endpoint knowledge** — Which URL to call (`/vehicles/details`)
- **Parameter mapping** — How to convert filters to API params

- **Response transformation** — How to convert API JSON to domain models

```
// Framework calls:
apiAdapter.fetchData(filters, highlights)



// Adapter implements: fetchData(filters) { const params =
this.filtersToApiParams(filters); return this.api.get('/vehicles/details',
params).pipe( map(response => ({ results: response.results.map(r =>
VehicleResult.fromApiResponse(r)), total: response.total, statistics:
VehicleStatistics.fromApiResponse(response.statistics) })) ); }
```

Not an Angular Service

The API adapter is a **plain class**, not an `@Injectable()` service. This is intentional:

Aspect	Service	Adapter Class
Lifecycle	Singleton	Created per config
Constructor	DI by Angular	Manual with args
Testability	Needs TestBed	Plain new()
Configuration	Environment-based	Constructor params

The adapter is instantiated in the domain config factory with specific configuration (base URL, API service).

Cache Key Builder

The `RequestCoordinatorService` deduplicates identical requests. It needs a unique key for each request. The cache key builder creates this key from filters:

```
// Same filters = same cache key = deduplicated

buildKey({ manufacturer: 'Toyota', page: 1 }) // → 'vehicles:manufacturer=Toyota|
page=1'



buildKey({ manufacturer: 'Toyota', page: 1 }) // → 'vehicles:manufacturer=Toyota|
page=1' (same, deduplicated)



buildKey({ manufacturer: 'Honda', page: 1 }) // → 'vehicles:manufacturer=Honda|
page=1' (different, new request)
```

Highlight Parameters in Cache Key

Highlight filters must be included in the cache key because they affect the response:

```
// Different highlights = different cache key

buildKey({ manufacturer: 'Toyota' }, { yearMin: 2020 }) // →
'vehicles:manufacturer=Toyota|h_yearMin=2020'



buildKey({ manufacturer: 'Toyota' }, { yearMin: 2022 }) // →
'vehicles:manufacturer=Toyota|h_yearMin=2022' (different)
```

What

Step 503.1: Create the API Adapter

Create the file `src/app/domains/automobile/adapters/automobile-api.adapter.ts`:

```
// src/app/domains/automobile/adapters/automobile-api.adapter.ts
// VERSION 1 (Section 503) - Automobile API adapter

import { Observable } from 'rxjs'; import { map } from 'rxjs/operators'; import
{ IApiAdapter, ApiAdapterResponse } from '../../../framework/models/resource-
management.interface'; import { ApiResponse } from '../../../framework/models/api-
response.interface'; import { ApiService } from '../../../framework/services/
api.service'; import { AutoSearchFilters } from '../models/automobile.filters'; import
{ VehicleResult } from '../models/automobile.data'; import { VehicleStatistics } from
'../models/automobile.statistics';

/**

  • Automobile API adapter

 *

  • Fetches vehicle data from the automobile discovery API.

  • Transforms API responses into domain models.

 *

  • NOTE: This is NOT an Angular service. It's instantiated manually

  • in the domain config factory with the base URL.

 *

  • @example

  •
```

typescript

- const adapter = new AutomobileApiAdapter(apiService, 'http://api.example.com/v1');
- const filters = new AutoSearchFilters({ manufacturer: 'Toyota' });

*

- adapter.fetchData(filters).subscribe(response => {
- console.log('Vehicles:', response.results);
- console.log('Total:', response.total);
- console.log('Statistics:', response.statistics);
- });

```
 /
export class AutomobileApiAdapter implements IApiAdapter<AutoSearchFilters,
VehicleResult, VehicleStatistics> { /**



  • API endpoint for vehicle search


   */
private readonly VEHICLES_ENDPOINT = '/vehicles/details';



/**



  • API endpoint for statistics only


   */
private readonly STATISTICS_ENDPOINT = '/statistics';



/**



  • Base URL for API requests


   */
private baseUrl: string;



/**



  • API service for making HTTP requests


   */
private apiService: ApiService;
```

```
/**


  • Constructor

   *

  • @param apiService - Injected API service for HTTP requests


  • @param baseUrl - Base URL for automobile API

   */
constructor(apiService: ApiService, baseUrl: string) { this.apiService = apiService;
this.baseUrl = baseUrl; }


/**


  • Fetch vehicle data from API

   *

  • @param filters - Search filters

    @param highlights - Optional highlight filters (h_ parameters)

  • @returns Observable of vehicle results with statistics

   */
fetchData( filters: AutoSearchFilters, highlights?: any ):
Observable<ApiAdapterResponse<VehicleResult, VehicleStatistics>> { // Convert filters
to API parameters const params = this.filtersToApiParams(filters, highlights); const
url = ${this.baseUrl}${this.VEHICLES_ENDPOINT};
```

```
// Fetch vehicle data return this.apiService.get<VehicleResult>(url,
{ params }).pipe( map((apiResponse: ApiResponse<VehicleResult>) => { // Transform API
response to adapter response return { results: apiResponse.results.map(item =>
VehicleResult.fromApiResponse(item) ), total: apiResponse.total, statistics:
apiResponse.statistics ? VehicleStatistics.fromApiResponse(apiResponse.statistics) :
undefined }; }) ); }



/**



  • Fetch statistics only (without vehicle data)

   *

  • Useful for refreshing statistics panel without reloading table data.

   *

  • @param filters - Search filters



  • @returns Observable of statistics

   */
fetchStatistics(filters: AutoSearchFilters): Observable<VehicleStatistics> { const
params = this.filtersToApiParams(filters);



return this.apiService .get<VehicleStatistics>( ${this.baseUrl}$
{this.STATISTICS_ENDPOINT}, params) .pipe( map((response: any) => { // API might
return statistics directly or wrapped const statsData = response.statistics ||
response; return VehicleStatistics.fromApiResponse(statsData); }) ); }



/**
```

```
  • Convert filter object to API query parameters

   *

  • @param filters - Domain filters

    @param highlights - Optional highlight filters (h_ parameters)

  • @returns API query parameters

   */
private filtersToApiParams( filters: AutoSearchFilters, highlights?: any ):
Record<string, any> { const params: Record<string, any> = {};



// Search filters if (filters.manufacturer) { params['manufacturer'] =
filters.manufacturer; }



if (filters.model) { params['model'] = filters.model; }



if (filters.yearMin !== undefined && filters.yearMin !== null) { params['yearMin'] =
filters.yearMin; }



if (filters.yearMax !== undefined && filters.yearMax !== null) { params['yearMax'] =
filters.yearMax; }



if (filters.bodyClass) { params['bodyClass'] = filters.bodyClass; }



if (filters.instanceCountMin !== undefined && filters.instanceCountMin !== null)
{ params['instanceCountMin'] = filters.instanceCountMin; }
```

```
if (filters.instanceCountMax !== undefined && filters.instanceCountMax !== null)
{ params['instanceCountMax'] = filters.instanceCountMax; }



if (filters.search) { params['search'] = filters.search; }



// Model combinations from picker if (filters.modelCombos) { params['models'] =
filters.modelCombos; }



// Pagination if (filters.page !== undefined && filters.page !== null)
{ params['page'] = filters.page; }



if (filters.size !== undefined && filters.size !== null) { params['size'] =
filters.size; }



// Sorting if (filters.sort) { params['sortBy'] = filters.sort; }



if (filters.sortDirection) { params['sortOrder'] = filters.sortDirection; }



// Highlight parameters (h_* prefix for segmented statistics) if (highlights) { if
(highlights.yearMin !== undefined && highlights.yearMin !== null)
{ params['h_yearMin'] = String(highlights.yearMin); }



if (highlights.yearMax !== undefined && highlights.yearMax !== null)
{ params['h_yearMax'] = String(highlights.yearMax); }



if (highlights.manufacturer) { params['h_manufacturer'] = highlights.manufacturer; }



if (highlights.modelCombos) { params['h_modelCombos'] = highlights.modelCombos; }
```

```
if (highlights.bodyClass) { params['h_bodyClass'] = highlights.bodyClass; } }


return params; } }
```

Step 503.2: Create the Cache Key Builder

Create the file `src/app/domains/automobile/adapters/automobile-cache-key.builder.ts`:

```
// src/app/domains/automobile/adapters/automobile-cache-key.builder.ts

// VERSION 1 (Section 503) - Automobile cache key builder



import { ICacheKeyBuilder } from '../../../framework/models/resource-
management.interface'; import { AutoSearchFilters } from '../models/
automobile.filters';



/**



  • Automobile cache key builder

 *

  • Creates unique cache keys from filter objects for request deduplication.



  • Keys are deterministic: same filters always produce the same key.

 *

  • Key format: vehicles:{filter1}={value1}|{filter2}={value2}|...

 *

  • @example



  •
```

typescript

- const builder = new AutomobileCacheKeyBuilder();

*

    • const key = builder.buildKey({ manufacturer: 'Toyota', page: 1 });

    • // → 'vehicles:manufacturer=Toyota|page=1'

*

    • const keyWithHighlights = builder.buildKey(

    • { manufacturer: 'Toyota' },

    • { yearMin: 2020 }

    •);

    • // → 'vehicles:manufacturer=Toyota|h_yearMin=2020'

```
 /
export class AutomobileCacheKeyBuilder implements ICacheKeyBuilder<AutoSearchFilters>
{ /**



  • Prefix for all automobile cache keys


   */
private readonly PREFIX = 'vehicles';



/**



  • Build a unique cache key from filters


   *

  • @param filters - Filter object



  • @param highlights - Optional highlight filters



  • @returns Cache key string


   */
buildKey(filters: AutoSearchFilters, highlights?: any): string { const parts: string[]
= [];



// Add filter parts (sorted for consistency) if (filters.manufacturer)
{ parts.push(`manufacturer=${filters.manufacturer}`); }



if (filters.model) { parts.push(`model=${filters.model}`); }
```

```
if (filters.yearMin !== undefined && filters.yearMin !== null) { parts.push(`yearMin=${filters.yearMin}`); }

if (filters.yearMax !== undefined && filters.yearMax !== null) { parts.push(`yearMax=${filters.yearMax}`); }

if (filters.bodyClass) { const bodyClassValue = Array.isArray(filters.bodyClass) ? filters.bodyClass.join(',') : filters.bodyClass; parts.push(`bodyClass=${bodyClassValue}`); }

if (filters.instanceCountMin !== undefined && filters.instanceCountMin !== null) { parts.push(`instanceCountMin=${filters.instanceCountMin}`); }

if (filters.instanceCountMax !== undefined && filters.instanceCountMax !== null) { parts.push(`instanceCountMax=${filters.instanceCountMax}`); }

if (filters.search) { parts.push(`search=${filters.search}`); }

if (filters.modelCombos) { parts.push(`modelCombos=${filters.modelCombos}`); }

// Pagination if (filters.page !== undefined && filters.page !== null) { parts.push(`page=${filters.page}`); }

if (filters.size !== undefined && filters.size !== null) { parts.push(`size=${filters.size}`); }

// Sorting if (filters.sort) { parts.push(`sort=${filters.sort}`); }
```

```
if (filters.sortDirection) { parts.push(`sortDir=${filters.sortDirection}`); }


// Add highlight parts (with h_ prefix) if (highlights) { if (highlights.yearMin !==
undefined && highlights.yearMin !== null) { parts.push(`h_yearMin=$
{highlights.yearMin}`); }


if (highlights.yearMax !== undefined && highlights.yearMax !== null)
{ parts.push(`h_yearMax=${highlights.yearMax}`); }


if (highlights.manufacturer) { parts.push(`h_manufacturer=$
{highlights.manufacturer}`); }


if (highlights.modelCombos) { parts.push(`h_modelCombos=$
{highlights.modelCombos}`); }


if (highlights.bodyClass) { parts.push(`h_bodyClass=${highlights.bodyClass}`); } }


// Build final key return parts.length > 0 ? `${this.PREFIX}:${parts.join('|')}` :
this.PREFIX; } }
```

Step 503.3: Update the Barrel File

Update `src/app/domains/automobile/adapters/index.ts`:

```
// src/app/domains/automobile/adapters/index.ts

// VERSION 3 (Section 503) - Complete adapters



export * from './automobile-url-mapper'; export * from './automobile-api.adapter';
export * from './automobile-cache-key.builder';
```

Verification

1. Check Files Exist

```
$ ls -la src/app/domains/automobile/adapters/
```

Expected output:

```
total 20

drwxr-xr-x 2 user user 4096 Feb  9 12:00 . drwxr-xr-x 3 user user 4096 Feb  9 12:00 ..
-rw-r--r-- 1 user user 4567 Feb  9 12:00 automobile-api.adapter.ts -rw-r--r-- 1 user
user 2345 Feb  9 12:00 automobile-cache-key.builder.ts -rw-r--r-- 1 user user 6789
Feb  9 12:00 automobile-url-mapper.ts -rw-r--r-- 1 user user  200 Feb  9 12:00
index.ts
```

2. TypeScript Compilation Check

```
$ npx tsc --noEmit src/app/domains/automobile/adapters/index.ts
```

3. Test API Adapter

```
import { AutomobileApiAdapter } from '@app/domains/automobile/adapters';

import { ApiService } from '@app/framework/services'; import { AutoSearchFilters }
from '@app/domains/automobile/models';



// Create adapter (normally done in domain config factory) const adapter = new
AutomobileApiAdapter(apiService, 'http://localhost:3000/api/v1');



// Test fetch const filters = new AutoSearchFilters({ manufacturer: 'Toyota', yearMin:
2020, page: 1, size: 20 });



adapter.fetchData(filters).subscribe(response => { console.log('Results:',
response.results.length); console.log('Total:', response.total); console.log('First
vehicle:', response.results[0]?.getDisplayName()); });
```

4. Test Cache Key Builder

```
import { AutomobileCacheKeyBuilder } from '@app/domains/automobile/adapters';
import { AutoSearchFilters } from '@app/domains/automobile/models';



const builder = new AutomobileCacheKeyBuilder();



// Test basic key const key1 = builder.buildKey(new AutoSearchFilters({ manufacturer:
'Toyota', page: 1 })); console.log('Key 1:', key1); // →
'vehicles:manufacturer=Toyota|page=1'



// Test with highlights const key2 = builder.buildKey( new
AutoSearchFilters({ manufacturer: 'Toyota' }), { yearMin: 2020, yearMax: 2024 } );
console.log('Key 2:', key2); // → 'vehicles:manufacturer=Toyota|h_yearMin=2020|
h_yearMax=2024'



// Test determinism (same filters = same key) const key3 = builder.buildKey(new
AutoSearchFilters({ manufacturer: 'Toyota', page: 1 })); console.log('Keys match:',
key1 === key3); // → true
```

Common Problems

Symptom	Cause	Solution
"Cannot read property 'get'"	ApiService not passed	Pass ApiService to constructor
Results not transformed	Missing `fromApiResponse()`	Call model's static factory
Cache key collision	Missing filter in key	Add filter to `buildKey()`
Highlights not in cache key	Highlights not added	Include h_* params in key

---

Key Takeaways

- **Adapters are plain classes, not services** — Enables manual construction with config
- **Response transformation uses domain models** — `VehicleResult.fromApiResponse()`
- **Cache keys must be deterministic** — Same filters always produce same key
- **Highlights affect cache keys** — Different highlights = different cached response

---

Acceptance Criteria

- [ ] `src/app/domains/automobile/adapters/automobile-api.adapter.ts` exists
- [ ] Implements `IApiAdapter<AutoSearchFilters, VehicleResult, VehicleStatistics>`
- [ ] `fetchData()` makes API call and transforms response
- [ ] Response transformation uses `VehicleResult.fromApiResponse()`
- [ ] Statistics transformation uses `VehicleStatistics.fromApiResponse()`
- [ ] `src/app/domains/automobile/adapters/automobile-cache-key.builder.ts` exists
- [ ] Implements `ICacheKeyBuilder<AutoSearchFilters>`
- [ ] `buildKey()` produces deterministic keys
- [ ] Cache key includes highlight parameters
- [ ] Barrel file exports all adapters
- [ ] TypeScript compilation succeeds
- [ ] JSDoc comments on all public methods

Phase 5 Complete

Congratulations! You have completed Phase 5: Domain Adapters.

What you built:

- Adapter pattern understanding and interfaces
- `AutomobileUrlMapper` for URL ↔ filter conversion
- `AutomobileApiAdapter` for API data fetching
- `AutomobileCacheKeyBuilder` for request deduplication

The Aha Moment: "Adapters are the bridge between generic framework code and domain-specific logic. They enable reusability without sacrificing type safety."

Next Step

Proceed to `801-base-table-component.md` to begin Phase 8: Framework Components.

601: Filter Definitions

Status: Planning **Depends On:** 401-automobile-filters-model, 203-filter-definition-interface **Blocks:** 604-query-control-filters, 607-domain-config-assembly

---

Learning Objectives

After completing this section, you will:

- Understand how declarative filter definitions drive UI component generation
- Know how to configure different filter types (autocomplete, range, multiselect, text)
- Recognize the pattern of separating filter metadata from filter state

---

Objective

Create the automobile filter definitions array that tells the Query Panel component which filters to display and how to configure them. These definitions are pure data — they describe what filters exist, not how to render them.

Why

In traditional applications, you might create a separate component for each filter: ManufacturerFilterComponent, YearRangeFilterComponent, BodyClassFilterComponent. This approach leads to:

- **Code duplication** — Each filter component has similar structure (label, input, validation)
- **Tight coupling** — Adding a new filter requires writing new component code
- **Inconsistent UX** — Different developers implement filters differently

The configuration-driven approach solves these problems:

- **Single component, multiple instances** — One QueryPanelComponent renders all filters
- **Loose coupling** — Adding a new filter means adding a new object to an array
- **Consistent UX** — All filters use the same rendering logic

This is the Phase 6 "Aha Moment": Configuration is declarative code. You describe what you want, not how to get it. The filter definitions say "I need an autocomplete for manufacturer with these options" — they don't say "create an input element, attach a keyup listener, debounce for 300ms, call the API..."

Angular Style Guide References

- Style 03-01: Use consistent naming for symbols
- Configuration objects are a recognized Angular pattern for customizing component behavior

---

What

Step 601.1: Create the Filter Definitions File

Create the file that will contain all automobile filter definitions.

Create `src/app/domain-config/automobile/configs/automobile.filter-definitions.ts`:

```
// src/app/domain-config/automobile/configs/automobile.filter-definitions.ts

/**


  • Automobile Domain - Filter Definitions

 *

  • Defines query control filters for the automobile discovery interface.


  • These are UI filters that users can interact with to refine their search.

 *

  • Domain: Automobile Discovery

 */

import { FilterDefinition } from '../../../framework/models/filter-
definition.interface';


/**


  • Automobile filter definitions

 *

  • Array of filter controls for the query panel.


  • Users can combine these filters to search for specific vehicles.
```

```
 *
  • @example
  •
```

typescript

-
- <app-filter-control

ngFor="let filter of AUTOMOBILE_FILTER_DEFINITIONS"

- [definition]="filter"
- [(value)]="filterValues[filter.id]">
- </app-filter-control>
-

```
 /
export const AUTOMOBILE_FILTER_DEFINITIONS: FilterDefinition[] = [ /**



  • Manufacturer filter


   */
{ id: 'manufacturer', label: 'Manufacturer', type: 'autocomplete', placeholder: 'Enter
manufacturer name...', autocompleteEndpoint: 'filters/manufacturers',
autocompleteMinChars: 1, operators: ['contains', 'equals', 'startsWith'],
defaultOperator: 'contains', validation: { minLength: 1, maxLength: 100 } },



/**



  • Model filter


   *


  • Uses autocomplete with progressive refinement:



  • - User types 2+ characters



  • - Backend returns top 10 matching models



  • - Results narrow as user types more


   */
{ id: 'model', label: 'Model', type: 'autocomplete', placeholder: 'Type to search
models...', autocompleteEndpoint: 'filters/models', autocompleteMinChars: 1,
operators: ['contains', 'equals', 'startsWith'], defaultOperator: 'contains',
validation: { minLength: 1, maxLength: 100 } },
```

```
/**


  • Year range filter

   *

  • Uses format.number.useGrouping: false to prevent thousand separators



  • (displays "1980" instead of "1,980")

   *

  • Note: id is 'year' so Query Panel looks for 'yearMin'/'yearMax' in currentFilters,



  • which matches the actual filter model field names (AutoSearchFilters.yearMin/
    yearMax)

   */
{ id: 'year', label: 'Year Range', type: 'range', min: 1900, max: new
Date().getFullYear() + 1, // Include next year for upcoming models step: 1, format:
{ number: { useGrouping: false, // No commas in years minimumFractionDigits: 0,
maximumFractionDigits: 0 } } },



/**


  • Body class filter (multi-select)

   *

  • Uses optionsEndpoint to load options dynamically from the backend.



  • This ensures all body classes in the data are available as options.
```

```
  • Allows selecting multiple body classes with checkboxes.

   */
{ id: 'bodyClass', label: 'Body Class', type: 'multiselect', placeholder: 'Select body
classes...', format: { caseSensitive: false, // Match "Coupe", "coupe", "COUPE"
equally transform: 'titlecase' // Normalize to "Coupe" format }, optionsEndpoint:
'body_class' // Loads from /api/specs/v1/agg/body_class },

/**

  • Instance count range filter

   *

  • Uses format.number.useGrouping: true to show thousand separators

  • (displays "1,000" instead of "1000")

   */
{ id: 'instanceCountRange', label: 'VIN Count Range', type: 'range', min: 0, max:
10000, step: 1, format: { number: { useGrouping: true, // Show commas for VIN counts
minimumFractionDigits: 0, maximumFractionDigits: 0 } } },

/**

  • Global search filter

   */
{ id: 'search', label: 'Search', type: 'text', placeholder: 'Search manufacturer,
model, or body class...', operators: ['contains'], defaultOperator: 'contains',
validation: { minLength: 1, maxLength: 200 } } ];
```

```
/**


  • Quick filter presets

 *

  • Predefined filter combinations for common searches

 */
export const AUTOMOBILE_QUICK_FILTERS = { /**


  • Recent vehicles (last 5 years)

   */
recent: { label: 'Recent Vehicles', filters: { yearMin: new Date().getFullYear() - 5,
yearMax: new Date().getFullYear() } },


/**


  • Popular vehicles (high instance count)

   */
popular: { label: 'Popular Vehicles', filters: { instanceCountMin: 100 } },


/**


  • Classic vehicles (pre-2000)

   */
```

```
classic: { label: 'Classic Vehicles', filters: { yearMax: 2000 } },

/**

  • SUVs only

   */
suvs: { label: 'SUVs', filters: { bodyClass: 'SUV' } },

/**

  • Trucks only

   */
trucks: { label: 'Trucks', filters: { bodyClass: 'Truck' } },

/**

  • Sedans only

   */
sedans: { label: 'Sedans', filters: { bodyClass: 'Sedan' } } };

/**

  • Filter groups
```

```
 *

  • Organize filters into logical groups for better UX

 */
export const AUTOMOBILE_FILTER_GROUPS = { /**



  • Vehicle identification filters

   */
identification: { label: 'Vehicle Identification', filters: ['manufacturer', 'model',
'bodyClass'] },



/**



  • Time-based filters

   */
temporal: { label: 'Year', filters: ['year'] },



/**



  • Quantity filters

   */
quantity: { label: 'VIN Count', filters: ['instanceCountRange'] },



/**



  • General search
```

```
   */
general: { label: 'General Search', filters: ['search'] } };



/**



  • Filter validation rules


 *


  • Additional validation beyond basic type validation


 */
export const AUTOMOBILE_FILTER_VALIDATION = { /**



  • Validate year range


   */
yearRange: (min: number, max: number): boolean => { if (min && max && min > max)
{ return false; // Min cannot be greater than max } const currentYear = new
Date().getFullYear(); if (min && (min < 1900 || min > currentYear + 1)) { return
false; // Year out of valid range } if (max && (max < 1900 || max > currentYear + 1))
{ return false; // Year out of valid range } return true; },



/**



  • Validate instance count range


   */
instanceCountRange: (min: number, max: number): boolean => { if (min && max && min >
max) { return false; // Min cannot be greater than max } if (min && min < 0) { return
false; // Cannot be negative } if (max && max < 0) { return false; // Cannot be
negative } return true; } };
```

```
```

Step 601.2: Understand the Filter Definition Structure

Each filter definition object follows this pattern:

Property	Required	Description
`id`	Yes	Unique identifier, used as key in filter state
`label`	Yes	Display text shown to users
`type`	Yes	Filter type: 'autocomplete', 'text', 'range', 'multiselect'
`placeholder`	No	Placeholder text for input fields
`operators`	No	Comparison operators for text filters
`validation`	No	Validation rules (minLength, maxLength, pattern)
`format`	No	Number/date formatting options
`optionsEndpoint`	No	API endpoint for multiselect options
`autocompleteEndpoint`	No	API endpoint for autocomplete suggestions

Filter Types:

Type	Use Case	Example
`autocomplete`	Free text with suggestions	Manufacturer, Model
`text`	Free text without suggestions	Search
`range`	Numeric range (min/max)	Year, VIN Count
`multiselect`	Pick from list	Body Class

Step 601.3: Understanding the Quick Filters

Quick filters are predefined filter combinations that users can apply with one click:

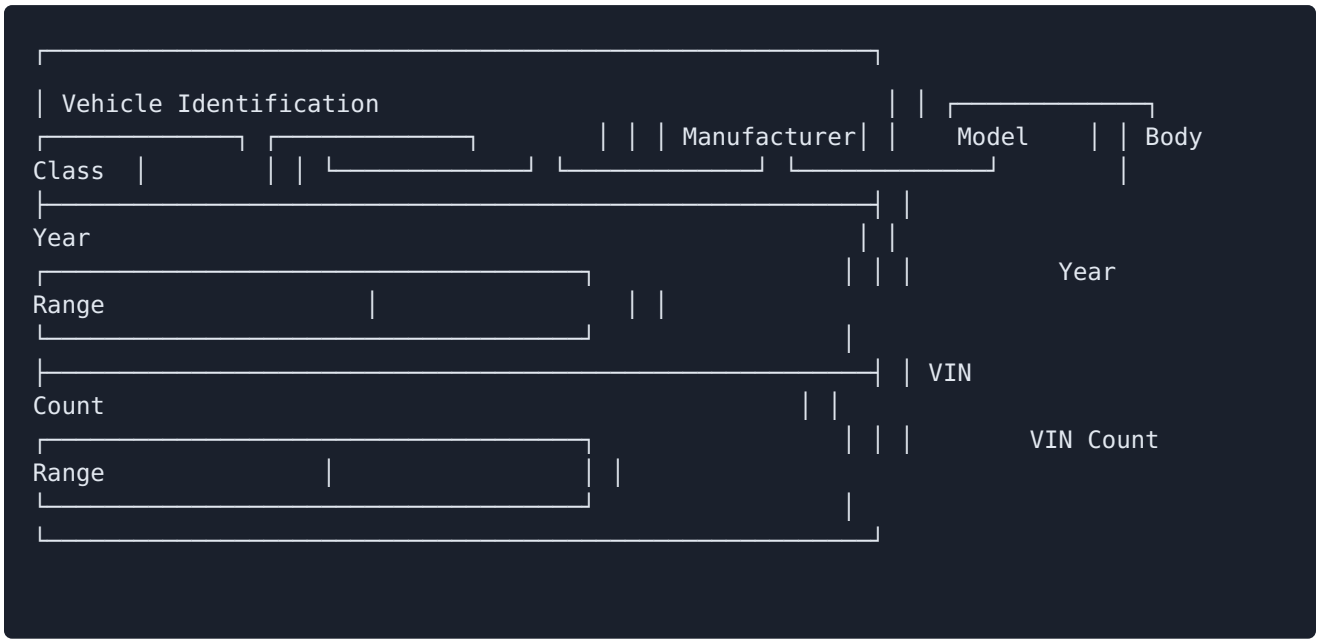
```
// Apply quick filter

applyQuickFilter(quickFilter: { label: string; filters: Partial<AutoSearchFilters> })
{ this.resourceService.updateFilters(quickFilter.filters); }
```

This pattern is useful for:

- Common search patterns (e.g., "Recent Vehicles")
- User onboarding (help new users discover filter capabilities)
- Power user shortcuts (reduce repetitive filter selection)

Step 601.4: Understanding Filter Groups

Filter groups organize related filters in the UI:



Verification

1. Verify File Created

```
$ cat src/app/domain-config/automobile/configs/automobile.filter-definitions.ts | head
-20
```

Expected: First 20 lines of the file shown without errors.

2. Verify TypeScript Compilation

```
$ cd ~/projects/vvroom

$ npx tsc --noEmit src/app/domain-config/automobile/configs/automobile.filter-
definitions.ts
```

Expected: No compilation errors.

3. Verify Export Count

```
$ grep "^export const" src/app/domain-config/automobile/configs/automobile.filter-
definitions.ts | wc -l
```

Expected: `4` (AUTOMOBILE_FILTER_DEFINITIONS, AUTOMOBILE_QUICK_FILTERS, AUTOMOBILE_FILTER_GROUPS, AUTOMOBILE_FILTER_VALIDATION)

Common Problems

Symptom	Cause	Solution
"Cannot find module '../../../framework/models/filter-definition.interface'"	Framework models not yet created	Ensure Phase 2 (document 203) is complete
TypeScript error on `FilterDefinition[]`	Interface doesn't match object shape	Verify filter objects match interface properties
`autocompleteEndpoint` unused	No autocomplete component consuming it	This is expected; components come in Phase 8
Year shows as "2,024"	`useGrouping: true` in year format	Set `useGrouping: false` for year filters

---

Key Takeaways

- **Configuration is data, not code** — Filter definitions are objects in an array, not component classes
- **The framework interprets configuration** — QueryPanelComponent reads definitions and generates UI
- **Adding filters is additive** — To add a new filter, add an object to the array; no component code needed

---

Acceptance Criteria

- [ ] `src/app/domain-config/automobile/configs/automobile.filter-definitions.ts` exists
- [ ] `AUTOMOBILE_FILTER_DEFINITIONS` array contains 6 filter definitions
- [ ] Each filter has `id`, `label`, and `type` properties
- [ ] `AUTOMOBILE_QUICK_FILTERS` contains 6 presets (recent, popular, classic, suvs, trucks, sedans)
- [ ] `AUTOMOBILE_FILTER_GROUPS` organizes filters into 4 groups
- [ ] `AUTOMOBILE_FILTER_VALIDATION` contains validation functions for year and instance count ranges
- [ ] File compiles without TypeScript errors

Next Step

Proceed to `602-table-config.md` to define the automobile results table configuration.

602: Table Config

Status: Planning **Depends On:** 402-automobile-data-model, 204-table-config-interface **Blocks:** 607-domain-config-assembly

---

Learning Objectives

After completing this section, you will:

- Understand how table configuration separates data structure from presentation
- Know how to define columns with sorting, filtering, and width properties
- Recognize the pattern of lazy loading for large datasets

Objective

Create the automobile table configuration that defines how vehicle results are displayed in the data table. This configuration controls columns, pagination, sorting, row expansion, and state persistence.

Why

Data tables are one of the most common UI patterns in enterprise applications. Without configuration, you might hard-code table structure:

```
<!-- Hard-coded table (anti-pattern) -->

<table> <tr> <th>Manufacturer</th> <th>Model</th> <th>Year</th> </tr> <tr *ngFor="let v of vehicles"> <td>{{ v.manufacturer }}</td> <td>{{ v.model }}</td> <td>{{ v.year }} </td> </tr> </table>
```

This approach has problems:

- **No reusability** — Every domain needs its own table template
- **No flexibility** — Adding columns requires template changes
- **Missing features** — Sorting, filtering, pagination require custom code

The configuration-driven approach:

```
<app-results-table
[config]="AUTOMOBILE_TABLE_CONFIG" [data]="vehicles"> </app-results-table>
```

One generic component, unlimited configurations.

Angular Style Guide References

- Style 05-14: Put logic in services, not components
- Configuration objects follow this principle by moving table structure to data

---

What

Step 602.1: Create the Table Configuration File

Create the file that will define the automobile results table.

Create `src/app/domain-config/automobile/configs/automobile.table-config.ts`:

```
// src/app/domain-config/automobile/configs/automobile.table-config.ts

/**


  • Automobile Domain - Table Configuration

 *

  • Defines the main data table for displaying vehicle results.



  • Configures columns, pagination, sorting, filtering, and row expansion.

 *

  • Domain: Automobile Discovery

 */

import { TableConfig } from '../../../framework/models/table-config.interface'; import
{ VehicleResult } from '../models/automobile.data';



/**


  • Automobile table configuration

 *

  • Main table for displaying vehicle search results.



  • Supports pagination, sorting, filtering, and row expansion for VIN details.
```

```
*

 • @example



 •
```

typescript

- <p-table
- [value]="vehicles"
- [columns]="AUTOMOBILE_TABLE_CONFIG.columns"
- [dataKey]="AUTOMOBILE_TABLE_CONFIG.dataKey"
- [stateStorage]="AUTOMOBILE_TABLE_CONFIG.stateStorage"
- [stateKey]="AUTOMOBILE_TABLE_CONFIG.stateKey">
- </p-table>

```
 /
export const AUTOMOBILE_TABLE_CONFIG: TableConfig<VehicleResult> = { /**


  • Unique table identifier

   */
tableId: 'automobile-vehicles-table',


/**


  • State persistence key


  • Saves column widths, order, visibility, filters, sorting, pagination

   */
stateKey: 'auto-vehicles-state',


/**


  • Data key field (must be unique per row)


  • Used for row expansion, selection, and state management

   */
dataKey: 'vehicle_id',


/**
```

```
  • Table columns configuration

   */
columns: [ { field: 'manufacturer', header: 'Manufacturer', sortable: true,
filterable: true, filterMatchMode: 'contains', reorderable: true, width: '150px' },
{ field: 'model', header: 'Model', sortable: true, filterable: true, filterMatchMode:
'contains', reorderable: true, width: '150px' }, { field: 'year', header: 'Year',
sortable: true, filterable: true, filterMatchMode: 'equals', reorderable: true, width:
'100px' }, { field: 'body_class', header: 'Body Class', sortable: true, filterable:
true, filterMatchMode: 'contains', reorderable: true, width: '120px' }, { field:
'instance_count', header: 'VIN Count', sortable: true, filterable: false, reorderable:
true, width: '100px' } ],



/**



  • Row expansion enabled



  • Clicking expand button shows VIN instances for the vehicle

   */
expandable: true,



/**



  • Row selection disabled



  • Enable if you need multi-select functionality

   */
selectable: false,
```

```
/**

  • Selection mode (if selectable=true)

   */
selectionMode: undefined,

/**

  • Pagination enabled

   */
paginator: true,

/**

  • Default rows per page

   */
rows: 20,

/**

  • Rows per page options

   */
rowsPerPageOptions: [10, 20, 50, 100],
```

```
/**

  • Lazy loading enabled

  • Data fetched on demand (pagination, sorting, filtering)

   */
lazy: true,

/**

  • State persistence

  • Saves table state to localStorage

   */
stateStorage: 'local',

/**

  • Table style class

  • PrimeNG classes for styling

   */
styleClass: 'p-datatable-striped p-datatable-gridlines',
```

```
/**

  • Responsive layout

   */
responsiveLayout: 'scroll',

/**

  • Show grid lines

   */
gridlines: true,

/**

  • Striped rows

   */
stripedRows: true,

/**

  • Loading indicator

   */
loading: false };
```

```
/**


  • Column visibility presets

 *

  • Predefined column visibility configurations for different use cases

 */
export const AUTOMOBILE_TABLE_COLUMN_PRESETS = { /**


  • All columns visible

   */
all: AUTOMOBILE_TABLE_CONFIG.columns,


/**


  • Minimal view (core fields only)

   */
minimal: AUTOMOBILE_TABLE_CONFIG.columns.filter((col) => ['manufacturer', 'model',
'year', 'instance_count'].includes(col.field) ),


/**


  • Summary view (no VIN count)

   */
```

```
summary: AUTOMOBILE_TABLE_CONFIG.columns.filter( (col) => col.field !==
'instance_count' ) };



/**



  • Default sort configuration


 */

export const AUTOMOBILE_TABLE_DEFAULT_SORT = { field: 'manufacturer', order: 1 // 1 =
ascending, -1 = descending };



/**



  • Export format configurations


 *

  • Define which columns to include in exports


 */

export const AUTOMOBILE_TABLE_EXPORT_CONFIG = { /**



  • CSV export column configuration


   */

csv: { columns: [ { field: 'manufacturer', header: 'Manufacturer' }, { field: 'model',
header: 'Model' }, { field: 'year', header: 'Year' }, { field: 'body_class', header:
'Body Class' }, { field: 'instance_count', header: 'VIN Count' } ], filename:
'automobile-vehicles' },



/**
```

```
  • Excel export column configuration


   */
excel: { columns: [ { field: 'vehicle_id', header: 'Vehicle ID' }, { field:
'manufacturer', header: 'Manufacturer' }, { field: 'model', header: 'Model' },
{ field: 'year', header: 'Year' }, { field: 'body_class', header: 'Body Class' },
{ field: 'instance_count', header: 'VIN Count' }, { field: 'first_seen', header:
'First Seen' }, { field: 'last_seen', header: 'Last Seen' } ], filename: 'automobile-
vehicles', sheetName: 'Vehicles' } };
```

Step 602.2: Understand the Column Configuration

Each column definition controls how that column behaves:

Property	Type	Description
`field`	`string`	Property name in data object (e.g., 'manufacturer')
`header`	`string`	Display text in column header (e.g., 'Manufacturer')
`sortable`	`boolean`	Enable column sorting
`filterable`	`boolean`	Enable column filtering
`filterMatchMode`	`string`	Filter comparison: 'contains', 'equals', 'startsWith'
`reorderable`	`boolean`	Allow drag-and-drop column reordering
`width`	`string`	Column width (CSS units)

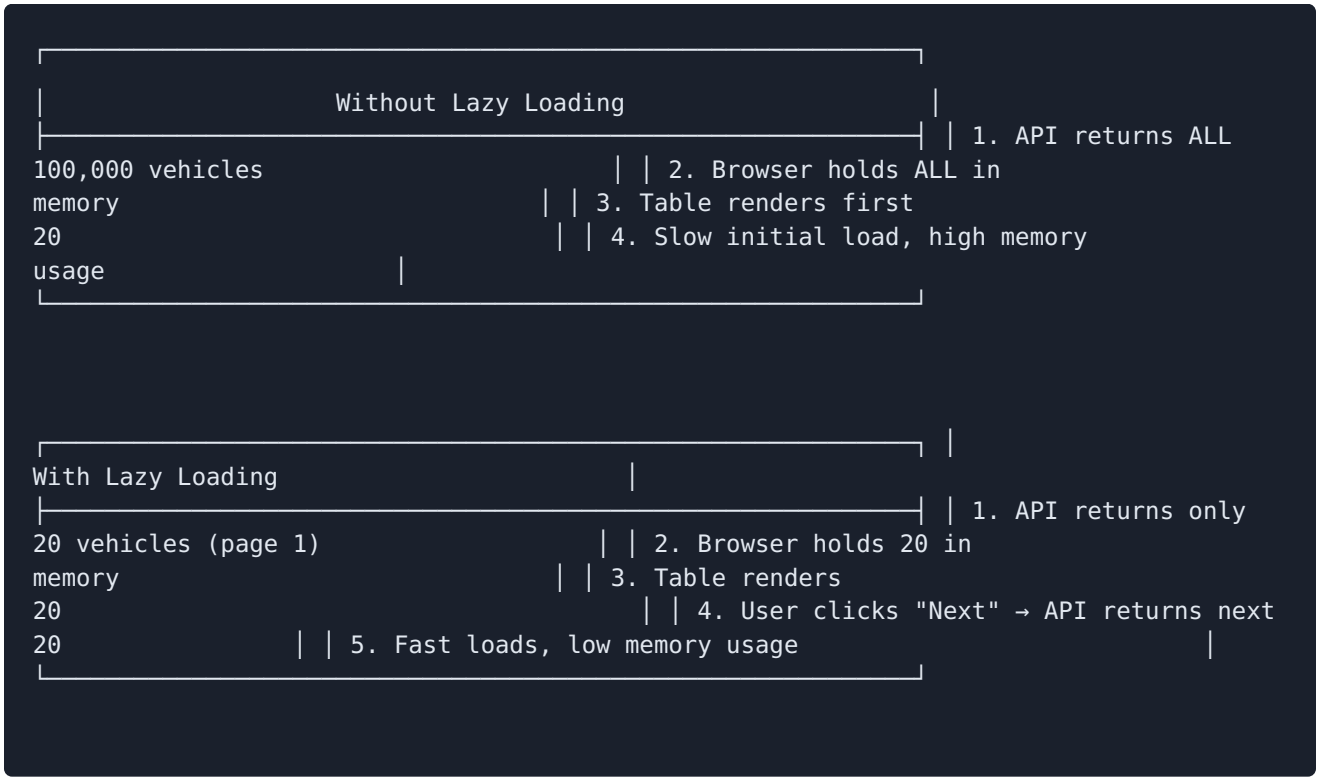
Field Mapping:

The `field` property maps to the `VehicleResult` interface:

```
interface VehicleResult {

vehicle_id: string;     // field: 'vehicle_id' manufacturer: string;   // field:
'manufacturer' model: string;          // field: 'model' year: number;           //
field: 'year' body_class: string;     // field: 'body_class' instance_count:
number; // field: 'instance_count' }
```

Step 602.3: Understanding Lazy Loading

The `lazy: true` setting is critical for performance with large datasets:



When `lazy: true`:

- The table emits `onLazyLoad` events with pagination/sort/filter parameters
- Your service calls the API with these parameters
- The API returns only the requested page

Step 602.4: Understanding State Persistence

The `stateStorage: 'local'` and `stateKey: 'auto-vehicles-state'` settings save table state:

What gets saved:

- Column widths (after user resizes)
- Column order (after user reorders)
- Sort column and direction
- Current page number
- Rows per page selection
- Column filters

Where it's saved:

- `localStorage` with key `auto-vehicles-state`

When it's restored:

- On page reload, the table returns to the user's last state

This provides a personalized experience without requiring server-side storage.

Step 602.5: Understanding Row Expansion

The `expandable: true` and `dataKey: 'vehicle_id'` settings enable row expansion:

```
┌──────────────┬──────────┬──────┬────────────┬──────────┐
│ Manufacturer │ Model    │ Year │ Body Class │ VIN Count │
├──────────────┼──────────┼──────┼────────────┼──────────┤ │ ▶ Toyota       │
Camry      │ 2023 │ Sedan      │ 1,234     │
├──────────────┼──────────┼──────┼────────────┼──────────┤ │ ▼ Honda        │
Civic      │ 2022 │ Sedan      │ 987       │ │
┌──────────────────────────────────────────────────┐  │ │   │ VIN Instances (987
total)                           │  │ │   │
┌──────────────────────────────────────────────────┐ │  │ │   │ │ 1HGBH41JXMN109186 │
First Seen: 2022-01-15      │ │  │ │   │ │ 1HGBH41JXMN109187 │ First Seen:
2022-01-16      │ │  │ │   │ │ 1HGBH41JXMN109188 │ First Seen: 2022-01-17      │ │  │
│   │ └──────────────────────────────────────────────────┘ │  │ │
└──────────────────────────────────────────────────┘  │
├──────────────┼──────────┼──────┼────────────┼──────────┤ │ ▶ Ford         │
F-150      │ 2023 │ Truck      │ 2,345     │
└──────────────┴──────────┴──────┴────────────┴──────────┘
```

The `dataKey` identifies which row is expanded. Without a unique key, expansion state would be lost on sort/filter.

Verification

1. Verify File Created

```
$ ls -la src/app/domain-config/automobile/configs/automobile.table-config.ts
```

Expected: File exists with correct size.

2. Verify TypeScript Compilation

```
$ cd ~/projects/vvroom
$ npx tsc --noEmit src/app/domain-config/automobile/configs/automobile.table-config.ts
```

Expected: No compilation errors.

3. Verify Column Count

```
$ grep -c "field:" src/app/domain-config/automobile/configs/automobile.table-config.ts
```

Expected: `13` (5 table columns + 5 CSV columns + 8 Excel columns - some overlap)

4. Verify Exports

```
$ grep "^export const" src/app/domain-config/automobile/configs/automobile.table-config.ts | wc -l
```

Expected: `4` (AUTOMOBILE_TABLE_CONFIG, AUTOMOBILE_TABLE_COLUMN_PRESETS, AUTOMOBILE_TABLE_DEFAULT_SORT, AUTOMOBILE_TABLE_EXPORT_CONFIG)

Common Problems

Symptom	Cause	Solution
"Cannot find module '../models/automobile.data'"	Data model not yet created	Ensure Phase 4 (document 402) is complete
"Property 'vehicle_id' does not exist on type 'VehicleResult'"	Field name mismatch	Verify field names match VehicleResult interface
Table shows "No records found"	Lazy loading not wired up	Ensure component calls API on onLazyLoad events
Column widths not persisting	State key conflict	Use unique stateKey per table
Sort not working	Missing sortable: true	Add sortable: true to column definition

---

Key Takeaways

- **Table configuration is data** — Column definitions, pagination, and sorting are all configuration objects
- **Lazy loading is essential for large datasets** — Only fetch what the user can see

- **State persistence provides personalization** — Users don't lose their column preferences on reload

---

Acceptance Criteria

- [ ] `src/app/domain-config/automobile/configs/automobile.table-config.ts` exists
- [ ] `AUTOMOBILE_TABLE_CONFIG` has 5 column definitions (manufacturer, model, year, body_class, instance_count)
- [ ] `lazy: true` is set for lazy loading
- [ ] `stateStorage: 'local'` and `stateKey` are configured for state persistence
- [ ] `expandable: true` and `dataKey: 'vehicle_id'` enable row expansion
- [ ] `AUTOMOBILE_TABLE_COLUMN_PRESETS` provides 3 presets (all, minimal, summary)
- [ ] `AUTOMOBILE_TABLE_EXPORT_CONFIG` defines CSV and Excel export formats
- [ ] File compiles without TypeScript errors

Next Step

Proceed to `603-picker-configs.md` to define the automobile picker configurations for selecting manufacturer-model combinations.

603: Picker Configs

Status: Planning **Depends On:** 205-picker-config-interface, 302-api-service **Blocks:** 607-domain-config-assembly

---

Learning Objectives

After completing this section, you will:

- Understand the picker pattern for selecting related data from searchable tables
- Know how to configure server-side pagination for large option sets
- Recognize URL serialization patterns for complex multi-value selections

---

Objective

Create the automobile picker configuration for selecting manufacturer-model combinations. Pickers are searchable, paginated tables that allow users to select one or more items from a large dataset.

Why

Simple dropdowns work for small option sets (10-50 items). But what happens when you have:

- 2,000+ manufacturers
- 50,000+ manufacturer-model combinations
- Options that change frequently (new models added)

Loading all options into a dropdown is impractical:

```
<!-- Anti-pattern: Loading 50,000 options into a dropdown -->

<select> <option *ngFor="let combo of allCombinations"> {{ combo.manufacturer }}:
{{ combo.model }} </option> </select>
```

Problems:

- **Slow initial load** — Fetching 50,000 items takes seconds
- **Poor UX** — Scrolling through 50,000 items is unusable
- **Memory bloat** — 50,000 DOM elements consume significant memory

The picker pattern solves this:

```
┌──────────────────────────────────────────────────────────────────────┐
│ Select Manufacturer & Model                                    [X]   │
├──────────────────────────────────────────────────────────────────────┤ │ [Search: Toyota
Cam_____________________]                          │
├──────────────────────────────────────────────────────────────────────┤ │ [ ]
Manufacturer          Model                  Count          │
├──────────────────────────────────────────────────────────────────────┤ │ [x]
Toyota                Camry                  1,234          │ │ [x]
Toyota                Camry Hybrid           567            │ │ [ ]
Toyota                Camry Solara           89             │ │ [ ]
Toyota                Camry XLE              234            │
├──────────────────────────────────────────────────────────────────────┤ │ Showing 1-20 of
47 results        [<] [1] [2] [3] [>]            │
├──────────────────────────────────────────────────────────────────────┤
│                              [Cancel]  [Apply 2 Selected]          │
└──────────────────────────────────────────────────────────────────────┘
```

Benefits:

- **Server-side pagination** — Only 20 items loaded at a time
- **Search** — Type to filter without loading all options
- **Multi-select** — Check multiple items before applying

Angular Style Guide References

- Style 05-14: Put complex logic in services

- Picker configuration keeps selection logic in configuration, not component code

---

What

Step 603.1: Create the Picker Configurations File

Create the file that will define automobile picker configurations.

Create `src/app/domain-config/automobile/configs/automobile.picker-configs.ts`:

```
// src/app/domain-config/automobile/configs/automobile.picker-configs.ts

/**


  • Automobile Domain - Picker Configurations

 *

  • Defines picker components for selecting related data.



  • Pickers are searchable tables with multi-select capabilities.

 *

  • Domain: Automobile Discovery

 */

import { PickerConfig } from '../../../framework/models/picker-config.interface';
import { Injector } from '@angular/core'; import { ApiService } from '../../../
framework/services'; import { environment } from '../../../environments/environment';



/**


  • Manufacturer/Model row data type

 *

  • Represents a manufacturer-model combination with count

 */
export interface ManufacturerModelRow { /**
```

```
  • Manufacturer name (e.g., "Toyota", "Honda", "BMW")

   */
manufacturer: string;


/**

  • Vehicle model name (e.g., "Camry", "Civic", "X5")

   */
model: string;


/**

  • Number of vehicles matching this manufacturer-model combination

   */
count: number; }


/**

  • Create Manufacturer-Model Picker Configuration

 *

  • Factory function that creates picker config with injected dependencies.

  • Allows selection of manufacturer-model combinations for filtering.
```

```
 *

  • @param apiService - Injected API service



  • @param configId - Unique config ID for this picker instance



  • @returns Configured picker

 */
export function createManufacturerModelPickerConfig( apiService: ApiService, configId:
string = 'manufacturer-model-picker' ): PickerConfig<ManufacturerModelRow> { return
{ id: configId, displayName: 'Select Manufacturer & Model',



// Column definitions (PrimeNGColumn format) columns: [ { field: 'manufacturer',
header: 'Manufacturer', sortable: true, filterable: true, width: '40%' }, { field:
'model', header: 'Model', sortable: true, filterable: true, width: '40%' }, { field:
'count', header: 'Count', sortable: true, filterable: false, width: '20%' } ],



// API configuration // Server-side pagination: API returns { data, total, page, size,
totalPages } api: { fetchData: (params) => { const endpoint = `
{environment.apiBaseUrl}/manufacturer-model-combinations`; return
apiService.get<any>(endpoint, { params: { page: params.page + 1, // API is 1-indexed,
picker is 0-indexed size: params.size, search: params.search || undefined, sortBy:
params.sortField || 'manufacturer', sortOrder: params.sortOrder === -1 ? 'desc' :
'asc' } }); },



responseTransformer: (response) => { return { results: response.data || [], total:
response.total || 0, page: response.page, size: response.size, totalPages:
response.totalPages }; } },



// Row key configuration row: { keyGenerator: (row) => `${row.manufacturer}:${
{row.model}`, keyParser: (key) => { const [manufacturer, model] = key.split(':');
return { manufacturer, model } as Partial<ManufacturerModelRow>; } },
```

```
// Selection configuration selection: { mode: 'multiple', urlParam: 'modelCombos',



// Serialize selected items to URL serializer: (items) => { if (!items || items.length
=== 0) return ''; return items.map(item => ${item.manufacturer}:$
{item.model}).join(','); },



// Deserialize URL to partial items (for hydration) deserializer: (urlValue) => { if
(!urlValue) return []; return urlValue.split(',').map(combo => { const [manufacturer,
model] = combo.split(':'); return { manufacturer, model } as
Partial<ManufacturerModelRow>; }); },



// Optional: Custom key generator (defaults to row.keyGenerator) keyGenerator: (item)
=> ${item.manufacturer}:${item.model} },



// Pagination configuration // Server-side pagination: fetches only current page from
API pagination: { mode: 'server', defaultPageSize: 20, pageSizeOptions: [10, 20, 50,
100] },



// Search configuration showSearch: true, searchPlaceholder: 'Search manufacturer or
model...' }; }



/**



  • Register all automobile picker configurations

 *

  • @param injector - Angular injector for dependency resolution
```

```
  • @param configIdPrefix - Optional prefix to make config IDs unique per page



  • @returns Array of picker configurations

 */
export function createAutomobilePickerConfigs(injector: Injector, configIdPrefix?:
string): PickerConfig<any>[] { const apiService = injector.get(ApiService); const
pickerId = configIdPrefix ? ${configIdPrefix}-manufacturer-model-picker :
'manufacturer-model-picker';



return [ createManufacturerModelPickerConfig(apiService, pickerId) // Add more pickers
here as needed ]; }



/**



  • Static export for backwards compatibility



  • Populated dynamically by domain config factory

 */
export const AUTOMOBILE_PICKER_CONFIGS: PickerConfig<any>[] = [];
```

Step 603.2: Understand the Picker Configuration Structure

Each picker configuration has several key sections:

Identity:

```
{
id: 'manufacturer-model-picker', displayName: 'Select Manufacturer & Model' }
```

Column Definitions:

```
columns: [
{ field: 'manufacturer', header: 'Manufacturer', sortable: true, width: '40%' } ]
```

API Configuration:

```
api: {
fetchData: (params) => { / Call API with page, size, search, sort / },
responseTransformer: (response) => { / Normalize response shape / } }
```

Row Key Generation:

```
row: {
keyGenerator: (row) => ${row.manufacturer}:${row.model},  // Create unique key
keyParser: (key) => { / Parse key back to partial object / } }
```

Selection Serialization:

```
selection: {
serializer: (items) => / Convert selected items to URL string /, deserializer:
(urlValue) => / Convert URL string back to items / }
```

Step 603.3: Understanding URL Serialization

Selected items must be serialized to the URL for URL-First architecture:

```
Before selection: /automobiles/discover

After selection:  /automobiles/discover?modelCombos=Toyota:Camry,Honda:Civic
```

The serialization flow:

Step 603.4: Understanding Server-Side Pagination

The picker uses server-side pagination:

```
api: {

fetchData: (params) => { return apiService.get(endpoint, { params: { page: params.page
+ 1,  // Convert 0-indexed to 1-indexed size: params.size,       // e.g., 20 search:
params.search,   // User's search term sortBy: params.sortField, sortOrder:
params.sortOrder === -1 ? 'desc' : 'asc' } }); } }
```

When user searches "Toyota":

- Picker sends: `GET /manufacturer-model-combinations?page=1&size=20&search=Toyota`
- API returns: `{ data: [/ 20 Toyota results /], total: 47, page: 1, size: 20, totalPages: 3 }`
- Picker displays 20 results with pagination: "Showing 1-20 of 47"

Step 603.5: Understanding Factory Pattern

The picker uses a factory function pattern:

```
export function createManufacturerModelPickerConfig(

apiService: ApiService, configId: string = 'manufacturer-model-picker' ):
PickerConfig<ManufacturerModelRow> { return { / config / }; }
```

Why a factory function instead of a constant?

The picker needs the `ApiService` to make API calls. Angular's dependency injection provides the service at runtime, but configuration constants are created at module load time (before injection is available).

The factory pattern solves this:

```
// In domain config factory

export function createAutomobileDomainConfig(injector: Injector) { const apiService =
injector.get(ApiService);



return { // ... pickers: createAutomobilePickerConfigs(injector) }; }
```

Verification

1. Verify File Created

```
$ ls -la src/app/domain-config/automobile/configs/automobile.picker-configs.ts
```

Expected: File exists.

2. Verify TypeScript Compilation

```
$ cd ~/projects/vvroom

$ npx tsc --noEmit src/app/domain-config/automobile/configs/automobile.picker-
configs.ts
```

Expected: No compilation errors.

3. Verify Interface Export

```
$ grep "^export interface" src/app/domain-config/automobile/configs/automobile.picker-
configs.ts
```

Expected: `export interface ManufacturerModelRow {`

4. Verify Function Exports

```
$ grep "^export function" src/app/domain-config/automobile/configs/automobile.picker-configs.ts
```

Expected: Two function exports (createManufacturerModelPickerConfig, createAutomobilePickerConfigs)

Common Problems

Symptom	Cause	Solution
"Cannot find module '../../../framework/services'"	Services not yet created	Ensure Phase 3 is complete
"Property 'get' does not exist on type 'ApiService'"	ApiService interface mismatch	Verify ApiService has get<T>() method
Picker shows empty	API endpoint not responding	Check API URL and network tab
Selection not persisting to URL	serializer/deserializer mismatch	Ensure serializer output matches deserializer input
Page numbers wrong	0-indexed vs 1-indexed mismatch	Picker is 0-indexed, API is 1-indexed; convert

---

Key Takeaways

- **Pickers handle large option sets** — Server-side pagination and search make 50,000+ items usable
- **URL serialization enables sharing** — Selected items are in the URL, so users can share links
- **Factory pattern enables dependency injection** — Services are injected at runtime, not import time

---

Acceptance Criteria

- [ ] `src/app/domain-config/automobile/configs/automobile.picker-configs.ts` exists

- [ ] `ManufacturerModelRow` interface defines manufacturer, model, and count properties
- [ ] `createManufacturerModelPickerConfig()` factory function returns picker configuration
- [ ] Picker configuration includes 3 columns (manufacturer, model, count)
- [ ] `api.fetchData` function handles pagination, search, and sorting parameters
- [ ] `selection.serializer` converts items to URL string format "Manufacturer:Model,..."
- [ ] `selection.deserializer` parses URL string back to partial items
- [ ] `createAutomobilePickerConfigs()` returns array with manufacturer-model picker
- [ ] File compiles without TypeScript errors

Next Step

Proceed to `604-query-control-filters.md` to define the query control filter definitions for dialog-based filter editing.

604: Query Control Filters

Status: Planning **Depends On:** 203-filter-definition-interface, 401-automobile-filters-model **Blocks:** 607-domain-config-assembly

---

Learning Objectives

After completing this section, you will:

- Understand how query control filters differ from simple filter definitions
- Know how to configure API endpoints for fetching filter options dynamically
- Recognize the pattern of transforming API responses into filter option formats

---

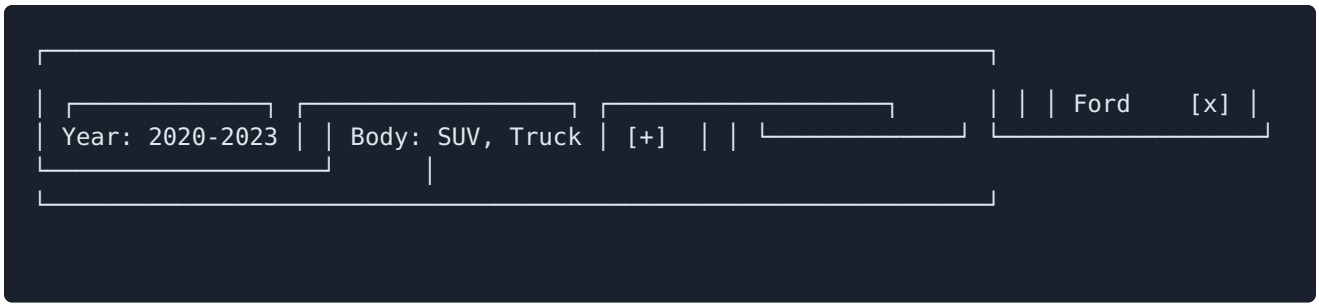
Objective

Create the query control filter definitions that power the "Add Filter" dialog. These definitions tell the QueryControlComponent which filters are available, how to fetch their options, and how to serialize selections to the URL.

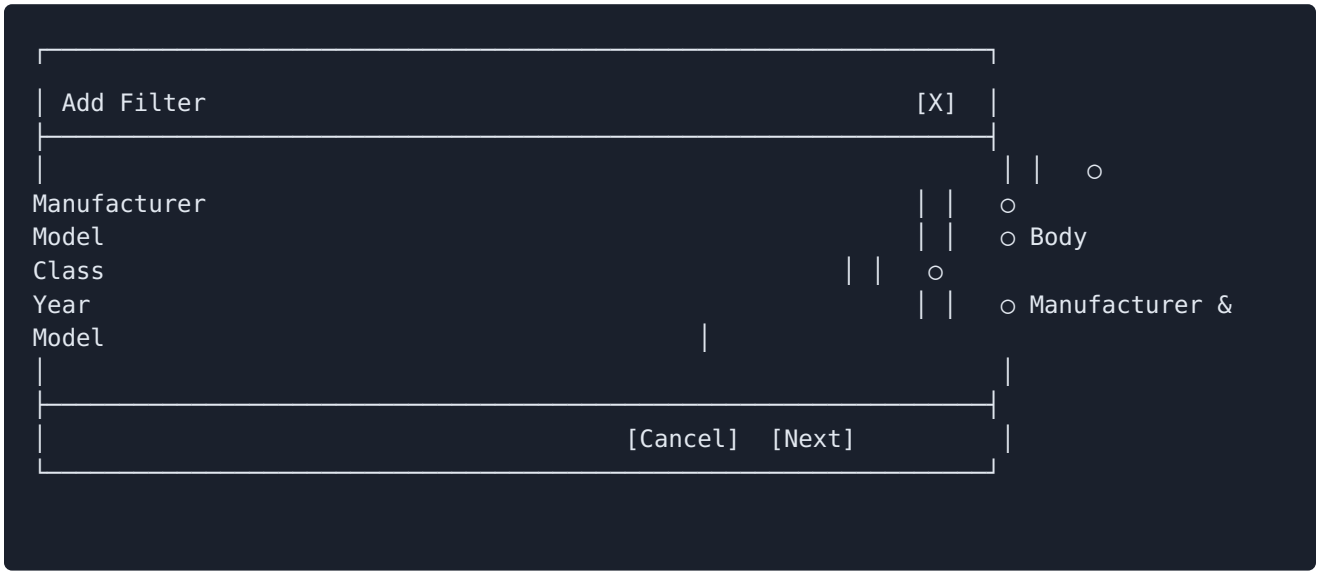
Why

Document 601 defined filter definitions for the Query Panel (always-visible filter controls). But what about filters that users add on demand?

The QueryControlComponent provides a "chip-based" filter UI:



Clicking [+] opens a dialog:



Selecting "Manufacturer" and clicking "Next" opens the filter value dialog:

Query control filters define:

- Which filters appear in the "Add Filter" list
- Where to fetch options for each filter
- How to transform API responses into option format
- How to serialize/deserialize values to/from URL

What

Step 604.1: Create the Query Control Filters File

Create the file that will define query control filter configurations.

Create `src/app/domain-config/automobile/configs/automobile.query-control-filters.ts`:

```
// src/app/domain-config/automobile/configs/automobile.query-control-filters.ts

/**


  • Automobile Domain - Query Control Filter Definitions

 *

  • Defines filter definitions for the Query Control component.


  • These filters allow users to manually add/remove/edit filters via dialogs.

 *

  • Domain: Automobile Discovery

 */

import { FilterDefinition, FilterOption } from '../../../framework/models/filter-
definition.interface'; import { AutoSearchFilters } from '../models/
automobile.filters'; import { environment } from '../../../environments/environment';



/**


  • Query Control filter definitions

 *

  • Each definition specifies:


  • - Which field it filters
```

```
  • - What type of UI control to display (multiselect, range, etc.)



  • - Where to fetch options from (API endpoint)



  • - How to transform API responses



  • - URL parameter names

 *

  • These are used by the QueryControlComponent to dynamically render filter dialogs.

 */
export const AUTOMOBILE_QUERY_CONTROL_FILTERS: FilterDefinition<AutoSearchFilters>[] =
[ /**



  • Manufacturer filter (Multiselect)

   */
{ field: 'manufacturer', label: 'Manufacturer', type: 'multiselect', optionsEndpoint:
 ${environment.apiBaseUrl}/filters/manufacturers , optionsTransformer: (response:
any): FilterOption[] => { if (response && response.manufacturers) { return
response.manufacturers.map((m: string) => ({ value: m, label: m })); } return []; },
urlParams: 'manufacturer', searchPlaceholder: 'Type to search manufacturers...',
dialogSubtitle: 'Select one or more manufacturers to filter results.' },



/**



  • Model filter (Multiselect)

   */
```

```
{ field: 'model', label: 'Model', type: 'multiselect', optionsEndpoint: ${environment.apiBaseUrl}/filters/models, optionsTransformer: (response: any): FilterOption[] => { if (response && response.models) { return response.models.map((m: string) => ({ value: m, label: m })); } return []; }, urlParams: 'model', searchPlaceholder: 'Type to search models...', dialogSubtitle: 'Select one or more models to filter results.' },



/**



  • Body Class filter (Multiselect)


   */

{ field: 'bodyClass', label: 'Body Class', type: 'multiselect', optionsEndpoint: ${environment.apiBaseUrl}/filters/body-classes, optionsTransformer: (response: any): FilterOption[] => { if (response && response.body_classes) { return response.body_classes.map((b: string) => ({ value: b, label: b })); } return []; }, urlParams: 'bodyClass', searchPlaceholder: 'Type to search body classes...', dialogSubtitle: 'Select one or more body classes to filter results.' },



/**



  • Year Range filter (Range)


   *


  • Note: This uses yearMin as the field, but actually manages both yearMin and
    yearMax


   */

{ field: 'yearMin', label: 'Year', type: 'range', optionsEndpoint: ${environment.apiBaseUrl}/filters/year-range, urlParams: { min: 'yearMin', max: 'yearMax' }, dialogTitle: 'Select Year Range', dialogSubtitle: 'Select a year range to filter results. You can select just a start year, end year, or both.', rangeConfig: { valueType: 'integer', minLabel: 'Start Year', maxLabel: 'End Year', minPlaceholder: 'e.g., 1980', maxPlaceholder: 'e.g., 2023', step: 1, useGrouping: false, defaultRange: { min: 1900, max: new Date().getFullYear() } } },
```

```
/**


  • Manufacturer-Model Combinations filter (Multiselect)

   *

  • Used to display chips when selections are made via Manufacturer-Model Picker



  • Format: "Manufacturer:Model" (e.g., "Ford:F-150")

   */
{ field: 'modelCombos', label: 'Manufacturer & Model', type: 'multiselect',
optionsEndpoint: ${environment.apiBaseUrl}/manufacturer-model-combinations?
page=1&size=100, optionsTransformer: (response: any): FilterOption[] => { //
Response structure: { total, data: [ { manufacturer, count, models: [ { model,
count } ] } ] } if (response && response.data) { const options: FilterOption[] = [];
for (const mfr of response.data) { for (const modelObj of mfr.models || [])
{ options.push({ value: ${mfr.manufacturer}:${modelObj.model}, label: $
{mfr.manufacturer}: ${modelObj.model} }); } } return options; } return []; },
urlParams: 'modelCombos', searchPlaceholder: 'Type to search manufacturer-model
combinations...', dialogSubtitle: 'Select one or more manufacturer-model combinations.
Tip: Use the Manufacturer-Model Picker panel for easier selection.' } ];
```

Step 604.2: Understand the Query Control Filter Structure

Each query control filter definition has these key properties:

Property	Type	Description
`field`	`string`	Property name in AutoSearchFilters
`label`	`string`	Display text in filter list and dialog title
`type`	`'multiselect' \`	'range'	Filter UI type
`optionsEndpoint`	`string`	API endpoint to fetch options
`optionsTransformer`	`function`	Transform API response to FilterOption[]
`urlParams`	`string \`	object	URL parameter name(s) for this filter
`searchPlaceholder`	`string`	Placeholder text in search input
`dialogSubtitle`	`string`	Help text shown in filter dialog

For Range Filters:

Property	Type	Description
`rangeConfig.valueType`	`'integer' \`	'float'	Numeric type
`rangeConfig.minLabel`	`string`	Label for min input
`rangeConfig.maxLabel`	`string`	Label for max input
`rangeConfig.step`	`number`	Input step value
`rangeConfig.useGrouping`	`boolean`	Use thousand separators
`rangeConfig.defaultRange`	`{ min, max }`	Default range limits

Step 604.3: Understanding the Options Transformer

The `optionsTransformer` function converts API responses to the standard `FilterOption[]` format:

```
// API Response (raw)
{ manufacturers: ["Ford", "Toyota", "Honda", "BMW"] }



// After optionsTransformer [ { value: "Ford", label: "Ford" }, { value: "Toyota",
label: "Toyota" }, { value: "Honda", label: "Honda" }, { value: "BMW", label:
"BMW" } ]
```

Why this pattern?

Different APIs return data in different formats:

- Some return arrays of strings
- Some return arrays of objects with id/name
- Some return nested structures

The transformer normalizes all formats to `{ value, label }`.

Complex Example (Model Combinations):

```
// API Response (raw)
{ data: [ { manufacturer: "Ford", count: 100, models: [ { model: "F-150", count: 50 },
{ model: "Mustang", count: 30 } ] }, { manufacturer: "Toyota", count: 80, models:
[ { model: "Camry", count: 40 } ] } ] }



// After optionsTransformer [ { value: "Ford:F-150", label: "Ford: F-150" }, { value:
"Ford:Mustang", label: "Ford: Mustang" }, { value: "Toyota:Camry", label: "Toyota:
Camry" } ]
```

Step 604.4: Understanding URL Parameter Mapping

The `urlParams` property defines how filter values appear in the URL:

Simple (string):

```
urlParams: 'manufacturer'
// Selected: ["Ford", "Toyota"] // URL: ?manufacturer=Ford,Toyota
```

Range (object):

```
urlParams: { min: 'yearMin', max: 'yearMax' }
// Selected: { min: 2020, max: 2023 } // URL: ?yearMin=2020&yearMax=2023
```

This separation allows:

- Filter field names to differ from URL parameter names
- Range filters to use two separate URL parameters

Step 604.5: Understanding Filter Type Differences

Multiselect vs Range:

```
┌──────────────────────────────────────────────────────────────────┐
│                        MULTISELECT                               │
├──────────────────────────────────────────────────────────────────┤ │ • User selects 0
to N items from a list                          │ │ • Options fetched from
API                                          │ │ • URL: ?
manufacturer=Ford,Toyota,Honda                              │ │ • Chip: "Manufacturer:
Ford, Toyota, Honda"                        │
└──────────────────────────────────────────────────────────────────┘




┌──────────────────────────────────────────────────────────────────┐
│                          RANGE                                   │
├──────────────────────────────────────────────────────────────────┤ │ • User enters
min and/or max values                                 │ │ • No options fetched (user
types values)                               │ │ • URL: ?
yearMin=2020&yearMax=2023                                   │ │ • Chip: "Year: 2020 -
2023"                                        │
└──────────────────────────────────────────────────────────────────┘
```

Verification

1. Verify File Created

```
$ ls -la src/app/domain-config/automobile/configs/automobile.query-control-filters.ts
```

Expected: File exists.

2. Verify TypeScript Compilation

```
$ cd ~/projects/vvroom

$ npx tsc --noEmit src/app/domain-config/automobile/configs/automobile.query-control-
filters.ts
```

Expected: No compilation errors.

3. Verify Filter Count

```
$ grep -c "field:" src/app/domain-config/automobile/configs/automobile.query-control-filters.ts
```

Expected: `5` (manufacturer, model, bodyClass, yearMin, modelCombos)

4. Verify Exports

```
$ grep "^export const" src/app/domain-config/automobile/configs/automobile.query-control-filters.ts
```

Expected: `export const AUTOMOBILE_QUERY_CONTROL_FILTERS`

Common Problems

Symptom	Cause	Solution
"Cannot find module '../models/automobile.filters'"	Filters model not yet created	Ensure Phase 4 (document 401) is complete
Options not loading	API endpoint incorrect	Verify optionsEndpoint matches actual API
Transformer returns empty array	API response structure changed	Update optionsTransformer to match actual response
Chip shows "[object Object]"	Label not set correctly	Ensure optionsTransformer sets label property
Year filter shows commas	useGrouping not set to false	Set rangeConfig.useGrouping: false

---

Key Takeaways

- **Query control filters are on-demand** — Users add them via dialog, unlike always-visible filter panel
- **Options are fetched dynamically** — optionsEndpoint and optionsTransformer handle API integration

- **URL params can differ from field names** — urlParams provides the URL-side mapping

---

Acceptance Criteria

- [ ] `src/app/domain-config/automobile/configs/automobile.query-control-filters.ts` exists
- [ ] `AUTOMOBILE_QUERY_CONTROL_FILTERS` array contains 5 filter definitions
- [ ] Manufacturer, Model, Body Class filters are type 'multiselect'
- [ ] Year filter is type 'range' with rangeConfig
- [ ] Model Combinations filter handles nested API response structure
- [ ] Each filter has optionsEndpoint and optionsTransformer
- [ ] Each filter has urlParams for URL serialization
- [ ] File compiles without TypeScript errors

Next Step

Proceed to `605-highlight-filters.md` to define highlight filter definitions for chart segmentation.

605: Highlight Filters

Status: Planning **Depends On:** 203-filter-definition-interface, 401-automobile-filters-model **Blocks:** 607-domain-config-assembly

---

Learning Objectives

After completing this section, you will:

- Understand the purpose of highlight filters for chart segmentation
- Know the naming convention for highlight URL parameters (h_ prefix)
- Recognize how highlight data flows from URL to API to chart visualization

---

Objective

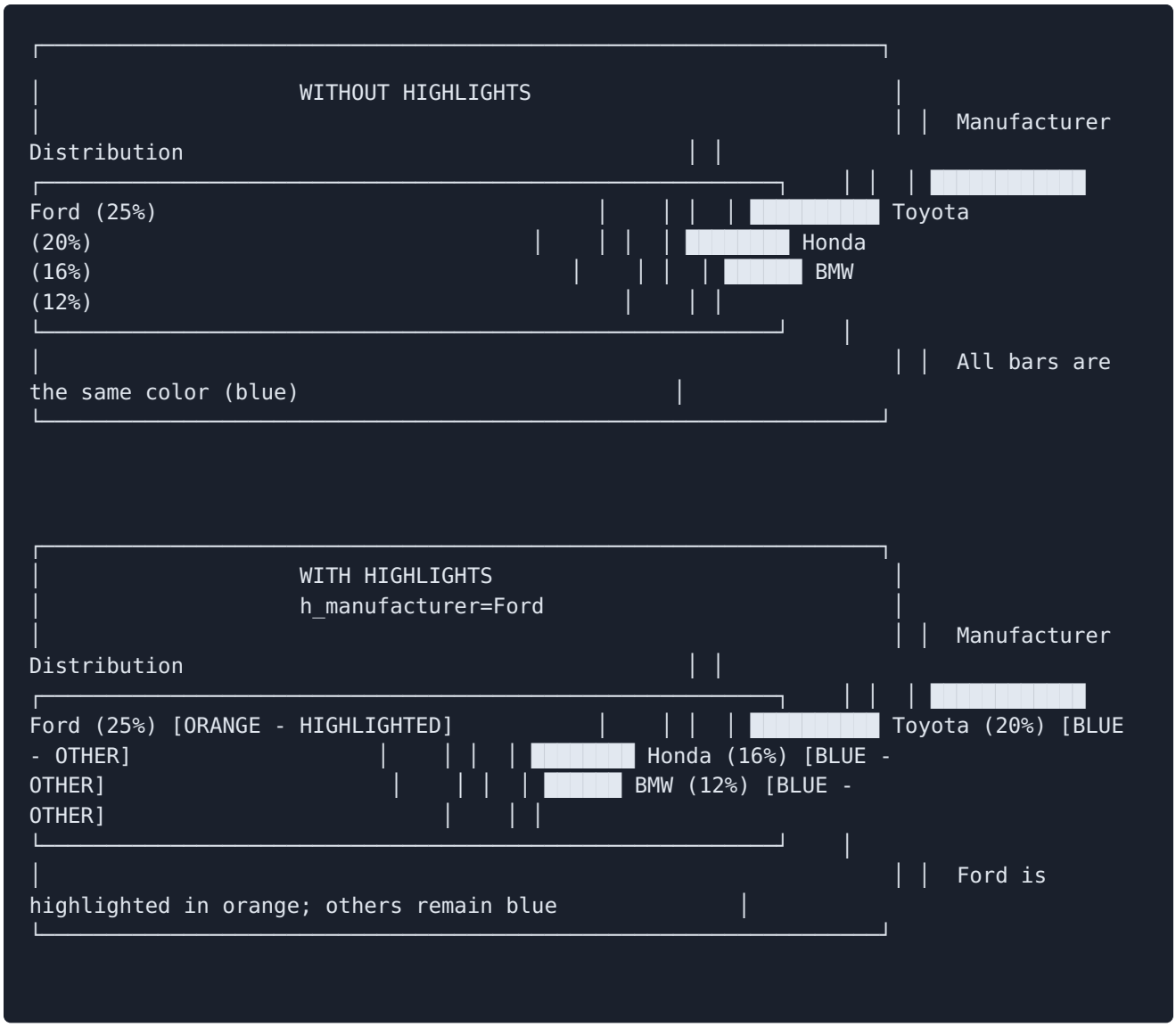
Create the highlight filter definitions that enable chart segmentation. Highlight filters use the same structure as query control filters but generate URL parameters with the `h_` prefix, triggering special API behavior that returns segmented statistics.

Why

Regular filters narrow results: "Show only Ford vehicles" removes all non-Ford data.

Highlight filters segment results: "Highlight Ford in the charts" keeps all data but colors Ford differently.

Visual comparison:



Use cases:

- **Competitive analysis** — Highlight your company vs competitors
- **Focus attention** — Draw user's eye to specific data points
- **Before/after comparison** — Highlight a year range to show trends

API behavior:

When the API receives highlight parameters, it returns segmented statistics:

```
// Without highlights: GET /vehicles/statistics

{ "manufacturers": [ { "name": "Ford", "count": 1000 }, { "name": "Toyota", "count":
800 } ] }



// With highlights: GET /vehicles/statistics?h_manufacturer=Ford { "manufacturers":
[ { "name": "Ford", "count": 1000, "highlighted": 1000 }, { "name": "Toyota", "count":
800, "highlighted": 0 } ] }
```

The `highlighted` field indicates how much of the count matches the highlight criteria.

What

Step 605.1: Create the Highlight Filters File

Create the file that will define highlight filter configurations.

Create `src/app/domain-config/automobile/configs/automobile.highlight-filters.ts`:

```
// src/app/domain-config/automobile/configs/automobile.highlight-filters.ts

/**


  • Automobile Domain - Highlight Filter Definitions

 *

  • Defines highlight filter definitions for the Query Control component.

  Highlight filters allow users to add h_ parameters to segment statistics

  • in charts (showing highlighted vs other data in stacked bars).

 *

  • Domain: Automobile Discovery

 */

import { FilterDefinition, FilterOption } from '../../../framework/models/filter-
definition.interface'; import { HighlightFilters } from '../models/
automobile.filters'; import { environment } from '../../../environments/environment';



/**


  • Highlight filter definitions

 *

  • Each definition specifies:


  • - Which highlight field it manages (h_manufacturer, h_modelCombos, etc.)
```

```
  • - What type of UI control to display (multiselect, range, etc.)


  • - Where to fetch options from (API endpoint)


  • - How to transform API responses


  • - URL parameter names (with h_ prefix)

*

  • These are used by the QueryControlComponent to dynamically render highlight filter
    dialogs.

*

  • @example


  •
```

typescript

- // User adds "Highlight Manufacturer: Ford" filter
- // URL updates to: ?h_manufacturer=Ford
- // API receives: GET /vehicles/details?h_manufacturer=Ford
- // Returns segmented statistics: {"Ford": {"total": 665, "highlighted": 665}, ...}
- // Charts render stacked bars showing highlighted (Ford) vs other manufacturers

```
 /
export const AUTOMOBILE_HIGHLIGHT_FILTERS: FilterDefinition<HighlightFilters>[] = [ /
**



  • Highlight Manufacturer filter (Multiselect)



  • URL parameter: h_manufacturer


   */
{ field: 'manufacturer', label: 'Highlight Manufacturer', type: 'multiselect',
optionsEndpoint: ${environment.apiBaseUrl}/filters/manufacturers,
optionsTransformer: (response: any): FilterOption[] => { if (response &&
response.manufacturers) { return response.manufacturers.map((m: string) => ({ value:
m, label: m })); } return []; }, urlParams: 'h_manufacturer', searchPlaceholder: 'Type
to search manufacturers...', dialogSubtitle: 'Select one or more manufacturers to
highlight in charts.' },



/**



  • Highlight Model Combinations filter (Multiselect)



  • URL parameter: h_modelCombos



  • Format: Manufacturer:Model,Manufacturer:Model (e.g., Ford:F-150,Toyota:Camry)


   */
{ field: 'modelCombos', label: 'Highlight Models', type: 'multiselect',
optionsEndpoint: ${environment.apiBaseUrl}/manufacturer-model-combinations,
optionsTransformer: (response: any): FilterOption[] => { const options: FilterOption[]
= []; if (response && response.data) { // Flatten nested structure:
data[].manufacturer + data[].models[].model for (const manufacturerGroup of
response.data) { const manufacturer = manufacturerGroup.manufacturer; if
(manufacturerGroup.models) { for (const modelObj of manufacturerGroup.models) { const
```

```
model = modelObj.model; options.push({ value: ${manufacturer}:${model}, label: ${manufacturer}:${model} }); } } } } return options; }, urlParams: 'h_modelCombos', searchPlaceholder: 'Type to search model combinations...', dialogSubtitle: 'Select one or more model combinations to highlight in charts.' },



/**



  • Highlight Body Class filter (Multiselect)



  • URL parameter: h_bodyClass

   */
{ field: 'bodyClass', label: 'Highlight Body Class', type: 'multiselect', optionsEndpoint: ${environment.apiBaseUrl}/filters/body-classes, optionsTransformer: (response: any): FilterOption[] => { if (response && response.body_classes) { return response.body_classes.map((b: string) => ({ value: b, label: b })); } return []; }, urlParams: 'h_bodyClass', searchPlaceholder: 'Type to search body classes...', dialogSubtitle: 'Select one or more body classes to highlight in charts.' },



/**



  • Highlight Year Range filter (Range)



  • URL parameters: h_yearMin, h_yearMax

   *

  • Note: This uses yearMin as the field, but actually manages both yearMin and
    yearMax

   */
```

```
{ field: 'yearMin', label: 'Highlight Year', type: 'range', optionsEndpoint: ${environment.apiBaseUrl}/filters/year-range, urlParams: { min: 'h_yearMin', max: 'h_yearMax' }, dialogTitle: 'Highlight Year Range', dialogSubtitle: 'Select a year range to highlight in charts. You can select just a start year, end year, or both.', rangeConfig: { valueType: 'integer', minLabel: 'Start Year', maxLabel: 'End Year', minPlaceholder: 'e.g., 1980', maxPlaceholder: 'e.g., 2023', step: 1, useGrouping: false, defaultRange: { min: 1900, max: new Date().getFullYear() } } } ];
```

Step 605.2: Understand the Highlight Filter Structure

Highlight filters use the same structure as query control filters with one key difference: the `urlParams` values have the `h_` prefix.

Query Control Filter:

```
{
field: 'manufacturer', urlParams: 'manufacturer'  // URL: ?manufacturer=Ford }
```

Highlight Filter:

```
{
field: 'manufacturer', urlParams: 'h_manufacturer'  // URL: ?h_manufacturer=Ford }
```

The `h_` prefix signals to the API adapter that this is a highlight filter, not a regular filter.

Step 605.3: Understanding the URL Parameter Convention

The `h_` prefix is a convention used throughout the application:

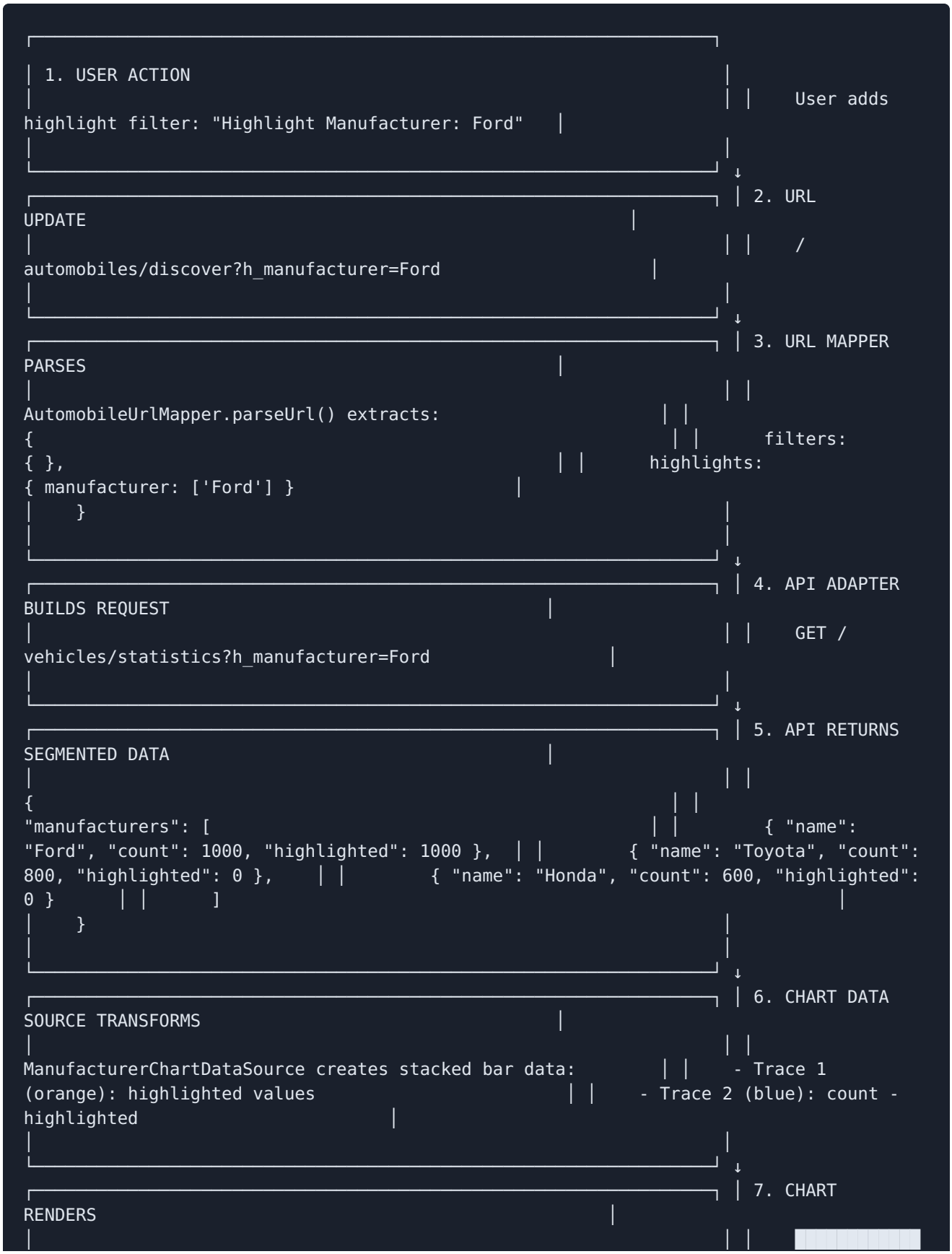
Regular Filter	Highlight Filter	Effect	
`manufacturer=Ford`	`h_manufacturer=Ford`	Filter: Only Ford	Highlight: All data, Ford colored
`yearMin=2020&yearMax=2023`	`h_yearMin=2020&h_yearMax=2023`	Filter: Only 2020-2023	Highlight: All years, 2020-2023 colored
`bodyClass=SUV`	`h_bodyClass=SUV`	Filter: Only SUVs	Highlight: All body classes, SUV colored

Combined example:

```
?manufacturer=Ford&h_bodyClass=SUV

Result:

  • Data filtered to Ford vehicles only

  • Charts show all Ford body classes

  • SUV bars highlighted in orange, other Ford body classes in blue
```

Step 605.4: Understanding the Data Flow

The complete flow from URL to visualization:


1. USER ACTION
User adds highlight filter: "Highlight Manufacturer: Ford"
2. URL UPDATE
/automobiles/discover?h_manufacturer=Ford
3. URL MAPPER PARSES
AutomobileUrlMapper.parseUrl() extracts:
{ filters: { }, highlights: { manufacturer: ['Ford'] } }
4. API ADAPTER BUILDS REQUEST
GET /vehicles/statistics?h_manufacturer=Ford
5. API RETURNS SEGMENTED DATA
{ "manufacturers": [{ "name": "Ford", "count": 1000, "highlighted": 1000 }, { "name": "Toyota", "count": 800, "highlighted": 0 }, { "name": "Honda", "count": 600, "highlighted": 0 }] }
6. CHART DATA SOURCE TRANSFORMS
ManufacturerChartDataSource creates stacked bar data:
- Trace 1 (orange): highlighted values
- Trace 2 (blue): count - highlighted
7. CHART RENDERS

Step 605.5: Comparing Query Control Filters vs Highlight Filters

Document 604 (Query Control Filters):

- Narrow the dataset
- Remove non-matching rows
- URL params: `manufacturer`, `model`, `yearMin`, etc.

Document 605 (Highlight Filters):

- Segment the visualization
- Keep all rows, color matching rows differently
- URL params: `h_manufacturer`, `h_model`, `h_yearMin`, etc.

They share:

- Same structure (FilterDefinition interface)
- Same options endpoints
- Same options transformers
- Same UI dialogs (QueryControlComponent handles both)

They differ:

- URL parameter names (with/without `h_` prefix)
- API behavior (filter vs segment)
- Visual effect (hide vs color)

Verification

1. Verify File Created

```
$ ls -la src/app/domain-config/automobile/configs/automobile.highlight-filters.ts
```

Expected: File exists.

2. Verify TypeScript Compilation

```
$ cd ~/projects/vvroom

$ npx tsc --noEmit src/app/domain-config/automobile/configs/automobile.highlight-filters.ts
```

Expected: No compilation errors.

3. Verify Filter Count

```
$ grep -c "field:" src/app/domain-config/automobile/configs/automobile.highlight-filters.ts
```

Expected: `4` (manufacturer, modelCombos, bodyClass, yearMin)

4. Verify h_ Prefix

```
$ grep "h_" src/app/domain-config/automobile/configs/automobile.highlight-filters.ts | wc -l
```

Expected: At least 6 occurrences (h_manufacturer, h_modelCombos, h_bodyClass, h_yearMin, h_yearMax in code and comments)

Common Problems

Symptom	Cause	Solution
Highlights not appearing in charts	API not returning highlighted field	Verify API supports h_ parameters
Same color for highlighted and non-highlighted	Chart data source not using highlighted field	Update chart data source to check for segmented data
Filter and highlight interfering	Using wrong prefix	Filter params no prefix, highlight params h_ prefix
"Cannot find 'HighlightFilters'"	Model not imported	Ensure automobile.filters.ts exports HighlightFilters

---

Key Takeaways

- **Highlight filters segment, they don't filter** — All data remains; selected items are visually distinguished
- **The h_ prefix is the convention** — URL parameters starting with h_ trigger highlight behavior
- **Same structure, different purpose** — Highlight filters reuse the FilterDefinition interface with different urlParams

---

Acceptance Criteria

- [ ] `src/app/domain-config/automobile/configs/automobile.highlight-filters.ts` exists
- [ ] `AUTOMOBILE_HIGHLIGHT_FILTERS` array contains 4 filter definitions
- [ ] All urlParams values start with `h_` prefix
- [ ] Manufacturer and Body Class filters are type 'multiselect'
- [ ] Model Combinations filter handles nested API response
- [ ] Year filter is type 'range' with h_yearMin and h_yearMax params
- [ ] Label text includes "Highlight" to distinguish from regular filters
- [ ] File compiles without TypeScript errors

Next Step

Proceed to `606-chart-configs.md` to define the chart configurations for the statistics panel.

606: Chart Configs

Status: Planning **Depends On:** 201-domain-config-interface **Blocks:** 607-domain-config-assembly, 651-manufacturer-chart-source

---

Learning Objectives

After completing this section, you will:

- Understand how chart configurations separate visualization metadata from data transformation
- Know the relationship between chart configs and chart data sources
- Recognize the pattern of declarative chart definition

Objective

Create the automobile chart configurations that define which charts appear in the statistics panel. These configurations specify chart identity, type, dimensions, and which data source transforms the statistics into chart data.

Why

Visualizations help users understand data patterns. For automobiles, we want charts showing:

- **Manufacturer distribution** — Which manufacturers have the most vehicles?
- **Model distribution** — Which specific models are most common?
- **Year distribution** — How are vehicles distributed across years?
- **Body class distribution** — What types of vehicles are in the data?

Without configuration, you might create each chart as a separate component:

```
@Component({

template:  <div class="chart-container"> <h3>Manufacturers</h3> <plotly-plot
[data]="manufacturerData" [layout]="layout"></plotly-plot> </div>

}) export class ManufacturerChartComponent { // Transform statistics to chart data //
Handle Plotly configuration // Manage visibility // ...50+ lines of code }
```

Four charts = four components = 200+ lines of similar code.

With configuration:

```
export const AUTOMOBILE_CHART_CONFIGS: ChartConfig[] = [

{ id: 'manufacturer-distribution', title: 'Manufacturers', dataSourceId:
'manufacturer', ... }, { id: 'year-distribution', title: 'Year', dataSourceId:
'year', ... }, // ... ];
```

One generic `BaseChartComponent` + configuration = minimal code.

This is the Phase 6 Aha Moment again: Configuration is declarative code. You describe what charts you want, not how to render them.

Angular Style Guide References

- Configuration objects follow the same pattern as Angular's built-in configurations (routes, providers)

What

Step 606.1: Create the Chart Configurations File

Create the file that will define automobile chart configurations.

Create `src/app/domain-config/automobile/configs/automobile.chart-configs.ts`:

```
// src/app/domain-config/automobile/configs/automobile.chart-configs.ts

/**

  • Automobile Domain - Chart Configurations (Plotly.js)

 *

  • Defines chart visualizations for automobile statistics using Plotly.js.

  • Charts display aggregated data and distributions.

 *

  • Domain: Automobile Discovery

 */

import { ChartConfig } from '../../../framework/models/domain-config.interface';

/**

  • Automobile chart configurations

 *

  • Array of chart definitions for the statistics panel.

  • Each chart visualizes a different aspect of the vehicle data.
```

```
 *
  • NOTE: These configs work with Plotly.js via BaseChartComponent.


  • Data transformation is handled by chart data sources in chart-sources/ directory.
 *
  • @example


  •
```

typescript

- <app-statistics-panel-2 [domainConfig]="domainConfig">
- </app-statistics-panel-2>

```
 /
export const AUTOMOBILE_CHART_CONFIGS: ChartConfig[] = [ /**



  • Manufacturer distribution (vertical stacked bar chart)


   */
{ id: 'manufacturer-distribution', title: 'Manufacturers', type: 'bar', dataSourceId:
'manufacturer', height: 400, width: '100%', visible: true, collapsible: true },



/**



  • Top models by VIN count (vertical stacked bar chart)


   */
{ id: 'top-models', title: 'Models', type: 'bar', dataSourceId: 'top-models', height:
400, width: '100%', visible: true, collapsible: true },



/**



  • Year distribution (vertical stacked bar chart)


   */
{ id: 'year-distribution', title: 'Year', type: 'bar', dataSourceId: 'year', height:
400, width: '100%', visible: true, collapsible: true },



/**



  • Body class distribution (vertical stacked bar chart)


   */
```

```
{ id: 'body-class-distribution', title: 'Body Class', type: 'bar', dataSourceId:
'body-class', height: 400, width: '100%', visible: true, collapsible: true } ];
```

Step 606.2: Understand the Chart Configuration Structure

Each chart configuration defines a visualization:

Property	Type	Description			
`id`	`string`	Unique identifier for the chart			
`title`	`string`	Display title shown above the chart			
`type`	`'bar' \`	'line' \	'pie' \	'scatter'	Plotly chart type
`dataSourceId`	`string`	Key to look up in chartDataSources map			
`height`	`number`	Chart height in pixels			
`width`	`string`	Chart width (CSS value)			
`visible`	`boolean`	Initial visibility state			
`collapsible`	`boolean`	Can user collapse/expand the chart			

Step 606.3: Understanding the Chart/Data Source Relationship

Chart configs and chart data sources are separate concerns:

Chart Config (this file):

- Defines chart metadata (id, title, type, dimensions)
- Says "render a bar chart called 'Manufacturers'"
- Does NOT know how to transform statistics to chart data

Chart Data Source (Phase 7, documents 651-654):

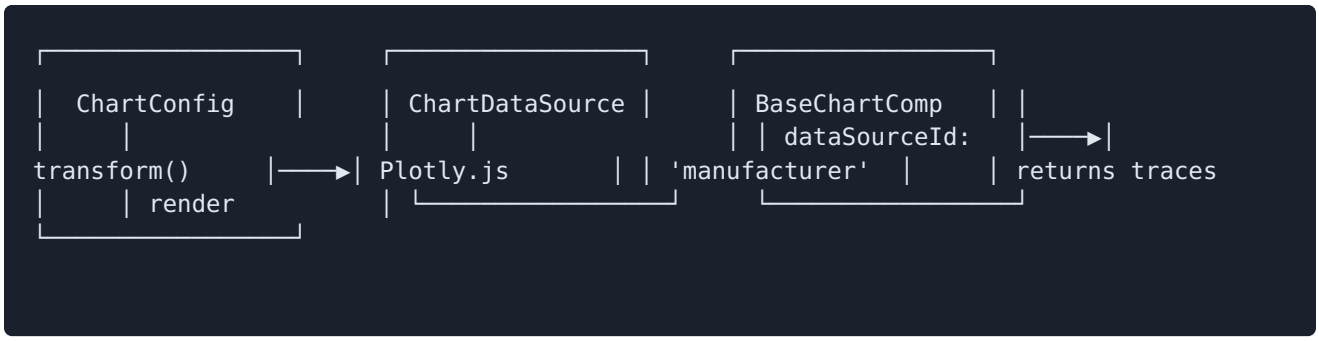
- Transforms statistics into Plotly trace format

- Knows the structure of VehicleStatistics
- Handles highlight segmentation

The connection is the `dataSourceId`:

```
// Chart config (document 606)
{ id: 'manufacturer-distribution', dataSourceId: 'manufacturer', ... }



// Domain config (document 607) connects them chartDataSources: { 'manufacturer': new
ManufacturerChartDataSource(),  // Phase 7 // ... }
```

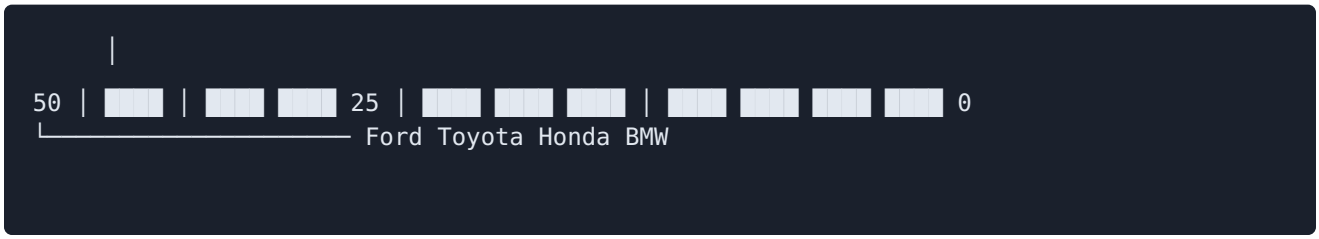
Data Flow:



Step 606.4: Understanding Chart Types

All automobile charts use `type: 'bar'`, but the framework supports multiple types:

Bar Charts (used here):



Best for: Comparing categories (manufacturers, body classes)

Line Charts:

Best for: Showing trends over time

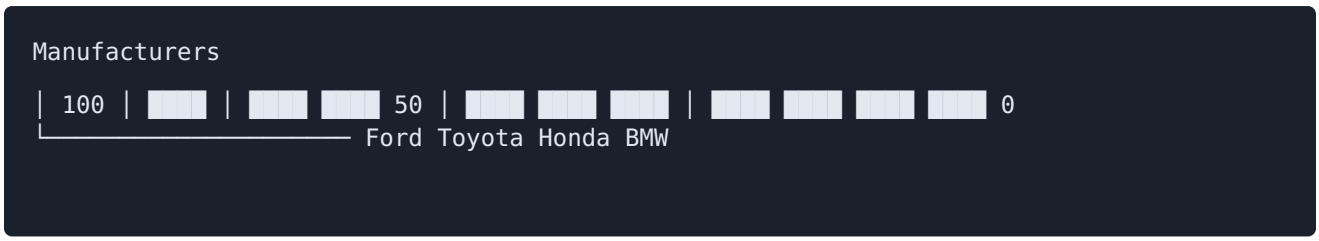
Pie Charts:

Best for: Showing proportions of a whole

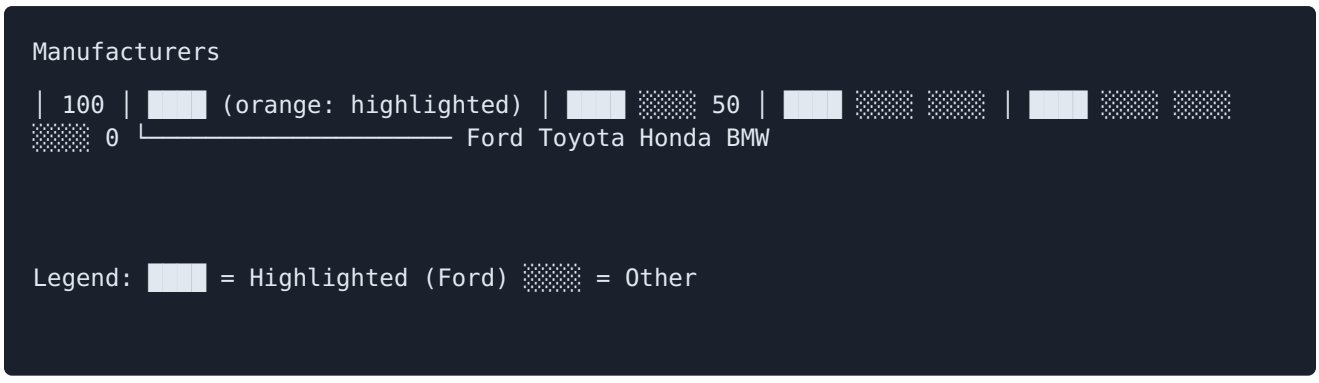
Step 606.5: Understanding Stacked Bar Charts with Highlights

When highlights are active, bar charts become stacked:

Without highlights:



With h_manufacturer=Ford:



The stacking is handled by the chart data source (Phase 7), not the chart config.

Step 606.6: Understanding Chart Dimensions

The `height` and `width` properties control chart sizing:

```
{
height: 400,   // Fixed 400px height width: '100%'  // Responsive width (fills
container) }
```

Why fixed height, responsive width?

- **Fixed height**: Charts need consistent vertical space for readability
- **Responsive width**: Charts should adapt to container width (desktop vs mobile)

The statistics panel uses a grid layout:

```
┌──────────────────────────────────────────────────────────────┐
| Statistics Panel                                             |
├──────────────────────────────┬───────────────────────────────┤ |
┌──────────────────────────────┐ | ┌─────────────────────────────┐| | |
Manufacturers           | | |        Models               || | |     height:
400px           | | |        height: 400px        || | |     width: 100%             |
| |        width: 100%          || | └─────────────────────────────┘ |
└──────────────────────────────┘|
├──────────────────────────────┼───────────────────────────────┤ |
┌──────────────────────────────┐ | ┌─────────────────────────────┐| | |
Year                 | | |      Body Class             || | |     height:
400px           | | |        height: 400px        || | |     width: 100%             |
| |        width: 100%          || | └─────────────────────────────┘ |
└──────────────────────────────┘|
└──────────────────────────────┴───────────────────────────────┘
```

Each chart fills its grid cell horizontally but maintains 400px height.

Step 606.7: Understanding Collapsible Charts

The `collapsible: true` property allows users to minimize charts:

Benefits:

- Users can hide charts they don't need
- Saves vertical space for charts they do use
- State can be persisted in user preferences

Verification

1. Verify File Created

```
$ ls -la src/app/domain-config/automobile/configs/automobile.chart-configs.ts
```

Expected: File exists.

2. Verify TypeScript Compilation

```
$ cd ~/projects/vvroom

$ npx tsc --noEmit src/app/domain-config/automobile/configs/automobile.chart-configs.ts
```

Expected: No compilation errors.

3. Verify Chart Count

```
$ grep -c "id:" src/app/domain-config/automobile/configs/automobile.chart-configs.ts
```

Expected: `4` (manufacturer-distribution, top-models, year-distribution, body-class-distribution)

4. Verify Data Source IDs

```
$ grep "dataSourceId:" src/app/domain-config/automobile/configs/automobile.chart-configs.ts
```

Expected: Four dataSourceId values (manufacturer, top-models, year, body-class)

Common Problems

Symptom	Cause	Solution
"Cannot find module '../../../framework/models/domain-config.interface'"	Interface not yet created	Ensure Phase 2 (document 201) is complete
Chart not rendering	dataSourceId not found in chartDataSources	Verify dataSourceId matches key in domain config
Chart too tall/short	Incorrect height value	Adjust height to appropriate pixel value
Chart not collapsing	collapsible: false	Set collapsible: true
"ChartConfig is not a type"	Import missing	Add import for ChartConfig from domain-config.interface

---

Key Takeaways

- **Chart configs are pure metadata** — They describe what to render, not how
- **Data sources handle transformation** — The dataSourceId links to code that transforms statistics
- **All charts use consistent dimensions** — height: 400, width: '100%' for uniform appearance

---

Acceptance Criteria

- [ ] `src/app/domain-config/automobile/configs/automobile.chart-configs.ts` exists
- [ ] `AUTOMOBILE_CHART_CONFIGS` array contains 4 chart definitions
- [ ] Each chart has id, title, type, dataSourceId, height, width, visible, collapsible
- [ ] All charts use type: 'bar' for consistent visualization
- [ ] dataSourceId values are: manufacturer, top-models, year, body-class
- [ ] All charts have height: 400 and width: '100%'
- [ ] All charts have visible: true and collapsible: true
- [ ] File compiles without TypeScript errors

Next Step

Proceed to `607-domain-config-assembly.md` to assemble all configurations into the complete automobile domain config.

607: Domain Config Assembly

Status: Planning **Depends On:** 201-domain-config-interface, 401-403 (models), 501-503 (adapters), 601-606 (configs) **Blocks:** 608-domain-providers, 902-automobile-landing

---

Learning Objectives

After completing this section, you will:

- Understand how the domain config factory pattern assembles all domain pieces
- Know how Angular's Injector enables runtime dependency resolution
- Recognize the complete structure of a DomainConfig object

Objective

Create the automobile domain configuration factory function that assembles all models, adapters, and UI configurations into a single DomainConfig object. This is the central integration point for everything automobile-related.

Why

Throughout Phase 4, 5, and 6, you created many separate pieces:

Phase 4 (Models):

- AutoSearchFilters
- VehicleResult
- VehicleStatistics

Phase 5 (Adapters):

- AutomobileUrlMapper
- AutomobileApiAdapter
- AutomobileCacheKeyBuilder

Phase 6 (Configs):

- AUTOMOBILE_FILTER_DEFINITIONS
- AUTOMOBILE_TABLE_CONFIG
- AUTOMOBILE_PICKER_CONFIGS (factory)
- AUTOMOBILE_QUERY_CONTROL_FILTERS
- AUTOMOBILE_HIGHLIGHT_FILTERS
- AUTOMOBILE_CHART_CONFIGS

Now you need to assemble these pieces into a single object that the framework can consume.

Why a factory function instead of a constant?

Some pieces need Angular services (like ApiService for the ApiAdapter and pickers). Angular's dependency injection provides services at runtime, not at module load time. A factory function receives the Injector and can resolve dependencies dynamically.

```
// This won't work - ApiService isn't available at import time

export const DOMAIN_CONFIG = { apiAdapter: new AutomobileApiAdapter(apiService) //
Error: apiService undefined };



// This works - Injector provides ApiService at runtime export function
createAutomobileDomainConfig(injector: Injector) { const apiService =
injector.get(ApiService); return { apiAdapter: new
AutomobileApiAdapter(apiService) }; }
```

Angular Style Guide References

- Factory functions are a recognized Angular pattern for complex object creation
- The Provider pattern (`useFactory` with `deps`) is documented in Angular DI guide

---

What

Step 607.1: Create the Domain Config File

Create the file that assembles the complete automobile domain configuration.

Create `src/app/domain-config/automobile/automobile.domain-config.ts`:

```
// src/app/domain-config/automobile/automobile.domain-config.ts

/**



  • Automobile Domain Configuration

 *

  • Complete domain configuration combining all models, adapters, and UI configs



  • from milestones D1-D4.

 */

import { Injector } from '@angular/core'; import { DomainConfig } from '../../
framework/models'; import { ApiService } from '../../framework/services'; import
{ environment } from '../../environments/environment'; import { AutoSearchFilters,
VehicleResult, VehicleStatistics } from './models'; import { AutomobileApiAdapter,
AutomobileUrlMapper, AutomobileCacheKeyBuilder } from './adapters'; import
{ AUTOMOBILE_TABLE_CONFIG, AUTOMOBILE_FILTER_DEFINITIONS,
AUTOMOBILE_QUERY_CONTROL_FILTERS, AUTOMOBILE_HIGHLIGHT_FILTERS,
AUTOMOBILE_CHART_CONFIGS, createAutomobilePickerConfigs } from './configs'; import
{ ManufacturerChartDataSource, TopModelsChartDataSource, BodyClassChartDataSource,
YearChartDataSource } from './chart-sources'; import { Provider } from '@angular/
core'; import { DOMAIN_CONFIG } from '../../framework/services';



/**



  • Factory function to create Automobile Domain Configuration

 *

  • This factory creates the domain configuration with properly injected dependencies.



  • Must be called with Angular's Injector to resolve service dependencies.
```

```
*
 • @param injector - Angular injector for resolving dependencies
 • @returns Complete automobile domain configuration
*
 • @example
 • // In app module
 • providers: [
 •   {
 •     provide: DOMAIN_CONFIG,
 •     useFactory: createAutomobileDomainConfig,
 •     deps: [Injector]
 •   }
 • ]
```

```
 */
export function createAutomobileDomainConfig(injector: Injector): DomainConfig<
AutoSearchFilters, VehicleResult, VehicleStatistics > { const apiService =
injector.get(ApiService); const apiBaseUrl = environment.apiBaseUrl;



return { // ==================== Identity ==================== domainName:
'automobile', domainLabel: 'Automobile Discovery', apiBaseUrl: apiBaseUrl,



// ==================== Type Models ==================== filterModel:
AutoSearchFilters, dataModel: VehicleResult, statisticsModel: VehicleStatistics,



// ==================== Adapters ==================== apiAdapter: new
AutomobileApiAdapter(apiService, apiBaseUrl), urlMapper: new AutomobileUrlMapper(),
cacheKeyBuilder: new AutomobileCacheKeyBuilder(),



// ==================== UI Configuration ==================== tableConfig:
AUTOMOBILE_TABLE_CONFIG, pickers: createAutomobilePickerConfigs(injector), filters:
AUTOMOBILE_FILTER_DEFINITIONS, queryControlFilters: AUTOMOBILE_QUERY_CONTROL_FILTERS,
highlightFilters: AUTOMOBILE_HIGHLIGHT_FILTERS, charts: AUTOMOBILE_CHART_CONFIGS,
chartDataSources: { 'manufacturer': new ManufacturerChartDataSource(), 'top-models':
new TopModelsChartDataSource(), 'body-class': new BodyClassChartDataSource(), 'year':
new YearChartDataSource() },



// ==================== Feature Flags ==================== features: { // Required
features highlights: true, popOuts: true, rowExpansion: true,



// Optional features statistics: true, export: true, columnManagement: true,
statePersistence: true },



// ==================== Metadata ==================== metadata: { version: '1.0.0',
description: 'Automobile vehicle discovery and analysis', author: 'Vvroom Development
Team', createdAt: '2026-02-09', updatedAt: '2026-02-09' } }; }
```

```
/**

  • Angular dependency injection provider for Automobile Domain Configuration

 *

  • Pre-configured provider that can be used directly in Angular module declarations

  • to register the automobile domain configuration with the dependency injection
    container.

 *

  • @constant {Provider} DOMAIN_PROVIDER

  • @remarks

  • This is an Angular Provider object that:

  • - Provides the DOMAIN_CONFIG injection token

  • - Uses a factory function to create the configuration instance

  • - Automatically resolves the Injector dependency

 *

  • Usage in Module:
```

•

typescript

• @NgModule({

• providers: [DOMAIN_PROVIDER] // Add to any module

• })

• export class AppModule { }

```
  • Internally:


  • - provide: Points to the DOMAIN_CONFIG injection token


  • - useFactory: References createAutomobileDomainConfig function


  • - deps: Specifies that Injector should be injected into the factory

 *

  • @see createAutomobileDomainConfig - The factory function that creates the
    configuration


  • @see DomainConfig - The interface describing configuration structure


  • @see DOMAIN_CONFIG - The injection token this provider uses

 */
export const DOMAIN_PROVIDER: Provider = { provide: DOMAIN_CONFIG, useFactory:
createAutomobileDomainConfig, deps: [Injector], };
```

Step 607.2: Create the Barrel Export

Create the index file that exports the domain config.

Create `src/app/domain-config/automobile/index.ts`:

```
// src/app/domain-config/automobile/index.ts

export * from './automobile.domain-config';
```

Step 607.3: Understand the DomainConfig Structure

The complete domain config has several sections:

Identity:

```
{
domainName: 'automobile',      // URL-safe identifier domainLabel: 'Automobile
Discovery',  // Human-readable name apiBaseUrl: 'http://...'       // API base URL
from environment }
```

Type Models:

```
{
filterModel: AutoSearchFilters,    // Class reference for filter type dataModel:
VehicleResult,          // Class reference for data type statisticsModel:
VehicleStatistics // Class reference for stats type }
```

These are class references (not instances) used for TypeScript type checking.

Adapters:

```
{
apiAdapter: new AutomobileApiAdapter(apiService, apiBaseUrl), urlMapper: new
AutomobileUrlMapper(), cacheKeyBuilder: new AutomobileCacheKeyBuilder() }
```

These are instances created with dependencies.

UI Configuration:

```
{
tableConfig: AUTOMOBILE_TABLE_CONFIG, pickers:
createAutomobilePickerConfigs(injector), filters: AUTOMOBILE_FILTER_DEFINITIONS,
queryControlFilters: AUTOMOBILE_QUERY_CONTROL_FILTERS, highlightFilters:
AUTOMOBILE_HIGHLIGHT_FILTERS, charts: AUTOMOBILE_CHART_CONFIGS, chartDataSources:
{ / ... / } }
```

Feature Flags:

```
{
features: { highlights: true,        // Enable highlight filters popOuts:
true,           // Enable pop-out windows rowExpansion: true,      // Enable row
expansion in table statistics: true,        // Enable statistics panel export:
true,            // Enable data export columnManagement: true,  // Enable column
visibility toggle statePersistence: true   // Enable state saving to localStorage } }
```

Metadata:

```
{
metadata: { version: '1.0.0', description: 'Automobile vehicle discovery and
analysis', author: 'Vvroom Development Team', createdAt: '2026-02-09', updatedAt:
'2026-02-09' } }
```

Step 607.4: Understanding the Provider Pattern

The `DOMAIN_PROVIDER` constant is an Angular Provider:

```
export const DOMAIN_PROVIDER: Provider = {
provide: DOMAIN_CONFIG,           // Token to inject useFactory:
createAutomobileDomainConfig,  // Factory function deps:
[Injector],                // Dependencies for factory };
```

This is equivalent to:

```
@NgModule({

providers: [ { provide: DOMAIN_CONFIG, useFactory: createAutomobileDomainConfig, deps:
[Injector] } ] })
```

But exported as a constant for reuse.

How it works:

- Angular sees `provide: DOMAIN_CONFIG`
- Angular calls `createAutomobileDomainConfig(injector)`
- The factory returns the domain config object
- Any component/service can inject `DOMAIN_CONFIG` to get the config

```
// In a component

constructor(@Inject(DOMAIN_CONFIG) private config: DomainConfig<...>)
{ console.log(config.domainName); // 'automobile' }
```

Step 607.5: Understanding Chart Data Sources Mapping

The `chartDataSources` object maps dataSourceId to data source instances:

```
chartDataSources: {

'manufacturer': new ManufacturerChartDataSource(), 'top-models': new
TopModelsChartDataSource(), 'body-class': new BodyClassChartDataSource(), 'year': new
YearChartDataSource() }
```

This connects to chart configs from document 606:

```
// Chart config

{ id: 'manufacturer-distribution', dataSourceId: 'manufacturer', ... }



// Resolution const dataSource = chartDataSources['manufacturer']; const traces =
dataSource.transform(statistics);
```

Note: Chart data sources are implemented in Phase 7 (documents 651-654). This file references them but they don't exist yet. You'll create placeholder classes or add them after Phase 7.

Step 607.6: Understanding Feature Flags

Feature flags enable/disable functionality without code changes:

```
features: {

highlights: true,        // Show highlight filter options popOuts: true,           //
Show "pop out" buttons on panels rowExpansion: true,      // Show expand button on
table rows statistics: true,        // Show statistics panel export:
true,            // Show export buttons columnManagement: true,  // Show column
visibility menu statePersistence: true   // Save/restore table state }
```

Components check these flags:

```
// In StatisticsPanelComponent

<div *ngIf="domainConfig.features.statistics"> <!-- Chart content --> </div>



// In TableComponent <button
*ngIf="domainConfig.features.export" (click)="exportCsv()"> Export CSV </button>
```

This allows:

- A/B testing features
- Disabling features for certain domains

- Progressive feature rollout

Verification

1. Verify Files Created

```
$ ls -la src/app/domain-config/automobile/automobile.domain-config.ts
$ ls -la src/app/domain-config/automobile/index.ts
```

Expected: Both files exist.

2. Verify TypeScript Compilation

```
$ cd ~/projects/vvroom
$ npx tsc --noEmit src/app/domain-config/automobile/automobile.domain-config.ts
```

Expected: No compilation errors.

3. Verify Exports

```
$ grep "^export" src/app/domain-config/automobile/automobile.domain-config.ts
```

Expected: Two exports (createAutomobileDomainConfig function, DOMAIN_PROVIDER constant)

4. Verify Index Exports

```
$ cat src/app/domain-config/automobile/index.ts
```

Expected: `export * from './automobile.domain-config';`

Common Problems

Symptom	Cause	Solution
"Cannot find module './models'"	Models barrel export missing	Create models/index.ts with exports
"Cannot find module './adapters'"	Adapters barrel export missing	Create adapters/index.ts with exports
"Cannot find module './configs'"	Configs barrel export missing	Create configs/index.ts with exports
"Cannot find module './chart-sources'"	Chart sources not yet created	Add after Phase 7 or create placeholder
"DOMAIN_CONFIG is not a type"	Injection token not imported	Import DOMAIN_CONFIG from framework/services
"Injector has no provider for ApiService"	ApiService not provided	Ensure ApiService is in root providers

---

Key Takeaways

- **The domain config is the integration point** — All domain pieces connect through this one object
- **Factory pattern enables dependency injection** — Services are resolved at runtime, not import time
- **Feature flags provide runtime control** — Enable/disable features without code changes

---

Acceptance Criteria

- [ ] `src/app/domain-config/automobile/automobile.domain-config.ts` exists
- [ ] `createAutomobileDomainConfig()` factory function is exported
- [ ] Factory receives Injector and resolves ApiService
- [ ] DomainConfig includes all sections: identity, models, adapters, UI config, features, metadata
- [ ] `DOMAIN_PROVIDER` constant is exported with correct provider shape
- [ ] `src/app/domain-config/automobile/index.ts` exports the domain config

- [ ] Chart data sources are mapped with correct keys matching dataSourceIds
- [ ] Feature flags include all required flags (highlights, popOuts, rowExpansion, statistics, export, columnManagement, statePersistence)
- [ ] File compiles without TypeScript errors

Next Step

Proceed to `608-domain-providers.md` to create the domain providers registry that collects all domain configurations.

608: Domain Providers

Status: Planning **Depends On:** 607-domain-config-assembly **Blocks:** 906-app-config

---

Learning Objectives

After completing this section, you will:

- Understand the purpose of a domain providers registry
- Know how to add new domains to the application
- Recognize the pattern of collecting providers from multiple sources

---

Objective

Create the domain providers file that exports an array of all domain configuration providers. This is the single point where all domains are registered for the application.

Why

The vvroom application is designed to support multiple domains. While we only have automobiles now, the architecture anticipates:

- Agriculture (crops, yields, regions)
- Chemistry (compounds, reactions)
- Physics (particles, experiments)
- Finance (stocks, portfolios)

Each domain has its own `DOMAIN_PROVIDER`. The domain providers file collects them all:

```
// Without domain providers registry

@NgModule({ providers: [ automobileDomainProvider, agricultureDomainProvider,
chemistryDomainProvider, // ... manually list each one ] })
```

```
// With domain providers registry

import { DOMAIN_PROVIDERS } from '../domain-config/domain-providers';



@NgModule({ providers: [ ...DOMAIN_PROVIDERS  // Spread all domain providers ] })
```

Benefits:

- **Single source of truth** — All domains listed in one file
- **Easy to add domains** — Add one line to add a new domain
- **Clear inventory** — Easy to see what domains exist

For Vvroom (Automobile Only)

Since vvroom only has automobiles, the domain providers file is simple. But the pattern prepares for future expansion.

What

Step 608.1: Create the Domain Providers File

Create the file that exports all domain providers.

Create `src/app/domain-config/domain-providers.ts`:

```
// src/app/domain-config/domain-providers.ts

import { Provider } from '@angular/core'; import { DOMAIN_PROVIDER as
automobileDomainProvider } from './automobile';



/**



  • Array of all domain configuration providers.

 *

  • Each domain should export a DOMAIN_PROVIDER that can be added to this array.



  • This allows for dynamic registration of domains at application startup.

 *

  • @example



  •
```

typescript

• // Add a new domain

• import { DOMAIN_PROVIDER as newDomainProvider } from './new-domain';

*

• export const DOMAIN_PROVIDERS: Provider[] = [

• automobileDomainProvider,

• newDomainProvider

•];

```
 /

export const DOMAIN_PROVIDERS: Provider[] = [ automobileDomainProvider, ];
```

Step 608.2: Understand the Provider Array Pattern

The `DOMAIN_PROVIDERS` array is a collection of Angular Provider objects:

```
// Each item is a Provider

const automobileDomainProvider: Provider = { provide: DOMAIN_CONFIG, useFactory:
createAutomobileDomainConfig, deps: [Injector] };



// The array collects all providers export const DOMAIN_PROVIDERS: Provider[] =
[ automobileDomainProvider, // future: agricultureDomainProvider, // future:
chemistryDomainProvider, ];
```

When used in a module:

```
@NgModule({

providers: [ ...DOMAIN_PROVIDERS  // Spreads all providers into the module ] })
```

This is equivalent to:

```
@NgModule({

providers: [ { provide: DOMAIN_CONFIG, useFactory: createAutomobileDomainConfig, deps:
[Injector] } // All other domain providers... ] })
```

Step 608.3: Understanding Single vs Multi Providers

Currently, `DOMAIN_PROVIDERS` contains one provider. In a multi-domain app, you might want the `DOMAIN_CONFIG` token to provide multiple configs.

Angular supports this with `multi: true`:

```
// Single provider (current approach)
{ provide: DOMAIN_CONFIG, useFactory: createAutomobileDomainConfig, deps: [Injector] }



// Multi-provider (for multiple domains) { provide: DOMAIN_CONFIG, useFactory:
createAutomobileDomainConfig, deps: [Injector], multi: true }, { provide:
DOMAIN_CONFIG, useFactory: createAgricultureDomainConfig, deps: [Injector], multi:
true }
```

With `multi: true`, injecting `DOMAIN_CONFIG` provides an array:

```
constructor(@Inject(DOMAIN_CONFIG) private configs: DomainConfig[]) {
// configs is an array of all registered domain configs }
```

For vvroom, we use single providers because we only have one domain. The pattern is established for future expansion.

Step 608.4: Understanding the Import Pattern

Note the import pattern:

```
import { DOMAIN_PROVIDER as automobileDomainProvider } from './automobile';
```

Each domain exports a `DOMAIN_PROVIDER`. We rename them on import to avoid conflicts:

```
// If we had multiple domains:

import { DOMAIN_PROVIDER as automobileDomainProvider } from './automobile'; import
{ DOMAIN_PROVIDER as agricultureDomainProvider } from './agriculture'; import
{ DOMAIN_PROVIDER as chemistryDomainProvider } from './chemistry';



export const DOMAIN_PROVIDERS: Provider[] = [ automobileDomainProvider,
agricultureDomainProvider, chemistryDomainProvider, ];
```

The `as` keyword prevents naming conflicts when all domains export the same `DOMAIN_PROVIDER` name.

Step 608.5: Understanding Usage in App Config

The domain providers are used in the application configuration:

```
// src/app/app.config.ts (document 906)

import { DOMAIN_PROVIDERS } from '../domain-config/domain-providers';



export const appConfig: ApplicationConfig = { providers: [ provideRouter(routes),
provideHttpClient(), ...DOMAIN_PROVIDERS,  // Register all domain configs ] };
```

This makes the domain configuration available throughout the application via dependency injection.

Step 608.6: Adding a New Domain (Future)

To add a new domain in the future, you would:

- Create the domain directory: `src/app/domain-config/new-domain/`
- Implement models, adapters, and configs (Phases 4-6 for new domain)
- Export `DOMAIN_PROVIDER` from `new-domain/index.ts`
- Add to domain-providers.ts:

```
import { DOMAIN_PROVIDER as automobileDomainProvider } from './automobile';
import { DOMAIN_PROVIDER as newDomainProvider } from './new-domain';


export const DOMAIN_PROVIDERS: Provider[] = [ automobileDomainProvider,
newDomainProvider,  // Add new domain ];
```

- Add routes for the new domain in `app.routes.ts`

That's it. No changes to framework code.

Verification

1. Verify File Created

```
$ ls -la src/app/domain-config/domain-providers.ts
```

Expected: File exists.

2. Verify TypeScript Compilation

```
$ cd ~/projects/vvroom
$ npx tsc --noEmit src/app/domain-config/domain-providers.ts
```

Expected: No compilation errors.

3. Verify Export

```
$ grep "^export const DOMAIN_PROVIDERS" src/app/domain-config/domain-providers.ts
```

Expected: `export const DOMAIN_PROVIDERS: Provider[] = [`

4. Verify Provider Count

```
$ grep -c "DomainProvider" src/app/domain-config/domain-providers.ts
```

Expected: `1` (automobileDomainProvider only)

Common Problems

Symptom	Cause	Solution
"Cannot find module './automobile'"	Barrel export missing	Ensure automobile/index.ts exists and exports DOMAIN_PROVIDER
"DOMAIN_PROVIDER is not exported"	Export missing from domain config	Add `export const DOMAIN_PROVIDER` to automobile.domain-config.ts
"Provider[] is not a type"	Missing import	Add `import { Provider } from '@angular/core';`
Circular dependency error	Domain imports from framework that imports domain	Check import paths; domain should only import from framework

---

Key Takeaways

- **Domain providers registry centralizes registration** — One file lists all domain providers
- **The pattern scales to multiple domains** — Add imports and array entries to add domains
- **Framework code remains unchanged** — New domains only require domain-config changes

---

Acceptance Criteria

- [ ] `src/app/domain-config/domain-providers.ts` exists
- [ ] File imports `DOMAIN_PROVIDER` from automobile domain

- [ ] `DOMAIN_PROVIDERS` array is exported
- [ ] Array contains automobile domain provider
- [ ] Documentation comments explain the pattern
- [ ] Example shows how to add new domains
- [ ] File compiles without TypeScript errors

Phase 6 Checkpoint

Congratulations! You have completed Phase 6: Automobile Domain Configs.

What you created:

Document	File	Purpose
601	automobile.filter-definitions.ts	Query panel filter controls
602	automobile.table-config.ts	Results table columns and behavior
603	automobile.picker-configs.ts	Manufacturer-model picker
604	automobile.query-control-filters.ts	Add filter dialog definitions
605	automobile.highlight-filters.ts	Chart segmentation filters
606	automobile.chart-configs.ts	Statistics panel charts
607	automobile.domain-config.ts	Assembly of all domain pieces
608	domain-providers.ts	Domain registration

The Phase 6 Aha Moment:

> "Configuration is declarative code. You describe what you want, not how to get it."

You defined filters, tables, pickers, and charts using configuration objects. The framework components (which you'll build in Phase 8) will interpret these configurations to generate the actual UI.

What's next:

- Phase 7: Chart Data Sources (documents 651-654) — Transform statistics into Plotly trace format
- Phase 8: Framework Components (documents 801-809) — Build the generic UI components

---

Next Step

Proceed to `651-manufacturer-chart-source.md` to implement the first chart data source.

651: Manufacturer Chart Source

Status: Planning **Depends On:** 403-automobile-statistics-model, 606-chart-configs **Blocks:** 801-base-chart-component

---

Learning Objectives

After completing this section, you will:

- Understand how chart data sources transform domain statistics into visualization-ready formats
- Know how to implement the abstract `ChartDataSource` class for a specific data type
- Be able to handle both simple and segmented (highlighted) statistics in chart transformations

---

Objective

Create the manufacturer chart data source that transforms vehicle statistics into a Plotly.js vertical stacked bar chart showing vehicle count by manufacturer. This chart source handles both simple counts and segmented statistics (with highlighted vs. non-highlighted data).

Why

Charts are one of the most powerful ways to communicate data insights. However, raw statistics from an API rarely match the exact format that charting libraries expect. We need a transformation layer between our domain data and our visualization library.

The Chart Data Source Pattern:

```
VehicleStatistics  →  ManufacturerChartDataSource  →  Plotly Chart Data
(domain)              (transformer)                   (visualization)
```

This separation provides several benefits:

- **Single Responsibility** — The chart component handles rendering; the data source handles transformation
- **Testability** — Data transformations can be unit tested without rendering actual charts
- **Reusability** — The same `BaseChartComponent` works with any data source
- **Flexibility** — Changing the chart appearance only requires modifying the data source, not the component

URL-First Architecture Reference

Chart sources connect to the URL-First pattern through click handling. When a user clicks a bar in the manufacturer chart, the data source converts that click into URL parameters (e.g., `manufacturer=Toyota`), which then updates the URL and triggers a new data fetch. This creates an interactive filtering experience where charts become navigation controls.

What

Step 651.1: Create the Manufacturer Chart Source File

Create the file `src/app/domain-config/automobile/chart-sources/manufacturer-chart-source.ts`:

```
// src/app/domain-config/automobile/chart-sources/manufacturer-chart-source.ts

/**


  • Manufacturer Chart Data Source

 *

  • Transforms vehicle statistics into Plotly.js vertical stacked bar chart



  • showing vehicle count by manufacturer with highlighted vs other.

 *

  • Domain: Automobile

 */

import { ChartDataSource, ChartData } from '../../../framework/components/base-chart/
base-chart.component'; import { VehicleStatistics } from '../models/
automobile.statistics';



/**


  • Manufacturer distribution chart data source

 *

  • Creates a vertical stacked bar chart of manufacturers by vehicle count.



  • Matches the visual style from the reference application.
```

```
 */

export class ManufacturerChartDataSource extends ChartDataSource<VehicleStatistics>
{ /**



  • Transform statistics into Plotly chart data


   */

transform( statistics: VehicleStatistics | null, highlights: any, _selectedValue:
string | null, _containerWidth: number ): ChartData | null { if (!statistics || !
statistics.byManufacturer) { return null; }



const entries = Object.entries(statistics.byManufacturer);



// Check if data has server-side segmented format ({total, highlighted}) const
isSegmented = entries.length > 0 && typeof entries[0][1] === 'object' && 'total' in
entries[0][1];



let traces: Plotly.Data[] = [];



if (isSegmented) { // Server-side segmented statistics: use backend data directly
const sorted = entries .sort((a, b) => { const aTotal = (a[1] as any).total || 0;
const bTotal = (b[1] as any).total || 0; return bTotal - aTotal; }) .slice(0, 20);



const manufacturers = sorted.map(([name]) => name); const highlightedCounts =
sorted.map(([, stats]: [string, any]) => stats.highlighted || 0); const otherCounts =
sorted.map(([, stats]: [string, any]) => (stats.total || 0) - (stats.highlighted ||
0) );



// Create stacked bar traces (Highlighted first at bottom, then Other on top) traces =
[ { type: 'bar', name: 'Highlighted', x: manufacturers, y: highlightedCounts, marker:
{ color: '#3B82F6' }, // Blue hovertemplate: '<b>%{x}</b><br>Highlighted: %{y}
<extra></extra>' }, { type: 'bar', name: 'Other', x: manufacturers, y: otherCounts,
marker: { color: '#9CA3AF' }, // Gray hovertemplate: '<b>%{x}</b><br>Other: %{y}
<extra></extra>' } ]; } else { // No highlights: simple blue bars using simple number
```

```
format const sorted = entries .map(([name, count]) => [name, typeof count === 'number' ? count : 0] as [string, number]) .sort((a, b) => b[1] - a[1]) .slice(0, 20);



const manufacturers = sorted.map(([name]) => name); const counts = sorted.map(([, count]) => count);



traces = [{ type: 'bar', x: manufacturers, y: counts, marker: { color: '#3B82F6' }, hovertemplate: '<b>%{x}</b><br>Count: %{y}<br><extra></extra>' }]; }



// Create layout const layout: Partial<Plotly.Layout> = { barmode: isSegmented ? 'stack' : undefined, xaxis: { tickangle: -45, automargin: true, color: '#FFFFFF', gridcolor: '#333333' }, yaxis: { title: { text: '' }, gridcolor: '#333333', automargin: true, color: '#FFFFFF' }, margin: { l: 60, r: 40, t: 40, b: 120 }, plot_bgcolor: '#000000', paper_bgcolor: '#1a1a1a', font: { color: '#FFFFFF' }, showlegend: isSegmented };



return { traces: traces, layout: layout }; }



/**



  • Get chart title

   */
getTitle(): string { return 'Vehicles by Manufacturer'; }



/**



  • Handle chart click event
```

```
   *

  • Supports both single-click and box selection.



  • Returns comma-separated manufacturers for OR filtering.



  • Backend API supports comma-separated values.

   */
handleClick(event: any): string | null { if (event.points && event.points.length > 0)
{ // Extract all manufacturer names from selected points const manufacturers: string[]
= event.points.map((point: any) => point.x as string);



// Remove duplicates (box selection may select both stacked bars) const
uniqueManufacturers: string[] = [...new Set(manufacturers)];



// Return comma-separated list (backend supports OR logic) return
uniqueManufacturers.join(',') || null; } return null; }



/**



  • Convert clicked value to URL parameters

   */
toUrlParams(value: string, isHighlightMode: boolean): Record<string, any> { const
paramName = isHighlightMode ? 'h_manufacturer' : 'manufacturer'; return { [paramName]:
value }; } }
```

Step 651.2: Understanding the Transform Method

The `transform` method is the heart of any chart data source. Let's break down its logic:

Input Parameters:

Parameter	Type	Purpose	
`statistics`	`VehicleStatistics \`	null	Domain statistics from the API
`highlights`	`any`	Highlight filter state (for visual distinction)	
`_selectedValue`	`string \`	null	Currently selected value (for active state)
`_containerWidth`	`number`	Container width for responsive sizing	

Null Check:

```
if (!statistics || !statistics.byManufacturer) {
return null; }
```

If statistics are missing or the `byManufacturer` property is absent, return `null`. The chart component will handle this gracefully by showing an empty state.

Segmentation Detection:

```
const isSegmented = entries.length > 0 &&
typeof entries[0][1] === 'object' && 'total' in entries[0][1];
```

The API can return manufacturer data in two formats:

- **Simple format:** `{ "Toyota": 234, "Honda": 187 }` — just counts
- **Segmented format:** `{ "Toyota": { total: 234, highlighted: 45 }, ... }` — with highlight breakdown

We detect which format we have by checking if the first value is an object with a `total` property.

Step 651.3: Understanding the Trace Structure

Plotly charts use "traces" — each trace is a data series. For the segmented case, we create two traces:

```
traces = [

{ type: 'bar', name: 'Highlighted', x: manufacturers,       // Category labels on X
axis y: highlightedCounts,   // Values on Y axis marker: { color: '#3B82F6' },  //
Blue color hovertemplate: '<b>%{x}</b><br>Highlighted: %{y}<extra></extra>' }, { type:
'bar', name: 'Other', x: manufacturers, y: otherCounts, marker: { color:
'#9CA3AF' },  // Gray color hovertemplate: '<b>%{x}</b><br>Other: %{y}<extra></
extra>' } ];
```

Why Two Traces?

When highlight filters are active, we want to show how much of each manufacturer's data matches the highlight criteria. The blue "Highlighted" bars show matching data; the gray "Other" bars show non-matching data. Stacking them shows the total while distinguishing the segments.

The Hover Template:

```
<b>%{x}</b><br>Highlighted: %{y}<extra></extra>
```

- `<b>%{x}</b>` — Bold manufacturer name
- `<br>` — Line break
- `Highlighted: %{y}` — Label and value
- `<extra></extra>` — Hides the trace name (which would otherwise appear)

Step 651.4: Understanding URL Parameter Mapping

The `toUrlParams` method converts a clicked chart value into URL query parameters:

```
toUrlParams(value: string, isHighlightMode: boolean): Record<string, any> {

const paramName = isHighlightMode ? 'h_manufacturer' : 'manufacturer'; return
{ [paramName]: value }; }
```

Filter Mode vs Highlight Mode:

- **Filter mode** (default): Click filters the dataset → `manufacturer=Toyota`
- **Highlight mode** (h key held): Click adds a highlight → `h_manufacturer=Toyota`

This dual-mode interaction follows the URL-First pattern: user actions translate directly to URL changes.

Verification

1. TypeScript Compilation

```
$ cd ~/projects/vvroom

$ ng build
```

Expected: Build succeeds with no errors.

2. File Location

```
$ ls -la src/app/domain-config/automobile/chart-sources/
```

Expected:

```
-rw-r--r-- 1 user user  4567 Feb  9 17:00 manufacturer-chart-source.ts
```

3. Export Check

Once the chart configs are updated (document 606), verify the class is importable:

```
import { ManufacturerChartDataSource } from './chart-sources/manufacturer-chart-source';
```

Common Problems

Symptom	Cause	Solution
`Cannot find module '../../../framework/components/base-chart/base-chart.component'`	BaseChartComponent not yet created	This is expected until Phase 8 (document 801)
`Cannot find module '../models/automobile.statistics'`	VehicleStatistics not created	Complete document 403 first
`Property 'byManufacturer' does not exist on type 'VehicleStatistics'`	Missing property in statistics model	Add `byManufacturer` property to VehicleStatistics
TypeScript error on `Plotly.Data`	Plotly types not installed	Run `npm install --save-dev @types/plotly.js`

Key Takeaways

- **Chart data sources are transformers** — They convert domain data into visualization-ready formats
- **Handle both simple and segmented data** — Statistics can come in multiple formats; check dynamically
- **URL parameters enable interactivity** — Clicks on charts become filter or highlight URL changes

---

Acceptance Criteria

- [ ] File exists at `src/app/domain-config/automobile/chart-sources/manufacturer-chart-source.ts`
- [ ] Class `ManufacturerChartDataSource` extends `ChartDataSource<VehicleStatistics>`
- [ ] `transform` method handles both simple and segmented statistics
- [ ] `getTitle` returns `'Vehicles by Manufacturer'`

- [ ] `handleClick` extracts manufacturer names from Plotly click events
- [ ] `toUrlParams` maps values to `manufacturer` or `h_manufacturer` parameters
- [ ] Code compiles without TypeScript errors (after dependencies are created)

Next Step

Proceed to `652-year-chart-source.md` to create the year distribution chart source.

652: Year Chart Source

Status: Planning **Depends On:** 403-automobile-statistics-model, 651-manufacturer-chart-source **Blocks:** 801-base-chart-component

---

Learning Objectives

After completing this section, you will:

- Understand how to handle time-series data in chart transformations
- Know how to implement range selection for year-based filtering
- Be able to convert multi-value selections into URL parameters

---

Objective

Create the year chart data source that transforms vehicle statistics into a Plotly.js bar chart showing vehicle distribution over time. This chart source handles year-based data and supports both single-year clicks and range selections.

Why

Time-series data presents unique visualization and interaction challenges:

- **Chronological ordering** — Years must be sorted ascending (oldest to newest), unlike other charts that sort by count
- **Range selection** — Users often want to filter by a range of years, not just a single year
- **Different URL parameter format** — Year ranges require two parameters (`yearMin` and `yearMax`) instead of one

The year chart demonstrates how a single abstract interface (`ChartDataSource`) can accommodate domain-specific requirements through concrete implementations.

Design Decision: Bar Chart for Years

While line charts are common for time series, we use a bar chart for years because:

- **Discrete values** — Vehicle model years are discrete integers, not continuous data
- **Highlight support** — Stacked bars clearly show highlighted vs. non-highlighted segments
- **Consistency** — Matches the visual style of other charts in the application

URL-First Architecture Reference

The year chart implements range selection by returning a pipe-delimited string (`2010|2020`) from `handleClick` , which `toUrlParams` then splits into separate `yearMin` and `yearMax` parameters. This demonstrates how chart sources can produce complex URL structures from simple user interactions.

What

Step 652.1: Create the Year Chart Source File

Create the file `src/app/domain-config/automobile/chart-sources/year-chart-source.ts` :

```
// src/app/domain-config/automobile/chart-sources/year-chart-source.ts

/**


  • Year Chart Data Source

 *

  • Transforms vehicle statistics into Plotly.js line chart



  • showing vehicle distribution over time.

 *

  • Domain: Automobile

 */

import { ChartDataSource, ChartData } from '../../../framework/components/base-chart/
base-chart.component'; import { VehicleStatistics } from '../models/
automobile.statistics';



/**


  • Year distribution chart data source

 *

  • Creates a line chart showing vehicle count by year.

 */
export class YearChartDataSource extends ChartDataSource<VehicleStatistics> { /**


```

```
  • Transform statistics into Plotly chart data


   */
transform( statistics: VehicleStatistics | null, _highlights: any, _selectedValue:
string | null, _containerWidth: number ): ChartData | null { if (!statistics || !
statistics.byYearRange) { return null; }



const entries = Object.entries(statistics.byYearRange);



// Check if data has server-side segmented format ({total, highlighted}) const
isSegmented = entries.length > 0 && typeof entries[0][1] === 'object' && 'total' in
entries[0][1];



let traces: Plotly.Data[] = [];



if (isSegmented) { // Server-side segmented statistics: use backend data directly
const sorted = entries .sort((a, b) => parseInt(a[0], 10) - parseInt(b[0], 10)); //
Sort by year ascending



const years = sorted.map(([year]) => year); const highlightedCounts = sorted.map(([,
stats]: [string, any]) => stats.highlighted || 0); const otherCounts = sorted.map(([,
stats]: [string, any]) => (stats.total || 0) - (stats.highlighted || 0) );



// Create stacked bar traces (Highlighted first at bottom, then Other on top) traces =
[ { type: 'bar', name: 'Highlighted', x: years, y: highlightedCounts, marker: { color:
'#3B82F6' }, hovertemplate: '<b>%{x}</b><br>Highlighted: %{y}<extra></extra>' },
{ type: 'bar', name: 'Other', x: years, y: otherCounts, marker: { color: '#9CA3AF' },
hovertemplate: '<b>%{x}</b><br>Other: %{y}<extra></extra>' } ]; } else { // No
highlights: simple blue bars using simple number format const sorted =
entries .map(([year, count]) => [year, typeof count === 'number' ? count : 0] as
[string, number]) .sort((a, b) => parseInt(a[0], 10) - parseInt(b[0], 10));
```

```
const years = sorted.map(([year]) => year); const counts = sorted.map(([, count]) =>
count);



traces = [{ type: 'bar', x: years, y: counts, marker: { color: '#3B82F6' },
hovertemplate: '<b>%{x}</b><br>Count: %{y}<br><extra></extra>' }]; }



// Create layout const layout: Partial<Plotly.Layout> = { barmode: isSegmented ?
'stack' : undefined, xaxis: { title: { text: '' }, gridcolor: '#333333', type:
'category', color: '#FFFFFF' }, yaxis: { title: { text: '' }, gridcolor: '#333333',
rangemode: 'tozero', automargin: true, color: '#FFFFFF' }, margin: { l: 60, r: 40, t:
40, b: 60 }, plot_bgcolor: '#000000', paper_bgcolor: '#1a1a1a', font: { color:
'#FFFFFF' }, showlegend: isSegmented };



return { traces: traces, layout: layout }; }



/**



  • Get chart title


   */
getTitle(): string { return 'Vehicles by Year'; }



/**



  • Handle chart click event


   *


  • Supports both single-click and box selection.



  • For box selection, returns year range as "min|max".
```

```
   */
handleClick(event: any): string | null { if (event.points && event.points.length > 0)
{ // Extract all years from selected points const years: number[] =
event.points.map((point: any) => parseInt(point.x, 10));



// Remove duplicates (box selection may select both stacked bars) const uniqueYears:
number[] = [...new Set(years)].sort((a, b) => a - b);



if (uniqueYears.length === 1) { // Single year selected return
uniqueYears[0].toString(); } else { // Multiple years: return as range min|max const
min = uniqueYears[0]; const max = uniqueYears[uniqueYears.length - 1]; return ${min}|
${max}; } } return null; }



/**



  • Convert clicked value to URL parameters

   *

  • Handles both single year and year range (min|max format).

   */
toUrlParams(value: string, isHighlightMode: boolean): Record<string, any> { if
(value.includes('|')) { // Year range: split into min/max params const [min, max] =
value.split('|'); return isHighlightMode ? { h_yearMin: min, h_yearMax: max } :
{ yearMin: min, yearMax: max }; } // Single year const paramName = isHighlightMode ?
'h_year' : 'year'; return { [paramName]: value }; } }
```

Step 652.2: Understanding Chronological Sorting

Unlike the manufacturer chart (which sorts by count descending), the year chart sorts chronologically:

```
const sorted = entries
.sort((a, b) => parseInt(a[0], 10) - parseInt(b[0], 10)); // Sort by year ascending
```

Why ascending order?

Time flows forward. Users expect to see older years on the left and newer years on the right. This matches mental models of timelines and makes trends (growth or decline over time) immediately visible.

Step 652.3: Understanding Category Type for X-Axis

Notice the layout configuration for the x-axis:

```
xaxis: {
title: { text: '' }, gridcolor: '#333333', type: 'category',  // Treat years as
categories, not numbers color: '#FFFFFF' }
```

Why `type: 'category'`?

Without this setting, Plotly would treat years as continuous numbers and might:

- Add decimal ticks (2015.5)
- Create gaps for missing years
- Misalign bars with labels

By specifying `type: 'category'`, each year becomes a discrete label with evenly spaced bars.

Step 652.4: Understanding Range Selection

The `handleClick` method demonstrates sophisticated selection handling:

```
handleClick(event: any): string | null {

if (event.points && event.points.length > 0) { // Extract all years from selected
points const years: number[] = event.points.map((point: any) => parseInt(point.x,
10));



// Remove duplicates (box selection may select both stacked bars) const uniqueYears:
number[] = [...new Set(years)].sort((a, b) => a - b);



if (uniqueYears.length === 1) { // Single year selected return
uniqueYears[0].toString(); } else { // Multiple years: return as range min|max const
min = uniqueYears[0]; const max = uniqueYears[uniqueYears.length - 1]; return ${min}|
${max}; } } return null; }
```

Single Click vs Box Selection:

Selection Type	Points Array	Return Value	
Single click on 2020	`[{x: '2020'}]`	`'2020'`	
Box select 2015-2020	`[{x: '2015'}, {x: '2016'}, ...]`	`'2015`	2020'

Duplicate Removal:

When a stacked bar chart has box selection, clicking might return the same year twice (once for each trace). Using `[...new Set(years)]` ensures each year appears only once.

Step 652.5: Understanding URL Parameter Conversion

The `toUrlParams` method handles both formats:

```
toUrlParams(value: string, isHighlightMode: boolean): Record<string, any> {

if (value.includes('|')) { // Year range: split into min/max params const [min, max] =
value.split('|'); return isHighlightMode ? { h_yearMin: min, h_yearMax: max } :
{ yearMin: min, yearMax: max }; } // Single year const paramName = isHighlightMode ?
'h_year' : 'year'; return { [paramName]: value }; }
```

URL Examples:

User Action	Return Value	URL Parameters
Click 2020	`'2020'`	`?year=2020`
Click 2020 (highlight mode)	`'2020'`	`?h_year=2020`
Box select 2015-2020	`'2015`	2020'	`?yearMin=2015&yearMax=2020`
Box select 2015-2020 (highlight mode)	`'2015`	2020'	`?h_yearMin=2015&h_yearMax=2020`

This demonstrates how a single abstract method can produce varied URL structures based on the interaction pattern.

Verification

1. TypeScript Compilation

```
$ cd ~/projects/vvroom

$ ng build
```

Expected: Build succeeds with no errors.

2. File Location

```
$ ls -la src/app/domain-config/automobile/chart-sources/
```

Expected:

```
-rw-r--r-- 1 user user  4567 Feb  9 17:00 manufacturer-chart-source.ts
-rw-r--r-- 1 user user  4123 Feb  9 17:05 year-chart-source.ts
```

3. Sorting Verification

Mentally trace through the code with sample data:

```
// Input (unsorted):
byYearRange: { "2022": 50, "2020": 100, "2021": 75 }


// After sorting: // years = ["2020", "2021", "2022"] // counts = [100, 75, 50]
```

The chart will show 2020 on the left with 100 vehicles, progressing to 2022 on the right with 50.

Common Problems

Symptom	Cause	Solution
Years appear in random order	Missing or incorrect sort	Verify `parseInt(a[0], 10) - parseInt(b[0], 10)` sorts ascending
Decimal year labels (2020.5)	X-axis type not set to category	Add `type: 'category'` to xaxis config
Range selection returns duplicates	Missing duplicate removal	Ensure `[...new Set(years)]` is present
`yearMin=2020&yearMax=2020` for single click	Pipe split on non-pipe string	Check for `` before splitting
`Cannot find module '../models/automobile.statistics'`	VehicleStatistics not created	Complete document 403 first

---

Key Takeaways

- **Time data requires chronological sorting** — Unlike counts, years have an inherent order that users expect
- **Range selections need special handling** — Convert multi-point selections into min/max pairs
- **URL parameters can be polymorphic** — The same method produces single or multi-parameter outputs based on input

---

Acceptance Criteria

- [ ] File exists at `src/app/domain-config/automobile/chart-sources/year-chart-source.ts`
- [ ] Class `YearChartDataSource` extends `ChartDataSource<VehicleStatistics>`
- [ ] Years are sorted chronologically (ascending) in the chart
- [ ] X-axis uses category type for discrete year labels
- [ ] `handleClick` returns single year or `min|max` format for ranges
- [ ] `toUrlParams` handles both single year and range formats
- [ ] Highlight mode produces `h_year`, `h_yearMin`, `h_yearMax` parameters

- [ ] Code compiles without TypeScript errors (after dependencies are created)

Next Step

Proceed to `653-body-class-chart-source.md` to create the body class distribution chart source.

653: Body Class Chart Source

Status: Planning **Depends On:** 403-automobile-statistics-model, 652-year-chart-source **Blocks:** 801-base-chart-component

---

Learning Objectives

After completing this section, you will:

- Understand how to maintain unused code for future flexibility (color mappings)
- Know how to implement multi-value selection for categorical data
- Recognize patterns that repeat across chart sources and opportunities for refactoring

---

Objective

Create the body class chart data source that transforms vehicle statistics into a Plotly.js bar chart showing distribution by body class (Sedan, SUV, Truck, etc.). This chart source handles categorical data and supports comma-separated multi-selection.

Why

Body class represents a categorical dimension of vehicle data. Unlike manufacturers (which can number in the dozens) or years (which span decades), body classes form a small, fixed set of categories. This affects our design choices:

- **Color scheme preparation** — Although we currently use uniform blue bars, we define a color mapping for potential future use (e.g., distinct colors per body class)
- **Full display** — We show all body classes rather than limiting to "top 20" since the set is naturally small

- **Descending count sort** — Like manufacturers, we sort by count (most common body classes first)

Design Decision: Bar Chart Over Pie Chart

The original file comment mentions "pie chart," but the implementation uses a bar chart. Why?

- **Comparison accuracy** — Humans judge bar heights more accurately than pie slice angles
- **Highlight support** — Stacked bars work better than segmented pie slices for showing highlighted vs. non-highlighted data
- **Consistency** — Matches the visual style of manufacturer and year charts

URL-First Architecture Reference

Body class filtering uses comma-separated values for OR logic: `bodyClass=Sedan,SUV` matches vehicles that are either Sedans OR SUVs. This supports intuitive multi-selection where clicking multiple bars expands rather than narrows the result set.

What

Step 653.1: Create the Body Class Chart Source File

Create the file `src/app/domain-config/automobile/chart-sources/body-class-chart-source.ts`:

```
// src/app/domain-config/automobile/chart-sources/body-class-chart-source.ts

/**


  • Body Class Chart Data Source

 *

  • Transforms vehicle statistics into Plotly.js pie chart



  • showing distribution by body class.

 *

  • Domain: Automobile

 */

import { ChartDataSource, ChartData } from '../../../framework/components/base-chart/
base-chart.component'; import { VehicleStatistics } from '../models/
automobile.statistics';



/**


  • Body class distribution chart data source

 *

  • Creates a pie chart showing vehicle distribution by body class.

 */
export class BodyClassChartDataSource extends ChartDataSource<VehicleStatistics> { /**
```

```
  • Color scheme for body classes


   */
private readonly BODY_CLASS_COLORS: Record<string, string> = { 'Sedan': '#3B82F6',
'SUV': '#10B981', 'Truck': '#F59E0B', 'Pickup': '#F59E0B', 'Coupe': '#EF4444',
'Wagon': '#8B5CF6', 'Van': '#EC4899', 'Minivan': '#06B6D4', 'Convertible': '#84CC16',
'Hatchback': '#F97316' };



/**



  • Transform statistics into Plotly chart data


   */
transform( statistics: VehicleStatistics | null, _highlights: any, _selectedValue:
string | null, _containerWidth: number ): ChartData | null { if (!statistics || !
statistics.byBodyClass) { return null; }



const entries = Object.entries(statistics.byBodyClass);



// Check if data has server-side segmented format ({total, highlighted}) const
isSegmented = entries.length > 0 && typeof entries[0][1] === 'object' && 'total' in
entries[0][1];



let traces: Plotly.Data[] = [];



if (isSegmented) { // Server-side segmented statistics: use backend data directly
const sorted = entries .sort((a, b) => { const aTotal = (a[1] as any).total || 0;
const bTotal = (b[1] as any).total || 0; return bTotal - aTotal; });
```

```
const labels = sorted.map(([name]) => name); const highlightedCounts = sorted.map(([,
stats]: [string, any]) => stats.highlighted || 0); const otherCounts = sorted.map(([,
stats]: [string, any]) => (stats.total || 0) - (stats.highlighted || 0) );



// Create stacked bar traces (Highlighted first at bottom, then Other on top) traces =
[ { type: 'bar', name: 'Highlighted', x: labels, y: highlightedCounts, marker:
{ color: '#3B82F6' }, hovertemplate: '<b>%{x}</b><br>Highlighted: %{y}<extra></
extra>' }, { type: 'bar', name: 'Other', x: labels, y: otherCounts, marker: { color:
'#9CA3AF' }, hovertemplate: '<b>%{x}</b><br>Other: %{y}<extra></extra>' } ]; } else
{ // No highlights: simple blue bars using simple number format const sorted =
entries .map(([name, count]) => [name, typeof count === 'number' ? count : 0] as
[string, number]) .sort((a, b) => b[1] - a[1]);



const labels = sorted.map(([name]) => name); const counts = sorted.map(([, count]) =>
count);



traces = [{ type: 'bar', x: labels, y: counts, marker: { color: '#3B82F6' },
hovertemplate: '<b>%{x}</b><br>Count: %{y}<br><extra></extra>' }]; }



// Create layout const layout: Partial<Plotly.Layout> = { barmode: isSegmented ?
'stack' : undefined, xaxis: { tickangle: -45, automargin: true, color: '#FFFFFF',
gridcolor: '#333333' }, yaxis: { title: { text: '' }, gridcolor: '#333333',
automargin: true, color: '#FFFFFF' }, margin: { l: 60, r: 40, t: 40, b: 100 },
plot_bgcolor: '#000000', paper_bgcolor: '#1a1a1a', font: { color: '#FFFFFF' },
showlegend: isSegmented };



return { traces: traces, layout: layout }; }



/**



  • Get chart title


   */
getTitle(): string { return 'Vehicles by Body Class'; }
```

```
/**


  • Handle chart click event

   *

  • Supports both single-click and box selection.


  • Returns comma-separated body classes for OR filtering.


  • Backend API v1.0.1+ supports comma-separated values.

   */
handleClick(event: any): string | null { if (event.points && event.points.length > 0)
{ // Extract all body class names from selected points const bodyClasses: string[] =
event.points.map((point: any) => point.x as string);



// Remove duplicates (box selection may select both stacked bars) const
uniqueBodyClasses: string[] = [...new Set(bodyClasses)];



// Return comma-separated list (backend supports OR logic as of v1.0.1) return
uniqueBodyClasses.join(',') || null; } return null; }



/**


  • Convert clicked value to URL parameters

   */
```

```
toUrlParams(value: string, isHighlightMode: boolean): Record<string, any> { const
paramName = isHighlightMode ? 'h_bodyClass' : 'bodyClass'; return { [paramName]:
value }; } }
```

Step 653.2: Understanding the Color Scheme

Notice the unused `BODY_CLASS_COLORS` property:

```
private readonly BODY_CLASS_COLORS: Record<string, string> = {

'Sedan': '#3B82F6', 'SUV': '#10B981', 'Truck': '#F59E0B', 'Pickup': '#F59E0B',
'Coupe': '#EF4444', 'Wagon': '#8B5CF6', 'Van': '#EC4899', 'Minivan': '#06B6D4',
'Convertible': '#84CC16', 'Hatchback': '#F97316' };
```

Why include unused code?

This is a deliberate design choice. The color mapping:

- **Documents intent** — Shows that distinct colors per body class were considered
- **Enables future enhancement** — Can be activated without research into appropriate colors
- **Provides consistency** — Truck and Pickup share the same orange (#F59E0B), indicating semantic similarity

Currently, the code uses uniform blue bars for consistency with other charts. The color scheme remains available for future features like:

- Color-coded bars when highlights are not active
- Legend entries matching body class colors
- Tooltips with color indicators

Step 653.3: Comparing Chart Sources

By now, you may notice similarities between chart sources. Let's compare:

Aspect	Manufacturer	Year	Body Class
Data property	`byManufacturer`	`byYearRange`	`byBodyClass`
Sort order	Count descending	Year ascending	Count descending
Limit	Top 20	All	All
X-axis labels	Angled (-45)	Normal	Angled (-45)
Bottom margin	120px	60px	100px
URL parameter	`manufacturer`	`year` / `yearMin` / `yearMax`	`bodyClass`

Patterns to Notice:

- **Same structure** — All three follow the same `transform` → `getTitle` → `handleClick` → `toUrlParams` pattern
- **Same segmentation logic** — The `isSegmented` check is identical
- **Same trace structure** — Blue for highlighted, gray for other

Why not abstract further?

You might be tempted to create a generic chart source factory. Resist this urge initially:

- **Explicit is better than implicit** — Each file is self-contained and understandable
- **Customization is easy** — Modifying one chart doesn't risk breaking others
- **Patterns emerge naturally** — After building several, refactoring opportunities become clear

The fourth chart source (Top Models) will show a more complex case that would break a simple abstraction.

Step 653.4: Understanding Bottom Margin Differences

Notice the different bottom margins:

```
// Manufacturer chart
margin: { l: 60, r: 40, t: 40, b: 120 }


// Year chart margin: { l: 60, r: 40, t: 40, b: 60 }


// Body class chart margin: { l: 60, r: 40, t: 40, b: 100 }
```

Why the differences?

- **Manufacturer (120px)** — Long names like "Mercedes-Benz" need more space when angled
- **Year (60px)** — 4-digit years are short; no angling needed
- **Body Class (100px)** — Medium-length names like "Convertible" need moderate space

These seemingly minor details significantly affect chart readability. When labels overlap or get cut off, users lose information.

Verification

1. TypeScript Compilation

```
$ cd ~/projects/vvroom
$ ng build
```

Expected: Build succeeds with no errors.

2. File Location

```
$ ls -la src/app/domain-config/automobile/chart-sources/
```

Expected:

```
-rw-r--r-- 1 user user  4567 Feb  9 17:00 manufacturer-chart-source.ts

-rw-r--r-- 1 user user  4123 Feb  9 17:05 year-chart-source.ts -rw-r--r-- 1 user user
4234 Feb  9 17:10 body-class-chart-source.ts
```

3. Color Scheme Verification

Verify the color scheme covers expected body classes:

```
$ grep -c "'" src/app/domain-config/automobile/chart-sources/body-class-chart-
source.ts
```

Expected: Multiple single quotes indicating the color mapping entries.

Common Problems

Symptom	Cause	Solution
Labels cut off at bottom	Margin too small	Increase `b` (bottom) margin value
"Pickup" and "Truck" have same color	Intentional design	These are semantically similar categories
Color scheme not applied	Code uses uniform blue	The color scheme is for future use; current implementation uses blue
Multi-selection returns duplicates	Missing `Set` deduplication	Ensure `[...new Set(bodyClasses)]` is present
`Cannot find module '../models/automobile.statistics'`	VehicleStatistics not created	Complete document 403 first

---

Key Takeaways

- **Unused code can be intentional** — Color schemes document design decisions for future enhancement
- **Small sets need different treatment** — No "top 20" limit needed for naturally small categories

- **Margins matter for readability** — Adjust bottom margins based on label length and angle

---

Acceptance Criteria

- [ ] File exists at `src/app/domain-config/automobile/chart-sources/body-class-chart-source.ts`
- [ ] Class `BodyClassChartDataSource` extends `ChartDataSource<VehicleStatistics>`
- [ ] Color scheme defined for common body classes (even if not currently used)
- [ ] Body classes sorted by count descending
- [ ] `handleClick` returns comma-separated body class names
- [ ] `toUrlParams` maps values to `bodyClass` or `h_bodyClass` parameters
- [ ] Code compiles without TypeScript errors (after dependencies are created)

Next Step

Proceed to `654-top-models-chart-source.md` to create the top models chart source, which demonstrates more complex data transformation.

654: Top Models Chart Source

Status: Planning **Depends On:** 403-automobile-statistics-model, 653-body-class-chart-source **Blocks:** 801-base-chart-component

---

Learning Objectives

After completing this section, you will:

- Understand how to transform nested data structures for visualization
- Know how to implement composite URL parameters from chart clicks
- Be able to handle data from multiple API response formats

Objective

Create the top models chart data source that transforms vehicle statistics into a Plotly.js bar chart showing the top models by VIN instance count. This chart source handles nested data structures (`modelsByManufacturer`) and produces composite URL parameters (`manufacturer:model` format).

Why

The top models chart presents unique challenges that the previous chart sources did not:

- **Nested data structure** — Models are nested within manufacturers in the API response
- **Composite labels** — Each bar shows "Manufacturer Model" (e.g., "Toyota Camry")
- **Composite URL parameters** — Clicking produces `modelCombos=Toyota:Camry` , not separate parameters
- **Two data sources** — Can use either `topModels` array or `modelsByManufacturer` object

This complexity demonstrates why we keep chart sources as separate classes rather than abstracting too aggressively. Each chart source handles its unique requirements while maintaining the same interface.

Design Decision: Combined Manufacturer-Model Format

When a user clicks on a model bar, we need to filter by both manufacturer AND model. We combine these into a single URL parameter using colon separator:

```
?modelCombos=Toyota:Camry
```

This approach:

- **Avoids ambiguity** — "Camry" alone could match multiple manufacturers if the API had duplicates
- **Supports multi-select** — `modelCombos=Toyota:Camry,Honda:Accord` selects multiple models
- **Matches API expectations** — The backend parses this format for efficient filtering

URL-First Architecture Reference

The top models chart shows how chart sources can produce complex URL structures. The `handleClick` method transforms display format ("Toyota Camry") to URL format ("Toyota:Camry"), demonstrating that chart sources aren't just data transformers but also interaction translators.

What

Step 654.1: Create the Top Models Chart Source File

Create the file `src/app/domain-config/automobile/chart-sources/top-models-chart-source.ts`:

```
// src/app/domain-config/automobile/chart-sources/top-models-chart-source.ts

/**


  • Top Models Chart Data Source

 *

  • Transforms vehicle statistics into Plotly.js horizontal bar chart


  • showing top models by VIN instance count.

 *

  • Domain: Automobile

 */

import { ChartDataSource, ChartData } from '../../../framework/components/base-chart/
base-chart.component'; import { VehicleStatistics } from '../models/
automobile.statistics';



/**


  • Top models chart data source

 *

  • Creates a horizontal bar chart of top models by VIN instance count.

 */
export class TopModelsChartDataSource extends ChartDataSource<VehicleStatistics> { /**
```

```
  • Transform statistics into Plotly chart data


   */
transform( statistics: VehicleStatistics | null, _highlights: any, _selectedValue:
string | null, _containerWidth: number ): ChartData | null { if (!statistics || !
statistics.topModels || statistics.topModels.length === 0) { return null; }



// Check if we have segmented statistics from API (with total/highlighted counts)
const hasSegmentedStats = statistics.modelsByManufacturer &&
Object.values(statistics.modelsByManufacturer).some(models => typeof models ===
'object' && Object.values(models).some(v => typeof v === 'object' && 'total' in v ) );



let traces: any[] = [];



if (hasSegmentedStats && statistics.modelsByManufacturer) { // Use API's segmented
statistics with {total, highlighted} const modelEntries: Array<[string, any]> = [];



Object.entries(statistics.modelsByManufacturer).forEach(([manufacturer, models]) =>
{ Object.entries(models).forEach(([modelName, stats]) => { modelEntries.push([ $
{manufacturer} ${modelName} , stats]); }); });



// Sort by total count descending and take top 20 const sorted =
modelEntries .sort((a, b) => ((b[1] as any).total || 0) - ((a[1] as any).total ||
0)) .slice(0, 20);



const modelLabels = sorted.map(([label]) => label); const highlightedCounts =
sorted.map(([, stats]: [string, any]) => stats.highlighted || 0); const otherCounts =
sorted.map(([, stats]: [string, any]) => (stats.total || 0) - (stats.highlighted ||
0) );



// Create stacked bar traces (Highlighted first at bottom, then Other on top) traces =
[ { type: 'bar', name: 'Highlighted', x: modelLabels, y: highlightedCounts, marker:
```

```
{ color: '#3B82F6' }, hovertemplate: '<b>%{x}</b><br>Highlighted: %{y}<extra></
extra>' }, { type: 'bar', name: 'Other', x: modelLabels, y: otherCounts, marker:
{ color: '#9CA3AF' }, hovertemplate: '<b>%{x}</b><br>Other: %{y}<extra></
extra>' } ]; } else { // Fallback: simple blue bars using topModels const topModels =
statistics.topModels.slice(0, 20); const modelLabels = topModels.map(m => $
{m.manufacturer} ${m.name}); const counts = topModels.map(m => m.instanceCount);



traces = [{ type: 'bar', x: modelLabels, y: counts, marker: { color: '#3B82F6' },
hovertemplate: '<b>%{x}</b><br>Count: %{y}<br><extra></extra>' }]; }



// Create layout const layout: Partial<any> = { barmode: hasSegmentedStats ? 'stack' :
undefined, xaxis: { tickangle: -45, automargin: true, color: '#FFFFFF', gridcolor:
'#333333' }, yaxis: { title: '', gridcolor: '#333333', automargin: true, color:
'#FFFFFF' }, margin: { l: 60, r: 40, t: 40, b: 140 }, plot_bgcolor: '#000000',
paper_bgcolor: '#1a1a1a', font: { color: '#FFFFFF' }, showlegend: hasSegmentedStats };



return { traces: traces, layout: layout }; }



/**



  • Get chart title


   */
getTitle(): string { return 'Top Models by VIN Count'; }



/**



  • Handle chart click event


   *


  • Supports both single-click and box selection.
```

```
  • For box selection, returns comma-separated list of unique models.



  • Converts label format from "Manufacturer Model" to "Manufacturer:Model".

   */
handleClick(event: any): string | null { if (event.points && event.points.length > 0)
{ // Extract all model labels from selected points (format: "Manufacturer Model")
const modelLabels: string[] = event.points.map((point: any) => point.x as string);



// Remove duplicates (box selection may select both stacked bars) const uniqueLabels =
[...new Set(modelLabels)];



// Convert from "Manufacturer Model" to "Manufacturer:Model" format const modelCombos
= uniqueLabels.map(label => { // Replace first space with colon (manufacturer doesn't
have spaces) return label.replace(' ', ':'); });



// Return comma-separated list (or single value) return modelCombos.join(','); }
return null; }



/**



  • Convert clicked value to URL parameters

   */
toUrlParams(value: string, isHighlightMode: boolean): Record<string, any> { const
paramName = isHighlightMode ? 'h_modelCombos' : 'modelCombos'; return { [paramName]:
value }; } }
```

Step 654.2: Understanding Nested Data Transformation

The top models chart handles a more complex data structure:

```
// Statistics structure

{ modelsByManufacturer: { "Toyota": { "Camry": { total: 234, highlighted: 45 },
"Corolla": { total: 189, highlighted: 32 } }, "Honda": { "Accord": { total: 156,
highlighted: 28 }, "Civic": { total: 145, highlighted: 21 } } } }
```

The transformation flattens this nested structure:

```
Object.entries(statistics.modelsByManufacturer).forEach(([manufacturer, models]) => {

Object.entries(models).forEach(([modelName, stats]) => { modelEntries.push([ $
{manufacturer} ${modelName} , stats]); }); });
```

Result:

```
modelEntries = [

["Toyota Camry", { total: 234, highlighted: 45 }], ["Toyota Corolla", { total: 189,
highlighted: 32 }], ["Honda Accord", { total: 156, highlighted: 28 }], ["Honda Civic",
{ total: 145, highlighted: 21 }] ]
```

This creates a flat array suitable for sorting and slicing to get the top 20.

Step 654.3: Understanding Dual Data Source Fallback

Notice the two code paths:

```
if (hasSegmentedStats && statistics.modelsByManufacturer) {

// Use nested structure with highlight support } else { // Fallback: use topModels
array const topModels = statistics.topModels.slice(0, 20); const modelLabels =
topModels.map(m => `${m.manufacturer} $
{m.name}`); const counts = topModels.map(m => m.instanceCount); }
```

Why two approaches?

The API can return model data in two formats:

- **Segmented format** (`modelsByManufacturer`) — Has highlight counts, needs flattening
- **Array format** (`topModels`) — Pre-sorted array, simpler to use but no highlight data

The segmented format is preferred when available (for highlight support), but the fallback ensures the chart works with simpler API responses.

Step 654.4: Understanding Label Format Conversion

The click handler converts display format to URL format:

```
// Convert from "Manufacturer Model" to "Manufacturer:Model" format

const modelCombos = uniqueLabels.map(label => { // Replace first space with colon
(manufacturer doesn't have spaces) return label.replace(' ', ':'); });
```

Why this conversion?

Format	Purpose	Example
Display	Human-readable chart labels	"Toyota Camry"
URL	Machine-parseable parameter	"Toyota:Camry"

The colon separator enables the backend to split the value and query by both manufacturer AND model efficiently.

Edge Case: Multi-Word Models

Consider "Ford Mustang Mach-E":

```
"Ford Mustang Mach-E".replace(' ', ':')
// Result: "Ford:Mustang Mach-E"
```

The first space becomes a colon, preserving "Mustang Mach-E" as the model name. This works because manufacturer names in this dataset don't contain spaces.

Step 654.5: Understanding the Larger Bottom Margin

The top models chart uses the largest bottom margin:

```
margin: {
l: 60, r: 40, t: 40, b: 140  // Largest of all charts }
```

Why 140px?

Model labels are the longest of any chart:

- "Toyota Camry" — Short
- "Mercedes-Benz S-Class" — Medium
- "Lamborghini Aventador LP700-4" — Long

When angled at -45 degrees, these long labels need substantial vertical space to remain readable without overlapping or being cut off.

Step 654.6: Understanding the Segmentation Check

The segmentation check for top models is more complex:

```
const hasSegmentedStats = statistics.modelsByManufacturer &&
Object.values(statistics.modelsByManufacturer).some(models => typeof models ===
'object' && Object.values(models).some(v => typeof v === 'object' && 'total' in v ) );
```

Why so complex?

Unlike `byManufacturer` (which is one level deep), `modelsByManufacturer` is two levels deep:

```
modelsByManufacturer → manufacturer → model → { total, highlighted }
```

We need to:

- Check that `modelsByManufacturer` exists
- Check that at least one manufacturer has models
- Check that at least one model has the `{ total, highlighted }` structure

This defensive checking prevents runtime errors when data is missing or in an unexpected format.

Verification

1. TypeScript Compilation

```
$ cd ~/projects/vvroom
$ ng build
```

Expected: Build succeeds with no errors.

2. File Location

```
$ ls -la src/app/domain-config/automobile/chart-sources/
```

Expected:

```
-rw-r--r-- 1 user user  4567 Feb  9 17:00 manufacturer-chart-source.ts
-rw-r--r-- 1 user user  4123 Feb  9 17:05 year-chart-source.ts -rw-r--r-- 1 user user
4234 Feb  9 17:10 body-class-chart-source.ts -rw-r--r-- 1 user user  4789 Feb  9 17:15
top-models-chart-source.ts
```

3. Label Conversion Test

Mentally trace the click handler:

```
// Input from Plotly click event
event.points = [{ x: "Toyota Camry" }, { x: "Honda Accord" }]



// After processing // uniqueLabels = ["Toyota Camry", "Honda Accord"] // modelCombos
= ["Toyota:Camry", "Honda:Accord"] // return value = "Toyota:Camry,Honda:Accord"



// URL parameter // ?modelCombos=Toyota:Camry,Honda:Accord
```

Common Problems

Symptom	Cause	Solution
Chart shows no data	`topModels` array empty	Verify API returns model data
Labels show "undefined undefined"	Missing manufacturer or name	Check `topModels` array structure
"Ford:Mustang:Mach-E" (two colons)	Multiple `replace()` calls	Use `replace(' ', ':')` not `replaceAll`
Segmentation not detected	Nested check failing	Verify `modelsByManufacturer` has expected structure
Labels overlap	Bottom margin too small	Increase `b` value in margins
`Cannot find module '../models/automobile.statistics'`	VehicleStatistics not created	Complete document 403 first

---

Key Takeaways

- **Nested data requires flattening** — Transform hierarchical API responses into flat arrays for charting
- **Display format differs from URL format** — Convert human-readable labels to machine-parseable parameters
- **Fallback data sources add resilience** — Handle multiple API response formats gracefully

---

Acceptance Criteria

- [ ] File exists at `src/app/domain-config/automobile/chart-sources/top-models-chart-source.ts`
- [ ] Class `TopModelsChartDataSource` extends `ChartDataSource<VehicleStatistics>`
- [ ] Handles both `modelsByManufacturer` (segmented) and `topModels` (array) formats
- [ ] Labels show "Manufacturer Model" format (e.g., "Toyota Camry")
- [ ] `handleClick` converts to "Manufacturer:Model" format
- [ ] `toUrlParams` maps values to `modelCombos` or `h_modelCombos` parameters
- [ ] Bottom margin (140px) accommodates long model names
- [ ] Code compiles without TypeScript errors (after dependencies are created)

Phase 7 Complete

You have now completed all chart data sources for the automobile domain:

Document	Chart Source	Data Property	URL Parameter
651	ManufacturerChartDataSource	`byManufacturer`	`manufacturer`
652	YearChartDataSource	`byYearRange`	`year`, `yearMin`, `yearMax`
653	BodyClassChartDataSource	`byBodyClass`	`bodyClass`
654	TopModelsChartDataSource	`modelsByManufacturer`, `topModels`	`modelCombos`

Phase 7 Aha Moment: "Chart sources transform domain data into visualization-ready formats."

Each chart source:

- Transforms domain statistics into Plotly traces and layouts
- Handles both simple and segmented (highlighted) data
- Converts user interactions into URL parameters

Next Step

Proceed to Phase 8 (document 801) to build the `BaseChartComponent` that uses these data sources to render interactive charts.

801: Base Chart Component

Status: Planning **Depends On:** 201-domain-config-interface, 301-url-state-service, 306-resource-management-service, 651-654 (Chart Data Sources) **Blocks:** 804-statistics-panel-component

---

Learning Objectives

After completing this section, you will:

- Understand the data source pattern for transforming domain data into chart-ready formats
- Know how to integrate Plotly.js with Angular's change detection strategy
- Be able to implement reusable chart containers that work with any domain configuration

---

Objective

Build a domain-agnostic chart component that renders Plotly.js visualizations based on pluggable data sources. This component accepts any `ChartDataSource` implementation and handles all chart lifecycle concerns: rendering, resizing, event handling, and cleanup.

Why

Charts are a critical feature for data discovery applications. However, each domain (automobiles, agriculture, etc.) has different data structures and visualization needs. Rather than creating domain-specific chart components, we build one generic container that works with any data.

The **Data Source Pattern** separates concerns:

- **BaseChartComponent** handles Plotly.js integration, DOM management, and events
- **ChartDataSource** implementations handle domain-specific data transformation

This pattern is the Phase 8 "aha moment": **Generic components + specific configuration = infinite reusability**.

Angular Style Guide References

- Style 03-01: Use single responsibility principle
- Style 05-15: Provide a selector prefix

URL-First Architecture Reference

When a user clicks on a chart element (e.g., a bar for "Toyota"), the click event flows through the data source's `toUrlParams()` method, which converts the click into URL query parameters. This maintains our URL-First principle: all state changes flow through URL updates.

What

Step 801.1: Install Plotly.js Dependency

First, install the Plotly.js minified distribution:

```
$ cd ~/projects/vvroom
$ npm install plotly.js-dist-min --save
```

Verify installation:

```
$ grep "plotly" package.json
"plotly.js-dist-min": "^2.29.1",
```

Step 801.2: Create the Chart Data Interface

Create the file `src/app/framework/components/base-chart/chart-data.interface.ts`:

```
// src/app/framework/components/base-chart/chart-data.interface.ts
// VERSION 1 (Section 801) - Chart data structures


/**


  • Chart data structure for Plotly.js

 *

  • Contains the traces (data series), layout configuration, and optional


  • interaction state from previous user clicks.

 */
export interface ChartData { /**


  • Array of Plotly data traces (each trace represents a data series)

   */
traces: any[];


/**


  • Plotly layout configuration controlling chart appearance, axes, titles

   */
layout: Partial<any>;


/**
```

```
  • Optional click event data from the last chart interaction

   */
clickData?: any; }


/**

  • Abstract chart data source

 *

  • Transforms domain statistics into Plotly-ready chart data.


  • Each domain implements concrete data sources that extend this class.

 *

  • @template TStatistics - Domain-specific statistics type

 *

  • @example


  •
```

typescript

- export class ManufacturerChartSource extends ChartDataSource<VehicleStatistics> {
- transform(statistics, highlights, selectedValue, width) {
- // Transform VehicleStatistics into bar chart traces
- }
- }

```
 /
export abstract class ChartDataSource<TStatistics = any> { /**


  • Transform statistics into chart data

   *

  • @param statistics - Domain statistics from ResourceManagementService


  • @param highlights - Highlight filters for visual emphasis


  • @param selectedValue - Currently selected value to highlight


  • @param containerWidth - Container width for responsive sizing


  • @returns Chart data or null if no data available

   */
abstract transform( statistics: TStatistics | null, highlights: any, selectedValue:
string | null, containerWidth: number ): ChartData | null;



/**


  • Get chart title for display

   */
abstract getTitle(): string;
```

```
/**

  • Handle chart click event

   *

  • Extracts the meaningful value from a Plotly click event.

  • For a bar chart, this might be the category name.

  • For a scatter plot, this might be coordinates.

   *

  • @param event - Plotly click event object

  • @returns Clicked value string or null if click is not actionable

   */
abstract handleClick(event: any): string | null;

/**

  • Convert clicked value to URL parameters

   *

  • Maps the chart's clicked value to URL query parameters.

    Handles both filter mode (regular params) and highlight mode (h_ params).
*
```

```
  • @param value - The clicked value from handleClick()


  • @param isHighlightMode - Whether highlight mode is active (h key held)


  • @returns URL parameters object to merge into the query string
   *
  • @example


  •
```

typescript

- // ManufacturerChartDataSource
- toUrlParams('Toyota', false) // { manufacturer: 'Toyota' }
- toUrlParams('Toyota', true) // { h_manufacturer: 'Toyota' }

```
   /
abstract toUrlParams(value: string, isHighlightMode: boolean): Record<string, any>; }
```

Step 801.3: Create the Base Chart Component TypeScript

Create the file `src/app/framework/components/base-chart/base-chart.component.ts`:

```
// src/app/framework/components/base-chart/base-chart.component.ts

// VERSION 1 (Section 801) - Generic Plotly.js chart container



import { Component, Input, Output, EventEmitter, OnInit, OnDestroy, OnChanges,
AfterViewInit, SimpleChanges, ChangeDetectionStrategy, ChangeDetectorRef, ElementRef,
ViewChild, HostListener } from '@angular/core'; import { CommonModule } from
'@angular/common'; import { Subject } from 'rxjs'; import { ButtonModule } from
'primeng/button';



import { ChartData, ChartDataSource } from './chart-data.interface';



// Import Plotly.js minified distribution import * as Plotly from 'plotly.js-dist-
min';



/**



  • Extended HTMLElement interface for Plotly charts

 *

  • Plotly augments the DOM element with chart-specific methods and properties



  • after calling Plotly.newPlot().

 */

interface PlotlyHTMLElement extends HTMLElement { on(event: string, callback: (data:
any) => void): void; data?: any[]; layout?: any; }



/**
```

```
  • Base Chart Component
*
  • Reusable Plotly.js chart container that works with any ChartDataSource.

  • Handles chart rendering, resizing, click events, and memory cleanup.
*
  • @example

  •
```

html

- <app-base-chart
- [dataSource]="manufacturerChartSource"
- [statistics]="statistics$ | async"
- [highlights]="highlights$ | async"
- (chartClick)="onChartClick($event)">
- </app-base-chart>

```
 /
@Component({ selector: 'app-base-chart', templateUrl: './base-chart.component.html',
styleUrls: ['./base-chart.component.scss'], changeDetection:
ChangeDetectionStrategy.OnPush }) export class BaseChartComponent implements OnInit,
AfterViewInit, OnDestroy, OnChanges {



/**



  • Reference to the chart container div where Plotly renders


   */
@ViewChild('chartContainer', { static: false }) chartContainer!:
ElementRef<HTMLDivElement>;



/**



  • Chart data source implementation (required)


   */
@Input() dataSource!: ChartDataSource;



/**



  • Statistics data to visualize


   */
@Input() statistics: any | null = null;



/**
```

```
   • Highlight filters for visual emphasis

    */
@Input() highlights: any = {};

/**

   • Currently selected value to highlight

    */
@Input() selectedValue: string | null = null;

/**

   • Whether to hide the chart title

    */
@Input() hideTitle = false;

/**

   • Whether to show pop-out button in mode bar

    */
@Input() canPopOut = false;

/**
```

```
   • Emits when user clicks on a chart element

    */
@Output() chartClick = new EventEmitter<{ value: string; isHighlightMode: boolean; }
>();



/**



   • Emits when user clicks the pop-out button

    */
@Output() popOutClick = new EventEmitter<void>();



/**



   • Whether highlight mode is active (h key held)

    */
isHighlightModeActive = false;



/**



   • Chart title from data source

    */
chartTitle = '';



/**
```

```
  • Error state flag

   */
hasError = false;



/**



  • Error message for display

   */
errorMessage = '';



private destroy$ = new Subject<void>(); private plotlyElement: PlotlyHTMLElement |
null = null;



constructor(private cdr: ChangeDetectorRef) {}



// ============================================================================ //
Lifecycle Hooks //
============================================================================



ngOnInit(): void { if (!this.dataSource) { console.error('BaseChartComponent:
dataSource is required'); } this.chartTitle = this.dataSource?.getTitle() ||
'Chart'; }



ngAfterViewInit(): void { // Render chart after view is ready this.renderChart(); }
```

```
ngOnChanges(changes: SimpleChanges): void { // Re-render when inputs change (after
initial render) if (this.chartContainer) { this.renderChart(); } }



ngOnDestroy(): void { this.destroy$.next(); this.destroy$.complete();



// Clean up Plotly chart to prevent memory leaks if (this.plotlyElement)
{ Plotly.purge(this.plotlyElement); } }



// ============================================================================ //
Event Handlers //
==============================================================================



/**



  • Handle window resize events


   */
@HostListener('window:resize') onWindowResize(): void { if (this.plotlyElement)
{ Plotly.Plots.resize(this.plotlyElement); } }



/**



  • Enable highlight mode when h key is pressed


   */
@HostListener('document:keydown.h') onHighlightKeyDown(): void
{ this.isHighlightModeActive = true; this.cdr.markForCheck(); }



/**
```

```
  • Disable highlight mode when h key is released

   */
@HostListener('document:keyup.h') onHighlightKeyUp(): void
{ this.isHighlightModeActive = false; this.cdr.markForCheck(); }



/**



  • Retry rendering after an error

   */
retryRender(): void { this.hasError = false; this.errorMessage = '';
this.cdr.markForCheck(); setTimeout(() => this.renderChart(), 0); }



// ============================================================================ //
Private Methods //
==============================================================================



/**



  • Render the Plotly chart with error boundary

   */
private renderChart(): void { if (!this.chartContainer || !this.dataSource)
{ return; }



try { this.hasError = false; this.errorMessage = '';
```

```
const element = this.chartContainer.nativeElement; const containerWidth =
element.clientWidth;



// Transform data using the data source const chartData =
this.dataSource.transform( this.statistics, this.highlights, this.selectedValue,
containerWidth );



if (!chartData) { // No data - clear chart if (this.plotlyElement)
{ Plotly.purge(this.plotlyElement); this.plotlyElement = null; } return; }



// Configure layout with title const layout: Record<string, any> =
{ ...chartData.layout }; if (!this.hideTitle && this.chartTitle) { layout['title'] =
{ text: this.chartTitle, font: { size: 14, color: '#ffffff' }, x: 0.01, xanchor:
'left', y: 0.97, yanchor: 'top' }; layout['margin'] = { ...layout['margin'], t:
(layout['margin']?.t || 30) + 20 }; }



// Plotly configuration const config: Partial<any> = { responsive: true,
displayModeBar: true, displaylogo: false, scrollZoom: false, modeBarButtonsToRemove:
['sendDataToCloud', 'lasso2d'] };



// Render or update chart if (this.plotlyElement) { Plotly.react(this.plotlyElement,
chartData.traces, layout, config); } else { Plotly.newPlot(element, chartData.traces,
layout, config) .then((gd: PlotlyHTMLElement) => { this.plotlyElement = gd;
this.attachEventHandlers(gd); }) .catch((err: Error) => { this.handleRenderError(err,
'Failed to create chart'); }); } } catch (err) { this.handleRenderError(err as Error,
'Chart rendering failed'); } }



/**



  • Attach Plotly event handlers


   */
private attachEventHandlers(gd: PlotlyHTMLElement): void { gd.on('plotly_click',
(data: any) => { try { const clickedValue = this.dataSource.handleClick(data); if
(clickedValue) { this.chartClick.emit({ value: clickedValue, isHighlightMode:
```

```
this.isHighlightModeActive }); } } catch (err) { console.error('[BaseChart] Click
handler error:', err); } });



gd.on('plotly_selected', (data: any) => { try { const selectedValue =
this.dataSource.handleClick(data); if (selectedValue) { this.chartClick.emit({ value:
selectedValue, isHighlightMode: this.isHighlightModeActive }); } } catch (err)
{ console.error('[BaseChart] Selection handler error:', err); } }); }



/**



  • Handle render errors with user-friendly fallback


   */
private handleRenderError(err: Error, context: string): void
{ console.error( [BaseChart] ${context}: , err); this.hasError = true;
this.errorMessage = ${context}: ${err.message || 'Unknown error'} ;
this.cdr.markForCheck();



if (this.plotlyElement) { try { Plotly.purge(this.plotlyElement); } catch { // Ignore
purge errors } this.plotlyElement = null; } } }
```

Step 801.4: Create the Base Chart Component Template

Create the file `src/app/framework/components/base-chart/base-chart.component.html`:

```
<!-- src/app/framework/components/base-chart/base-chart.component.html -->

<!-- VERSION 1 (Section 801) - Chart container template -->



<div class="chart-wrapper"> <!-- Error boundary fallback --> <div *ngIf="hasError"
class="chart-error"> <div class="error-content"> <i class="pi pi-exclamation-
triangle"></i> <p class="error-title">Chart failed to render</p> <p class="error-
message">{{ errorMessage }}</p> <button pButton type="button" label="Retry" icon="pi
pi-refresh" class="p-button-outlined p-button-sm" (click)="retryRender()"> </button>
</div> </div>



<!-- Plotly chart container (hidden when error) --> <div #chartContainer class="chart-
container" [hidden]="hasError"></div>



<!-- No data message --> <div *ngIf="!statistics && !hasError" class="no-data-
message"> <p>No data available</p> </div> </div>
```

Step 801.5: Create the Base Chart Component Styles

Create the file `src/app/framework/components/base-chart/base-chart.component.scss`:

```
// src/app/framework/components/base-chart/base-chart.component.scss

// VERSION 1 (Section 801) - Chart container styles



.chart-wrapper { width: 100%; height: 100%; display: flex; flex-direction: column; }



.chart-container { flex: 1; min-height: 300px; width: 100%; overflow: hidden; }



.no-data-message { display: flex; justify-content: center; align-items: center;
height: 300px; color: var(--text-color-secondary); font-style: italic; }



.chart-error { display: flex; justify-content: center; align-items: center; min-
height: 300px; background-color: var(--surface-ground, #f8f9fa); border: 1px dashed
var(--red-400, #f87171); border-radius: 6px; padding: 2rem;



.error-content { text-align: center; max-width: 400px;



.pi-exclamation-triangle { font-size: 2.5rem; color: var(--red-500, #ef4444); margin-
bottom: 1rem; }



.error-title { font-size: 1.1rem; font-weight: 600; color: var(--text-color); margin:
0 0 0.5rem 0; }



.error-message { font-size: 0.875rem; color: var(--text-color-secondary); margin: 0 0
1rem 0; word-break: break-word; }



button { margin-top: 0.5rem; } } }
```

Step 801.6: Create the Module Export

Create the file `src/app/framework/components/base-chart/index.ts`:

```
// src/app/framework/components/base-chart/index.ts

// VERSION 1 (Section 801) - Barrel export



export { BaseChartComponent } from './base-chart.component'; export { ChartData,
ChartDataSource } from './chart-data.interface';
```

Step 801.7: Register in Framework Module

Open `src/app/framework/framework.module.ts` and add the component:

```
// src/app/framework/framework.module.ts

// VERSION 2 (Section 801) - Add BaseChartComponent // Replaces VERSION 1 from Section
315



import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/
common'; import { ButtonModule } from 'primeng/button';



import { BaseChartComponent } from './components/base-chart/base-chart.component';



@NgModule({ declarations: [ BaseChartComponent ], imports: [ CommonModule,
ButtonModule ], exports: [ BaseChartComponent ] }) export class FrameworkModule {}
```

Verification

1. Build the Application

```
$ cd ~/projects/vvroom
$ ng build
```

Expected: Build succeeds with no errors related to BaseChartComponent.

2. Check TypeScript Compilation

```
$ npx tsc --noEmit
```

Expected: No type errors.

3. Verify Module Registration

```
$ grep -r "BaseChartComponent" src/app/framework/
```

Expected output shows the component in the module's declarations and exports.

4. Manual Inspection

Open `src/app/framework/components/base-chart/base-chart.component.ts` and verify:

- The `ChartDataSource` abstract class is imported
- All lifecycle hooks are implemented
- The `@HostListener` decorators are present for resize and keyboard events

---

Common Problems

Symptom	Cause	Solution
`Cannot find module 'plotly.js-dist-min'`	Package not installed	Run `npm install plotly.js-dist-min --save`
`Property 'Plots' does not exist on type 'typeof Plotly'`	TypeScript type issue	The `any` type on Plotly import handles this; verify import syntax
Chart doesn't resize on window resize	`@HostListener` not triggering	Ensure component is in change detection tree
Memory leak warnings in console	Plotly not cleaned up	Verify `Plotly.purge()` is called in `ngOnDestroy`
Chart not rendering	Container has zero dimensions	Add `min-height: 300px` to the chart container

---

Key Takeaways

- **The Data Source Pattern separates concerns** - The component handles rendering; data sources handle transformation
- **Plotly requires explicit cleanup** - Always call `Plotly.purge()` in `ngOnDestroy` to prevent memory leaks
- **Abstract classes define contracts** - `ChartDataSource` ensures all chart types implement required methods

---

Acceptance Criteria

- [ ] `plotly.js-dist-min` package is installed and listed in `package.json`
- [ ] `ChartDataSource` abstract class defines all required methods
- [ ] `BaseChartComponent` renders a container div for Plotly
- [ ] Window resize triggers chart resize via `@HostListener`
- [ ] `h` key toggles highlight mode
- [ ] Error boundary displays user-friendly message on render failure
- [ ] `ngOnDestroy` calls `Plotly.purge()` for cleanup

- [ ] Component is registered in `FrameworkModule`
- [ ] `ng build` completes with no errors

Next Step

Proceed to `802-base-picker-component.md` to build the configuration-driven multi-select picker component.

802: Base Picker Component

Status: Planning **Depends On:** 205-picker-config-interface, 301-url-state-service, 306-resource-management-service, 311-picker-config-registry **Blocks:** 806-query-panel-component

---

Learning Objectives

After completing this section, you will:

- Understand the configuration-driven approach to building reusable UI components
- Know how to synchronize component state with URL parameters bidirectionally
- Be able to implement server-side pagination with selection persistence across pages

---

Objective

Build a configuration-driven multi-select table component that wraps PrimeNG Table with selection management and URL synchronization. The picker loads data from an API, displays it in a paginated table, allows multi-row selection, and emits selected items for URL parameter updates.

Why

Many data discovery applications need "picker" interfaces: tables where users select one or more items to filter results. For example, selecting specific vehicle models, manufacturers, or years.

Without a reusable picker component, each domain would require a custom implementation. The **Base Picker Component** eliminates this duplication by accepting configuration that describes:

- Which API endpoint to call
- Which columns to display

- How to serialize/deserialize selections to URL parameters

This is the Phase 8 pattern in action: **Generic components + specific configuration = infinite reusability**.

Angular Style Guide References

- Style 03-01: Use single responsibility principle
- Style 05-04: Put logic in the component class

URL-First Architecture Reference

The picker maintains bidirectional sync with URL parameters:

- **URL to Picker**: When URL contains selection parameters, the picker hydrates those selections
- **Picker to URL**: When user clicks "Apply", the component emits a selection event that the parent uses to update the URL

This ensures selections survive page refreshes and are shareable via URL.

What

Step 802.1: Create the Picker State Interface

Create the file `src/app/framework/components/base-picker/picker-state.interface.ts`:

```
// src/app/framework/components/base-picker/picker-state.interface.ts
// VERSION 1 (Section 802) - Picker component state


/**


  • Internal state for BasePickerComponent


 *


  • Tracks loaded data, pagination, selection, and loading/error states.


 */
export interface PickerState<T> { /**


  • Currently loaded data rows (current page only)


   */
data: T[];


/**


  • Total count of records (for pagination)


   */
totalCount: number;


/**


  • Current page index (0-based)
```

```
   */
currentPage: number;

/**

  • Number of rows per page

   */
pageSize: number;

/**

  • Current sort field (if any)

   */
sortField?: string;

/**

  • Sort order: 1 = ascending, -1 = descending

   */
sortOrder: number;

/**

  • Current search/filter term
```

```
   */
searchTerm: string;

/**

  • Loading state flag

   */
loading: boolean;

/**

  • Error object (if any)

   */
error: Error | null;

/**

  • Whether initial data has been loaded

   */
dataLoaded: boolean;

/**

  • Set of selected row keys (for O(1) lookup)
```

```
   */
selectedKeys: Set<string>;



/**



  • Array of selected row objects

   */
selectedItems: T[];



/**



  • Keys pending hydration from URL (before data loads)

   */
pendingHydration: string[]; }



/**



  • Get default picker state

 */
export function getDefaultPickerState<T>(pageSize = 20): PickerState<T> { return
{ data: [], totalCount: 0, currentPage: 0, pageSize, sortField: undefined, sortOrder:
1, searchTerm: '', loading: false, error: null, dataLoaded: false, selectedKeys: new
Set<string>(), selectedItems: [], pendingHydration: [] }; }
```

Step 802.2: Create the Base Picker Component TypeScript

Create the file `src/app/framework/components/base-picker/base-picker.component.ts`:

```
// src/app/framework/components/base-picker/base-picker.component.ts

// VERSION 1 (Section 802) - Configuration-driven multi-select table



import { AfterViewInit, ChangeDetectionStrategy, ChangeDetectorRef, Component,
ElementRef, EventEmitter, Input, OnDestroy, OnInit, Output } from '@angular/core';
import { CommonModule } from '@angular/common'; import { FormsModule } from '@angular/
forms'; import { Subject } from 'rxjs'; import { distinctUntilChanged, map,
takeUntil } from 'rxjs/operators';



import { TableModule } from 'primeng/table'; import { ButtonModule } from 'primeng/
button'; import { CheckboxModule } from 'primeng/checkbox'; import { InputTextModule }
from 'primeng/inputtext'; import { SkeletonModule } from 'primeng/skeleton'; import
{ MessageModule } from 'primeng/message';



import { PickerConfig, PickerSelectionEvent, PickerApiParams } from '../../models/
picker-config.interface'; import { PickerConfigRegistry } from '../../services/picker-
config-registry.service'; import { UrlStateService } from '../../services/url-
state.service'; import { ResourceManagementService } from '../../services/resource-
management.service'; import { PickerState, getDefaultPickerState } from './picker-
state.interface';



/**



  • Base Picker Component

 *

  • Configuration-driven multi-select table with:



  • - Server-side pagination and sorting



  • - Search filtering
```

```
  • - Selection persistence across pages


  • - URL parameter synchronization

 *

  • @template T - The data model type for table rows

 *

  • @example


  •
```

html

- <!-- Using config from registry -->
- <app-base-picker
- [configId]="'vehicle-picker'"
- (selectionChange)="onSelectionChange($event)">
- </app-base-picker>

*

- <!-- Using direct config object -->
- <app-base-picker
- [config]="vehiclePickerConfig"
- (selectionChange)="onSelectionChange($event)">
- </app-base-picker>

```
 /
@Component({ selector: 'app-base-picker', templateUrl: './base-picker.component.html',
styleUrls: ['./base-picker.component.scss'], changeDetection:
ChangeDetectionStrategy.OnPush }) export class BasePickerComponent<T> implements
OnInit, OnDestroy, AfterViewInit {



/**



  • Picker configuration ID (loads from registry)



  • Either configId or config must be provided.

   */
@Input() configId?: string;



/**



  • Direct picker configuration object



  • Either configId or config must be provided.

   */
@Input() config?: PickerConfig<T>;



/**



  • Emits when user clicks "Apply" with selected items

   */
```

```
@Output() selectionChange = new EventEmitter<PickerSelectionEvent<T>>();



/**



  • Current picker state


   */
state: PickerState<T> = getDefaultPickerState();



/**



  • Active configuration (resolved from configId or config)


   */
activeConfig?: PickerConfig<T>;



private destroy$ = new Subject<void>();



constructor( private registry: PickerConfigRegistry, private urlState:
UrlStateService, private cdr: ChangeDetectorRef, private elementRef: ElementRef,
private resourceService: ResourceManagementService<any, any, any> ) {}



// ============================================================================ //
Lifecycle //
============================================================================



ngOnInit(): void { this.loadConfiguration();
```

```
if (!this.activeConfig) { throw new Error('BasePickerComponent requires either
configId or config input'); }



this.initializeState(); this.subscribeToUrlChanges(); this.loadData(); }



ngAfterViewInit(): void { this.syncPaginatorWidth(); }



ngOnDestroy(): void { this.destroy$.next(); this.destroy$.complete(); }



// ========================================================================== //
Configuration //
==========================================================================



private loadConfiguration(): void { if (this.config) { this.activeConfig =
this.config; } else if (this.configId) { this.activeConfig =
this.registry.get<T>(this.configId); } }



private initializeState(): void { const pageSize =
this.activeConfig!.pagination.defaultPageSize || 20; this.state =
getDefaultPickerState<T>(pageSize); }



// ========================================================================== // URL
Synchronization //
==========================================================================



private subscribeToUrlChanges(): void { const urlParam =
this.activeConfig!.selection.urlParam;



this.resourceService.filters$ .pipe( map(filters => (filters as any)[urlParam] ||
null), distinctUntilChanged(), takeUntil(this.destroy$) ) .subscribe(filterValue =>
{ if (filterValue) { this.hydrateFromUrl(String(filterValue)); } else { // Clear
selections this.state.selectedKeys = new Set<string>(); this.state.selectedItems = [];
```

```
this.state.pendingHydration = []; this.state.data = [...this.state.data];
this.cdr.detectChanges(); } }); }



private hydrateFromUrl(urlValue: string): void { const config = this.activeConfig!;
const partialItems = config.selection.deserializer(urlValue); const keyGenerator =
config.selection.keyGenerator || config.row.keyGenerator; const keys =
partialItems.map(item => keyGenerator(item as T));



if (this.state.dataLoaded) { this.hydrateSelections(keys); } else
{ this.state.pendingHydration = keys; }



this.cdr.markForCheck(); }



private hydrateSelections(keys: string[]): void { const config = this.activeConfig!;
this.state.selectedKeys = new Set<string>(keys);



// Preserve items from other pages const existingItemsByKey = new Map<string, T>();
this.state.selectedItems.forEach(item => { const key = config.row.keyGenerator(item);
existingItemsByKey.set(key, item); });



// Build new selectedItems array const newSelectedItems: T[] = []; keys.forEach(key =>
{ const itemInCurrentPage = this.state.data.find( row => config.row.keyGenerator(row)
=== key );



if (itemInCurrentPage) { newSelectedItems.push(itemInCurrentPage); } else if
(existingItemsByKey.has(key))
{ newSelectedItems.push(existingItemsByKey.get(key)!); } });



this.state.selectedItems = newSelectedItems; this.cdr.markForCheck(); }
```

```
// ============================================================================ //
Data Loading //
==============================================================================



private loadData(): void { const config = this.activeConfig!; this.state.loading =
true; this.state.error = null; this.cdr.markForCheck();



const params: PickerApiParams = { page: this.state.currentPage, size:
this.state.pageSize, search: this.state.searchTerm || undefined, sortField:
this.state.sortField, sortOrder: this.state.sortOrder };



const apiParams = config.api.paramMapper ? config.api.paramMapper(params) : params;



config.api.fetchData(apiParams) .pipe(takeUntil(this.destroy$)) .subscribe({ next:
response => { const transformed = config.api.responseTransformer(response);
this.state.data = transformed.results; this.state.totalCount = transformed.total;
this.state.loading = false; this.state.dataLoaded = true;



// Hydrate pending selections if (this.state.pendingHydration.length > 0)
{ this.hydrateSelections(this.state.pendingHydration); this.state.pendingHydration =
[]; }



this.cdr.markForCheck(); }, error: error => { this.state.loading = false;
this.state.error = error; this.cdr.markForCheck(); } }); }



// ============================================================================ //
Selection Handlers //
==============================================================================



onRowSelectionChange(row: T, checked: boolean): void { const key =
this.activeConfig!.row.keyGenerator(row);
```

```
if (checked) { this.state.selectedKeys.add(key); this.state.selectedItems.push(row); }
else { this.state.selectedKeys.delete(key); this.state.selectedItems =
this.state.selectedItems.filter( item => this.activeConfig!.row.keyGenerator(item) !==
key ); }



this.cdr.markForCheck(); }



onSelectAll(checked: boolean): void { if (checked) { this.state.data.forEach(row =>
{ const key = this.activeConfig!.row.keyGenerator(row); if (!
this.state.selectedKeys.has(key)) { this.state.selectedKeys.add(key);
this.state.selectedItems.push(row); } }); } else { this.state.data.forEach(row =>
{ const key = this.activeConfig!.row.keyGenerator(row);
this.state.selectedKeys.delete(key); }); this.state.selectedItems =
this.state.selectedItems.filter(item => { const key =
this.activeConfig!.row.keyGenerator(item); return
this.state.selectedKeys.has(key); }); }



this.cdr.markForCheck(); }



isRowSelected(row: T): boolean { const key = this.activeConfig!.row.keyGenerator(row);
return this.state.selectedKeys.has(key); }



get allVisibleSelected(): boolean { if (this.state.data.length === 0) { return
false; } return this.state.data.every(row => this.isRowSelected(row)); }



// ========================================================================== //
Pagination & Sorting //
============================================================================



onLazyLoad(event: any): void { if (this.state.loading) { return; }
```

```
this.state.currentPage = event.first / event.rows; this.state.pageSize = event.rows;
this.state.sortField = event.sortField || undefined; this.state.sortOrder =
event.sortOrder || 1;



this.loadData(); }



onSearch(term: string): void { this.state.searchTerm = term; this.state.currentPage =
0; this.loadData(); }



// ============================================================================ //
Actions //
==============================================================================



applySelections(): void { const config = this.activeConfig!; const urlValue =
config.selection.serializer(this.state.selectedItems);



const event: PickerSelectionEvent<T> = { pickerId: config.id, selections:
this.state.selectedItems, selectedKeys: Array.from(this.state.selectedKeys),
urlValue };



this.selectionChange.emit(event); }



clearSelections(): void { this.state.selectedKeys = new Set<string>();
this.state.selectedItems = []; this.state.data = [...this.state.data];
this.cdr.markForCheck();



// Emit empty selection const event: PickerSelectionEvent<T> = { pickerId:
this.activeConfig!.id, selections: [], selectedKeys: [], urlValue: '' };
this.selectionChange.emit(event); }
```

```
// ============================================================================ //
Helpers //
==============================================================================



fieldToString(field: keyof T): string { return String(field); }



private syncPaginatorWidth(): void { const nativeEl = this.elementRef.nativeElement;
const table = nativeEl.querySelector('.p-datatable-table') as HTMLElement; const
paginator = nativeEl.querySelector('.p-paginator') as HTMLElement;



if (table && paginator) { paginator.style.width = ${table.offsetWidth}px; } } }
```

Step 802.3: Create the Base Picker Component Template

Create the file `src/app/framework/components/base-picker/base-picker.component.html`:

```
<!-- src/app/framework/components/base-picker/base-picker.component.html -->

<!-- VERSION 1 (Section 802) - Picker table template -->



<div *ngIf="activeConfig" class="picker-container"> <p-table
[value]="state.data" [loading]="state.loading" [lazy]="activeConfig.pagination.mode
===
'server'" [paginator]="true" [rows]="state.pageSize" [totalRecords]="state.totalCount"
[rowsPerPageOptions]="activeConfig.pagination.pageSizeOptions || [10, 20, 50,
100]" [showCurrentPageReport]="true" currentPageReportTemplate="Showing {first} to
{last} of {totalRecords} entries" (onLazyLoad)="onLazyLoad($event)" styleClass="p-
datatable-gridlines" responsiveLayout="scroll">



<!-- Caption with search and actions --> <ng-template pTemplate="caption"> <div
class="table-caption"> <!-- Search --> <div *ngIf="activeConfig.showSearch !== false"
class="picker-search"> <input type="text" pInputText
[placeholder]="activeConfig.searchPlaceholder ||
'Search...'" [value]="state.searchTerm" (input)="onSearch($any($event.target).value)"
class="search-input"> </div> <div class="table-actions"> <button pButton type="button"
label="Clear" icon="pi pi-times" class="p-button-outlined p-button-
secondary" (click)="clearSelections()" [disabled]="state.selectedItems.length === 0">
</button> <button pButton type="button" label="Apply" icon="pi pi-check" class="p-
button-primary" (click)="applySelections()" [disabled]="state.selectedItems.length ===
0"> </button> </div> </div> </ng-template>



<!-- Header --> <ng-template pTemplate="header"> <tr> <!-- Select All Checkbox --> <th
style="width: 3rem"> <p-checkbox
[ngModel]="allVisibleSelected" (onChange)="onSelectAll($any($event).checked)" [binary]
="true" [disabled]="state.data.length === 0"> </p-checkbox> </th> <!-- Column Headers
--> <th *ngFor="let col of activeConfig.columns" [pSortableColumn]="col.sortable ?
fieldToString(col.field) : ''" [ngStyle]="{ width: col.width, textAlign: col.align ||
'left' }"> {{ col.header }} <p-sortIcon
*ngIf="col.sortable" [field]="fieldToString(col.field)"> </p-sortIcon> </th> </tr> </
ng-template>



<!-- Body --> <ng-template pTemplate="body" let-row> <tr> <!-- Row Checkbox --> <td>
<p-checkbox [ngModel]="isRowSelected(row)" (onChange)="onRowSelectionChange(row,
$any($event).checked)" [binary]="true"> </p-checkbox> </td> <!-- Column Data --> <td
*ngFor="let col of activeConfig.columns" [ngStyle]="{ textAlign: col.align ||
'left' }"> {{ row[col.field] }} </td> </tr> </ng-template>
```

```
<!-- Empty Message --> <ng-template pTemplate="emptymessage"> <tr> <td
[attr.colspan]="activeConfig.columns.length + 1"> <div class="empty-message"> <i
class="pi pi-inbox empty-icon"></i> <p>No data available</p> </div> </td> </tr> </ng-
template>



<!-- Loading Body --> <ng-template pTemplate="loadingbody"> <tr *ngFor="let i of [1,
2, 3, 4, 5]"> <td [attr.colspan]="activeConfig.columns.length + 1"> <div
class="loading-row"> <p-skeleton width="100%" height="2rem"></p-skeleton> </div> </td>
</tr> </ng-template> </p-table>



<!-- Error Display --> <div *ngIf="state.error" class="picker-error"> <p-message
severity="error" [text]="state.error.message || 'An error occurred'"> </p-message> </
div> </div>
```

Step 802.4: Create the Base Picker Component Styles

Create the file `src/app/framework/components/base-picker/base-picker.component.scss`:

```
// src/app/framework/components/base-picker/base-picker.component.scss

// VERSION 1 (Section 802) - Picker table styles



.picker-container { display: flex; flex-direction: column; gap: 0; padding: 0;
background: var(--surface-card); border-radius: var(--border-radius); border: 1px
solid var(--surface-border); }



.picker-search { display: flex; align-items: center; gap: 0.5rem; flex: 1;



.search-input { flex: 1; width: 100%; } }



.table-caption { display: flex; justify-content: space-between; align-items: center;
padding: 0.5rem 1rem; gap: 1rem; background-color: var(--surface-50);



@media (max-width: 768px) { flex-direction: column; align-items: flex-start; } }



.table-actions { display: flex; gap: 0.5rem; white-space: nowrap; }



.empty-message { display: flex; flex-direction: column; align-items: center; justify-
content: center; padding: 3rem 1rem; color: var(--text-color-secondary);



.empty-icon { font-size: 3rem; margin-bottom: 1rem; opacity: 0.5; }



p { margin: 0; font-size: 1rem; } }



.loading-row { padding: 0.5rem; }
```

```
.picker-error { margin-top: 1rem; }



// PrimeNG table overrides ::ng-deep { .p-datatable { .p-datatable-thead > tr > th
{ background: var(--surface-section); color: var(--text-color); font-weight: 600;
padding: 0.75rem 1rem; }



.p-datatable-tbody > tr { background: var(--surface-card); color: var(--text-color);



&:hover { background: var(--surface-hover); }



> td { padding: 0.75rem 1rem; } }



.p-paginator { background: var(--surface-section); color: var(--text-color); border:
none; padding: 0.5rem 1rem; } } }
```

Step 802.5: Create the Module Export

Create the file `src/app/framework/components/base-picker/index.ts`:

```
// src/app/framework/components/base-picker/index.ts

// VERSION 1 (Section 802) - Barrel export



export { BasePickerComponent } from './base-picker.component'; export { PickerState,
getDefaultPickerState } from './picker-state.interface';
```

Step 802.6: Register in Framework Module

Open `src/app/framework/framework.module.ts` and add the component:

```
// src/app/framework/framework.module.ts

// VERSION 3 (Section 802) - Add BasePickerComponent // Replaces VERSION 2 from
Section 801



import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/
common'; import { FormsModule } from '@angular/forms'; import { ButtonModule } from
'primeng/button'; import { TableModule } from 'primeng/table'; import
{ CheckboxModule } from 'primeng/checkbox'; import { InputTextModule } from 'primeng/
inputtext'; import { SkeletonModule } from 'primeng/skeleton'; import
{ MessageModule } from 'primeng/message';



import { BaseChartComponent } from './components/base-chart/base-chart.component';
import { BasePickerComponent } from './components/base-picker/base-picker.component';



@NgModule({ declarations: [ BaseChartComponent, BasePickerComponent ], imports:
[ CommonModule, FormsModule, ButtonModule, TableModule, CheckboxModule,
InputTextModule, SkeletonModule, MessageModule ], exports: [ BaseChartComponent,
BasePickerComponent ] }) export class FrameworkModule {}
```

Verification

1. Build the Application

```
$ cd ~/projects/vvroom

$ ng build
```

Expected: Build succeeds with no errors.

2. Check TypeScript Compilation

```
$ npx tsc --noEmit
```

Expected: No type errors.

3. Verify Dependencies

```
$ grep -E "TableModule|CheckboxModule" src/app/framework/framework.module.ts
```

Expected: Both modules are in the imports array.

4. Manual Inspection

Check `src/app/framework/components/base-picker/base-picker.component.ts`:

- `PickerConfig` is used for configuration
- Selection is tracked with both `Set<string>` and `T[]`
- Lazy loading is implemented for server-side pagination

Common Problems

Symptom	Cause	Solution
`No provider for PickerConfigRegistry`	Service not provided	Ensure `PickerConfigRegistry` is provided in root
Selections lost on page change	Hydration not preserving off-page items	Verify `hydrateSelections` merges with existing items
Checkbox state not updating	Change detection not triggered	Use `cdr.detectChanges()` instead of `markForCheck()`
Paginator width doesn't match table	`syncPaginatorWidth` not called	Verify it runs in `ngAfterViewInit`
`ExpressionChangedAfterItHasBeenCheckedError`	Mutating state during render	Move mutations to lifecycle hooks or use `setTimeout`

Key Takeaways

- **Configuration drives behavior** - The same component renders different data based on `PickerConfig`
- **Selections persist across pages** - Use a `Set` for O(1) lookup and an array for full item data
- **URL hydration requires two-phase approach** - Store pending keys until data loads, then match

---

Acceptance Criteria

- [ ] `BasePickerComponent` accepts `configId` or `config` input
- [ ] Table displays columns from configuration
- [ ] Server-side pagination triggers `loadData()` on page change
- [ ] Search input filters results (if enabled in config)
- [ ] Checkbox selection tracks items across pages
- [ ] "Apply" button emits `selectionChange` with serialized URL value
- [ ] "Clear" button resets all selections
- [ ] URL parameter changes hydrate selections
- [ ] Loading skeleton displays during data fetch
- [ ] Error message displays on API failure
- [ ] Component is registered in `FrameworkModule`
- [ ] `ng build` completes with no errors

Next Step

Proceed to `803-basic-results-table.md` to build the domain-agnostic data table for displaying search results.

803: Basic Results Table

Status: Planning **Depends On:** 204-table-config-interface, 301-url-state-service, 306-resource-management-service **Blocks:** 904-automobile-discover

---

Learning Objectives

After completing this section, you will:

- Understand how to bind a data table to reactive streams from a service
- Know how to implement server-side pagination and sorting via URL state
- Be able to create expandable row details using PrimeNG Table templates

---

Objective

Build a configuration-driven results table component that displays data from `ResourceManagementService`, supports server-side pagination and sorting, and shows expandable row details. This component is the primary way users view search results in the discover interface.

Why

The results table is the heart of any data discovery application. Users search, filter, and browse results in this table. However, different domains have different columns and data structures. The **Basic Results Table** solves this by:

- Reading column definitions from `DomainConfig.tableConfig`
- Binding to reactive streams (`results$` , `loading$` , `totalResults$`) from `ResourceManagementService`

- Emitting pagination and sort changes that update URL state

This keeps the component domain-agnostic while still providing rich functionality.

Angular Style Guide References

- Style 05-02: Use input properties for data binding
- Style 05-15: Provide a selector prefix

URL-First Architecture Reference

When users click pagination controls or column headers to sort, the table emits events that ultimately update URL parameters. The `ResourceManagementService` watches these URL changes and triggers new API requests. This maintains our URL-First principle: the URL is the single source of truth.

What

Step 803.1: Create the Basic Results Table Component TypeScript

Create the file `src/app/framework/components/basic-results-table/basic-results-table.component.ts`:

```
// src/app/framework/components/basic-results-table/basic-results-table.component.ts
// VERSION 1 (Section 803) - Configuration-driven results display



import { AfterViewInit, ChangeDetectionStrategy, ChangeDetectorRef, Component,
ElementRef, EventEmitter, Input, OnDestroy, OnInit, Output } from '@angular/core';
import { CommonModule } from '@angular/common'; import { Observable, Subject } from
'rxjs'; import { filter, takeUntil } from 'rxjs/operators';



import { TableModule } from 'primeng/table'; import { ButtonModule } from 'primeng/
button'; import { RippleModule } from 'primeng/ripple'; import { SkeletonModule } from
'primeng/skeleton';



import { DomainConfig } from '../../models/domain-config.interface'; import
{ ResourceManagementService } from '../../services/resource-management.service';
import { PopOutContextService } from '../../services/popout-context.service'; import
{ PopOutMessageType } from '../../models/popout.interface';



/**



  • Basic Results Table Component

 *

  • Displays search results in a paginated, sortable table with expandable rows.



  • Reads configuration from DomainConfig and data from ResourceManagementService.

 *

  • @template TFilters - Domain-specific filter model type



  • @template TData - Domain-specific data model type
```

```
  • @template TStatistics - Domain-specific statistics model type
*
  • @example
  •
```

html

- <app-basic-results-table
- [domainConfig]="automobileDomainConfig">
- </app-basic-results-table>

```
 /
@Component({ selector: 'app-basic-results-table', templateUrl: './basic-results-
table.component.html', styleUrls: ['./basic-results-table.component.scss'],
changeDetection: ChangeDetectionStrategy.OnPush }) export class
BasicResultsTableComponent<TFilters = any, TData = any, TStatistics = any> implements
OnInit, AfterViewInit, OnDestroy {



private readonly destroy$ = new Subject<void>();



/**



  • Domain configuration with table columns and settings

   */
@Input() domainConfig!: DomainConfig<TFilters, TData, TStatistics>;



/**



  • Emits when URL parameters should be updated (for pop-out sync)

   */
@Output() urlParamsChange = new EventEmitter<{ [key: string]: any }>();



/**



  • Track expanded rows by data key

   */
expandedRows: { [key: string]: boolean } = {};
```

```
/**


  • Reference to Object for template use


   */
Object = Object;



constructor( private readonly resourceService: ResourceManagementService<TFilters,
TData, TStatistics>, private readonly cdr: ChangeDetectorRef, private readonly
popOutContext: PopOutContextService, private readonly elementRef: ElementRef ) {}



// ============================================================================ //
Observable Streams //
==============================================================================



get filters$(): Observable<TFilters> { return this.resourceService.filters$; }



get results$(): Observable<TData[]> { return this.resourceService.results$; }



get totalResults$(): Observable<number> { return this.resourceService.totalResults$; }



get loading$(): Observable<boolean> { return this.resourceService.loading$; }



// ============================================================================ //
Computed Properties //
==============================================================================
```

```
/**


  • Calculate paginator first index from current filters


   */
get paginatorFirst(): number { const filters =
this.resourceService.getCurrentFilters() as Record<string, any>; const page =
filters['page'] || 1; const size = filters['size'] || 20; return (page - 1) * size; }



/**


  • Get current filters for template bindings


   */
get currentFilters(): Record<string, any> { return
this.resourceService.getCurrentFilters() as Record<string, any>; }



// ============================================================================ //
Template Helpers //
==============================================================================



/**


  • Get object keys for row expansion display


   */
getObjectKeys(obj: any): string[] { return Object.keys(obj); }



// ============================================================================ //
Lifecycle //
==============================================================================
```

```
ngOnInit(): void { if (!this.domainConfig) { throw new
Error('BasicResultsTableComponent requires domainConfig input'); }



// Pop-out window support: sync state from parent window if
(this.popOutContext.isInPopOut()) { this.popOutContext .getMessages$
() .pipe( filter(msg => msg.type === PopOutMessageType.STATE_UPDATE),
takeUntil(this.destroy$) ) .subscribe(message => { if (message.payload &&
message.payload.state)
{ this.resourceService.syncStateFromExternal(message.payload.state);
this.cdr.markForCheck(); } }); } }



ngAfterViewInit(): void { this.syncPaginatorWidth(); }



ngOnDestroy(): void { this.destroy$.next(); this.destroy$.complete(); }



// ========================================================================== //
Event Handlers //
==========================================================================



/**



  • Handle pagination change


   */
onPageChange(event: any): void { const page = event.first / event.rows + 1; const size
= event.rows;



if (this.popOutContext.isInPopOut()) { // In pop-out: emit for parent to handle
this.urlParamsChange.emit({ page, size }); } else { // In main window: update filters
directly const currentFilters = this.resourceService.getCurrentFilters() as
```

```
Record<string, any>; const newFilters = { ...currentFilters, page, size } as unknown
as TFilters; this.resourceService.updateFilters(newFilters); } }



/**



  • Handle sort change


   */

onSort(event: any): void { const sort = event.field; const sortDirection = event.order
=== 1 ? 'asc' : 'desc';



if (this.popOutContext.isInPopOut()) { this.urlParamsChange.emit({ sort,
sortDirection }); } else { const currentFilters =
this.resourceService.getCurrentFilters() as Record<string, any>; const newFilters =
{ ...currentFilters, sort, sortDirection } as unknown as TFilters;
this.resourceService.updateFilters(newFilters); } }



/**



  • Refresh current results


   */

refresh(): void { this.resourceService.refresh(); }



// ============================================================================ //
Private Methods //
==============================================================================



/**



  • Sync paginator width to match table width
```

```
   */
private syncPaginatorWidth(): void { const nativeEl = this.elementRef.nativeElement;
const table = nativeEl.querySelector('.p-datatable-table') as HTMLElement; const
paginator = nativeEl.querySelector('.p-paginator') as HTMLElement;



if (table && paginator) { paginator.style.width = ${table.offsetWidth}px; } } }
```

Step 803.2: Create the Basic Results Table Template

Create the file `src/app/framework/components/basic-results-table/basic-results-table.component.html`:

```
<!-- src/app/framework/components/basic-results-table/basic-results-
table.component.html -->

<!-- VERSION 1 (Section 803) - Results table template -->



<div class="basic-results-table-container"> <div class="table-container"> <p-table
[value]="(results$ | async) ||
[]" [columns]="domainConfig.tableConfig.columns" [dataKey]="$any(domainConfig.tableCon
fig.dataKey)" [loading]="(loading$ | async) ||
false" [lazy]="true" [paginator]="true" [rows]="currentFilters['size'] ||
20" [first]="paginatorFirst" [totalRecords]="(totalResults$ | async) ||
0" [rowsPerPageOptions]="domainConfig.tableConfig.rowsPerPageOptions || [10, 20,
50]" [styleClass]="domainConfig.tableConfig.styleClass ||
''" [responsiveLayout]="domainConfig.tableConfig.responsiveLayout ||
'scroll'" [expandedRowKeys]="expandedRows" (onPage)="onPageChange($event)" (onSort)="o
nSort($event)" [showCurrentPageReport]="true" [currentPageReportTemplate]="'Showing
{first} to {last} of {totalRecords} results'">



<!-- Caption with result count --> <ng-template pTemplate="caption"> <div
class="table-caption"> <span class="caption-count"> {{ (totalResults$ | async) || 0 }}
result(s) </span> </div> </ng-template>



<!-- Header --> <ng-template pTemplate="header" let-columns> <tr> <th
*ngIf="domainConfig.tableConfig.expandable" style="width: 3rem"></th> <th *ngFor="let
col of columns" [pSortableColumn]="col.sortable ? col.field :
undefined" [ngStyle]="{ 'width': col.width }"> {{ col.header }} <p-sortIcon
*ngIf="col.sortable" [field]="col.field"></p-sortIcon> </th> </tr> </ng-template>



<!-- Body --> <ng-template pTemplate="body" let-row let-columns="columns" let-
expanded="expanded"> <tr> <!-- Expand toggle --> <td
*ngIf="domainConfig.tableConfig.expandable"> <button type="button" pButton pRipple
[pRowToggler]="row" class="p-button-text p-button-rounded p-button-
plain" [icon]="expanded ? 'pi pi-chevron-down' : 'pi pi-chevron-right'"> </button> </
td>



<!-- Data columns --> <td *ngFor="let col of columns"> {{ row[col.field] }} </td> </
tr> </ng-template>
```

```
<!-- Row Expansion --> <ng-template pTemplate="rowexpansion" let-row> <tr> <td
[attr.colspan]="domainConfig.tableConfig.columns.length + 1"> <div class="row-
expansion"> <h3>Details</h3> <div class="expansion-grid"> <div *ngFor="let key of
getObjectKeys(row)" class="expansion-field"> <strong>{{ key }}:</strong>
{{ row[key] }} </div> </div> </div> </td> </tr> </ng-template>



<!-- Empty message --> <ng-template pTemplate="emptymessage"> <tr> <td
[attr.colspan]="domainConfig.tableConfig.columns.length +
(domainConfig.tableConfig.expandable ? 1 : 0)"> <div class="empty-message"> <i
class="pi pi-inbox" style="font-size: 3rem"></i> <p>No results found</p> <p
class="empty-hint">Try adjusting your filters</p> </div> </td> </tr> </ng-template>



<!-- Loading skeleton --> <ng-template pTemplate="loadingbody"> <tr *ngFor="let i of
[1, 2, 3, 4, 5]"> <td [attr.colspan]="domainConfig.tableConfig.columns.length +
(domainConfig.tableConfig.expandable ? 1 : 0)"> <p-skeleton width="100%"
height="2rem"></p-skeleton> </td> </tr> </ng-template> </p-table> </div> </div>
```

Step 803.3: Create the Basic Results Table Styles

Create the file `src/app/framework/components/basic-results-table/basic-results-table.component.scss`:

```
// src/app/framework/components/basic-results-table/basic-results-table.component.scss

// VERSION 1 (Section 803) - Results table styles



.basic-results-table-container { display: flex; flex-direction: column; gap: 1rem;
padding: 1rem;



.table-container { background: var(--surface-card); border-radius: 6px; padding: 1rem;



::ng-deep .p-datatable { .p-datatable-thead > tr > th, .p-datatable-tbody > tr > td
{ padding: 0.5rem 0.75rem; border: 1px solid var(--surface-border); }



.p-datatable-thead > tr > th { border-bottom-width: 2px; background: var(--surface-
section); font-weight: 600; }



.p-datatable-table { border: 1px solid var(--surface-border); }



.p-paginator { border: none; background: var(--surface-section); }



.p-datatable-table-container { overflow-x: auto; } }



.table-caption { display: flex; justify-content: flex-end; align-items: center;
padding: 0.5rem 1rem; background-color: var(--surface-50); border-bottom: 1px solid
var(--surface-border);



.caption-count { font-size: 0.875rem; color: var(--text-color-secondary); } }



.empty-message { text-align: center; padding: 3rem 1rem; color: var(--text-color-
secondary);
```

```
i { color: var(--text-color-secondary); opacity: 0.5; }


p { margin: 1rem 0 0.5rem; font-size: 1.125rem; }


.empty-hint { font-size: 0.875rem; margin: 0.5rem 0 0; } }


.row-expansion { padding: 1.5rem; background: var(--surface-ground); border-left: 3px
solid var(--primary-color);


h3 { margin: 0 0 1rem; font-size: 1.125rem; color: var(--text-color); }


.expansion-grid { display: grid; grid-template-columns: repeat(auto-fill,
minmax(200px, 1fr)); gap: 0;


.expansion-field { display: flex; flex-direction: column; gap: 0.25rem; padding:
0.75rem; border: 1px solid var(--surface-border);


strong { color: var(--text-color-secondary); font-size: 0.75rem; text-transform:
uppercase; } } } } } }
```

Step 803.4: Create the Module Export

Create the file `src/app/framework/components/basic-results-table/index.ts`:

```
// src/app/framework/components/basic-results-table/index.ts

// VERSION 1 (Section 803) - Barrel export



export { BasicResultsTableComponent } from './basic-results-table.component';
```

Step 803.5: Register in Framework Module

Open `src/app/framework/framework.module.ts` and add the component:

```
// src/app/framework/framework.module.ts

// VERSION 4 (Section 803) - Add BasicResultsTableComponent // Replaces VERSION 3 from
Section 802



import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/
common'; import { FormsModule } from '@angular/forms'; import { ButtonModule } from
'primeng/button'; import { TableModule } from 'primeng/table'; import
{ CheckboxModule } from 'primeng/checkbox'; import { InputTextModule } from 'primeng/
inputtext'; import { SkeletonModule } from 'primeng/skeleton'; import
{ MessageModule } from 'primeng/message'; import { RippleModule } from 'primeng/
ripple';



import { BaseChartComponent } from './components/base-chart/base-chart.component';
import { BasePickerComponent } from './components/base-picker/base-picker.component';
import { BasicResultsTableComponent } from './components/basic-results-table/basic-
results-table.component';



@NgModule({ declarations: [ BaseChartComponent, BasePickerComponent,
BasicResultsTableComponent ], imports: [ CommonModule, FormsModule, ButtonModule,
TableModule, CheckboxModule, InputTextModule, SkeletonModule, MessageModule,
RippleModule ], exports: [ BaseChartComponent, BasePickerComponent,
BasicResultsTableComponent ] }) export class FrameworkModule {}
```

Verification

1. Build the Application

```
$ cd ~/projects/vvroom
$ ng build
```

Expected: Build succeeds with no errors.

2. Check TypeScript Compilation

```
$ npx tsc --noEmit
```

Expected: No type errors.

3. Verify Observable Bindings

Check `src/app/framework/components/basic-results-table/basic-results-table.component.ts`:

- `results$`, `loading$`, `totalResults$` are getter properties
- They delegate to `ResourceManagementService`

4. Verify Template Bindings

Check `src/app/framework/components/basic-results-table/basic-results-table.component.html`:

- `async` pipe is used with all observables
- Null coalescing (`|| []`, `|| 0`, `|| false`) handles initial null values

---

Common Problems

Symptom	Cause	Solution
Table shows no data	`results$` returning empty array	Verify `ResourceManagementS` has data loaded
Pagination not working	`[lazy]="true"` without `onLazyLoad` handler	This component uses `onPageChange` instead; e updates filters
Sort not persisting on refresh	Sort params not in URL	Verify `onSort` updates UR
Expandable rows not working	`dataKey` not set	Ensure `tableConfig.da` is set in domain config
`ExpressionChangedAfterItHasBeenCheckedError`	Calling `getCurrentFilters()` in template	Use computed property instea

Key Takeaways

- **Reactive binding with async pipe** - The template subscribes to observables; no manual subscription management needed
- **Lazy loading enables server-side operations** - The table delegates pagination/sorting to the service
- **Pop-out support is built-in** - The same component works in both main window and pop-out windows

---

Acceptance Criteria

- [ ] `BasicResultsTableComponent` accepts `domainConfig` input
- [ ] Table columns render from `tableConfig.columns`
- [ ] Pagination controls trigger `onPageChange` which updates filters
- [ ] Column headers trigger `onSort` which updates filters
- [ ] Loading skeleton displays while `loading$` is true
- [ ] Empty message displays when `results$` is empty
- [ ] Expandable rows show all object properties (if `tableConfig.expandable` is true)
- [ ] Result count displays in caption

- [ ] Pop-out window receives state updates via `PopOutContextService`
- [ ] Component is registered in `FrameworkModule`
- [ ] `ng build` completes with no errors

Next Step

Proceed to `804-statistics-panel-component.md` to build the chart grid container component.

804: Statistics Panel Component

Status: Planning **Depends On:** 801-base-chart-component, 306-resource-management-service, 651-654 (Chart Data Sources) **Blocks:** 904-automobile-discover

---

Learning Objectives

After completing this section, you will:

- Understand how to compose multiple chart components into a single panel
- Know how to implement drag-and-drop reordering with Angular CDK
- Be able to coordinate chart clicks with URL state updates

Objective

Build a statistics panel component that renders multiple charts in a draggable grid layout. This component composes `BaseChartComponent` instances and handles chart ordering, pop-out functionality, and click event coordination.

Why

Data discovery applications often display multiple charts showing different statistical views: distributions by manufacturer, year, body type, etc. Rather than hard-coding chart layouts, we build a **Statistics Panel** that:

- Accepts an array of chart IDs or reads from `DomainConfig.chartDataSources`
- Renders each chart using `BaseChartComponent`
- Allows users to reorder charts via drag-and-drop
- Coordinates click events with URL state

The Angular CDK (Component Dev Kit) provides drag-and-drop functionality without requiring a full component library.

Angular Style Guide References

- Style 03-02: Use delegation over inheritance
- Style 05-02: Use input properties for data binding

URL-First Architecture Reference

When a user clicks on a chart element (e.g., a bar representing "Toyota"), the click flows:

- `BaseChartComponent` emits `chartClick` event
- `StatisticsPanelComponent` receives the event and gets the data source
- Data source's `toUrlParams()` converts the click to URL parameters
- URL state is updated, triggering new API requests

This maintains our URL-First principle while keeping chart interaction logic in data sources.

What

Step 804.1: Install Angular CDK

The Angular CDK provides drag-and-drop functionality:

```
$ cd ~/projects/vvroom
$ npm install @angular/cdk --save
```

Verify installation:

```
$ grep "@angular/cdk" package.json
"@angular/cdk": "^13.3.0",
```

Step 804.2: Create the Statistics Panel Component TypeScript

Create the file `src/app/framework/components/statistics-panel/statistics-panel.component.ts`:

```
// src/app/framework/components/statistics-panel/statistics-panel.component.ts
// VERSION 1 (Section 804) - Chart grid with drag-drop reordering


import { ChangeDetectionStrategy, ChangeDetectorRef, Component, EventEmitter, Input,
OnDestroy, OnInit, Output } from '@angular/core'; import { CommonModule } from
'@angular/common'; import { Observable, Subject } from 'rxjs'; import { takeUntil }
from 'rxjs/operators';


import { DragDropModule, CdkDragDrop, moveItemInArray } from '@angular/cdk/drag-drop';


import { DomainConfig } from '../../models/domain-config.interface'; import
{ ChartDataSource } from '../base-chart/chart-data.interface'; import
{ ResourceManagementService } from '../../services/resource-management.service';
import { UrlStateService } from '../../services/url-state.service'; import
{ PopOutContextService } from '../../services/popout-context.service'; import
{ PopOutMessageType } from '../../models/popout.interface'; import
{ DomainConfigRegistry } from '../../services/domain-config-registry.service';


/**


  • Statistics Panel Component

 *

  • Renders multiple charts in a CDK drag-drop grid.


  • Charts can be reordered by dragging.

 *

  • @example
```

```
•
```

html

```
• <app-statistics-panel
• [domainConfig]="domainConfig"
• [chartIds]="['manufacturer', 'year']"
• (chartPopOut)="onChartPopOut($event)"
• (chartClicked)="onChartClick($event)">
• </app-statistics-panel>
```

```
 /
@Component({ selector: 'app-statistics-panel', templateUrl: './statistics-
panel.component.html', styleUrls: ['./statistics-panel.component.scss'],
changeDetection: ChangeDetectionStrategy.OnPush }) export class
StatisticsPanelComponent implements OnInit, OnDestroy {



private readonly destroy$ = new Subject<void>();



/**



  • Domain configuration with chart data sources


   */
@Input() domainConfig!: DomainConfig<any, any, any>;



/**



  • Optional subset of chart IDs to display



  • If not provided, all charts from domainConfig.chartDataSources are shown


   */
@Input() chartIds?: string[];



/**



  • Function to check if a chart is popped out
```

```
   • Provided by parent component

    */
@Input() isPanelPoppedOut: (panelId: string) => boolean = () => false;



/**

   • Emits when user clicks the pop-out button on a chart

    */
@Output() chartPopOut = new EventEmitter<string>();



/**

   • Emits when user clicks on a chart element

    */
@Output() chartClicked = new EventEmitter<{ event: { value: string; isHighlightMode:
boolean }; dataSource: ChartDataSource; }>();



/**

   • Ordered list of chart IDs for the grid

    */
chartOrder: string[] = [];



constructor( private readonly resourceService: ResourceManagementService<any, any,
any>, private readonly urlState: UrlStateService, private readonly popOutContext:
```

```
PopOutContextService, private readonly cdr: ChangeDetectorRef, private readonly
domainRegistry: DomainConfigRegistry ) {}



// ============================================================================ //
Observable Streams //
==============================================================================



get statistics$(): Observable<any | undefined> { return
this.resourceService.statistics$; }



get highlights$(): Observable<any> { return this.resourceService.highlights$; }



/**



  • Check if running in a pop-out window


   */
get isInPopOut(): boolean { return this.popOutContext.isInPopOut(); }



// ============================================================================ //
Lifecycle //
==============================================================================



ngOnInit(): void { // If domainConfig not provided (e.g., in pop-out), get from
registry if (!this.domainConfig) { this.domainConfig =
this.domainRegistry.getActive(); }



// Initialize chart order if (this.chartIds && this.chartIds.length > 0)
{ this.chartOrder = this.chartIds; } else if (this.domainConfig.chartDataSources)
{ this.chartOrder = Object.keys(this.domainConfig.chartDataSources); } }
```

```
ngOnDestroy(): void { this.destroy$.next(); this.destroy$.complete(); }



// ============================================================================ //
Event Handlers //
==============================================================================



/**



  • Handle chart drag-drop to reorder

   */
onChartDrop(event: CdkDragDrop<string[]>): void { moveItemInArray(this.chartOrder,
event.previousIndex, event.currentIndex); this.cdr.markForCheck(); }



/**



  • Handle chart pop-out request

   */
onChartPopOut(chartId: string): void { this.chartPopOut.emit(chartId); }



/**



  • Handle chart click

   *

  • Gets URL params from the data source and updates URL state.

   */
```

```
onChartClick(event: { value: string; isHighlightMode: boolean }, chartId: string):
void { const dataSource = this.domainConfig.chartDataSources?.[chartId]; if (!
dataSource) return;



// Get URL params from data source const newParams =
dataSource.toUrlParams(event.value, event.isHighlightMode);



// Update URL (either directly or via pop-out message) if
(this.popOutContext.isInPopOut()) { this.popOutContext.sendMessage({ type:
PopOutMessageType.URL_PARAMS_CHANGED, payload: { params: newParams }, timestamp:
Date.now() }); } else { this.urlState.setParams(newParams); } }



/**



  • Get data source for a chart ID


   */
getDataSource(chartId: string): ChartDataSource | undefined { return
this.domainConfig.chartDataSources?.[chartId]; } }
```

Step 804.3: Create the Statistics Panel Template

Create the file `src/app/framework/components/statistics-panel/statistics-panel.component.html`:

```
<!-- src/app/framework/components/statistics-panel/statistics-panel.component.html -->

<!-- VERSION 1 (Section 804) - Chart grid with drag-drop -->



<div class="statistics-content"> <!-- No data message --> <div *ngIf="!(statistics$ |
async)" class="no-data-message"> <i class="pi pi-info-circle"></i> <p>No statistics
available. Add filters or select data to view distributions.</p> </div>



<!-- Chart Grid with CDK Drag-Drop --> <div *ngIf="statistics$ | async" cdkDropList
cdkDropListOrientation="horizontal" (cdkDropListDropped)="onChartDrop($event)"
class="chart-grid">



<ng-container *ngFor="let chartId of chartOrder"> <div *ngIf="getDataSource(chartId)
as dataSource" class="chart-box" cdkDrag> <!-- Drag Handle --> <div class="chart-drag-
handle" cdkDragHandle> <i class="pi pi-bars"></i> </div>



<!-- Chart or Placeholder --> <ng-container *ngIf="isInPopOut || !
isPanelPoppedOut('chart-' + chartId); else chartPoppedOut"> <app-base-chart
[dataSource]="dataSource" [statistics]="statistics$ | async" [highlights]="highlights$
| async" [selectedValue]="null" [canPopOut]="!
isInPopOut" (popOutClick)="onChartPopOut(chartId)" (chartClick)="onChartClick($event,
chartId)"> </app-base-chart> </ng-container>



<ng-template #chartPoppedOut> <div class="popout-placeholder"> <i class="pi pi-
external-link"></i> <span>Chart is open in a separate window</span> </div> </ng-
template> </div> </ng-container> </div> </div>
```

Step 804.4: Create the Statistics Panel Styles

Create the file `src/app/framework/components/statistics-panel/statistics-panel.component.scss`:

```
// src/app/framework/components/statistics-panel/statistics-panel.component.scss
// VERSION 1 (Section 804) - Chart grid styles

.statistics-content { padding: 0.5rem; }

.no-data-message { display: flex; flex-direction: column; align-items: center; justify-content: center; padding: 3rem 1rem; text-align: center; color: var(--text-color-secondary);

i.pi { font-size: 3rem; margin-bottom: 1rem; opacity: 0.5; }

p { margin: 0; font-size: 1rem; } }

.chart-grid { display: flex; flex-wrap: wrap; gap: 1rem; padding: 0.5rem; background: var(--surface-ground); border-radius: 6px; border: 1px solid var(--surface-border); min-height: 200px;

.chart-box { // Two charts per row flex: 0 0 calc(50% - 0.5rem); background: var(--surface-card); border-radius: 4px; padding: 1rem; position: relative; box-sizing: border-box;

// Drag handle .chart-drag-handle { position: absolute; top: 0.5rem; left: 0.5rem; cursor: move; padding: 0.25rem 0.5rem; color: var(--text-color-secondary); opacity: 0.6; transition: opacity 0.2s, color 0.2s; z-index: 10;

&:hover { opacity: 1; color: var(--text-color); }

i { font-size: 1rem; } }
```

```
app-base-chart { display: block; height: 300px; }



.popout-placeholder { display: flex; align-items: center; justify-content: center;
gap: 0.75rem; height: 300px; background-color: var(--surface-ground); border: 2px
dashed var(--surface-border); border-radius: 4px; color: var(--text-color-secondary);
font-style: italic;



i { font-size: 1.5rem; color: var(--primary-color); } }



// CDK Drag states &.cdk-drag-preview { box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
opacity: 0.95; border: 2px solid var(--primary-color); }



&.cdk-drag-placeholder { opacity: 0.3; background: var(--surface-hover); border: 2px
dashed var(--surface-border); } } }



// Responsive adjustments @media (max-width: 1200px) { .chart-grid { .chart-box
{ flex: 0 0 100%; } } }
```

Step 804.5: Create the Module Export

Create the file `src/app/framework/components/statistics-panel/index.ts`:

```
// src/app/framework/components/statistics-panel/index.ts

// VERSION 1 (Section 804) - Barrel export



export { StatisticsPanelComponent } from './statistics-panel.component';
```

Step 804.6: Register in Framework Module

Open `src/app/framework/framework.module.ts` and add the component:

```
// src/app/framework/framework.module.ts

// VERSION 5 (Section 804) - Add StatisticsPanelComponent // Replaces VERSION 4 from
Section 803



import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/
common'; import { FormsModule } from '@angular/forms'; import { DragDropModule } from
'@angular/cdk/drag-drop'; import { ButtonModule } from 'primeng/button'; import
{ TableModule } from 'primeng/table'; import { CheckboxModule } from 'primeng/
checkbox'; import { InputTextModule } from 'primeng/inputtext'; import
{ SkeletonModule } from 'primeng/skeleton'; import { MessageModule } from 'primeng/
message'; import { RippleModule } from 'primeng/ripple';



import { BaseChartComponent } from './components/base-chart/base-chart.component';
import { BasePickerComponent } from './components/base-picker/base-picker.component';
import { BasicResultsTableComponent } from './components/basic-results-table/basic-
results-table.component'; import { StatisticsPanelComponent } from './components/
statistics-panel/statistics-panel.component';



@NgModule({ declarations: [ BaseChartComponent, BasePickerComponent,
BasicResultsTableComponent, StatisticsPanelComponent ], imports: [ CommonModule,
FormsModule, DragDropModule, ButtonModule, TableModule, CheckboxModule,
InputTextModule, SkeletonModule, MessageModule, RippleModule ], exports:
[ BaseChartComponent, BasePickerComponent, BasicResultsTableComponent,
StatisticsPanelComponent ] }) export class FrameworkModule {}
```

Verification

1. Build the Application

```
$ cd ~/projects/vvroom
$ ng build
```

Expected: Build succeeds with no errors.

2. Check CDK Installation

```
$ npm list @angular/cdk
```

Expected: Shows CDK version installed.

3. Verify Drag-Drop Import

```
$ grep "DragDropModule" src/app/framework/framework.module.ts
```

Expected: `DragDropModule` is in imports array.

4. Manual Inspection

Check `src/app/framework/components/statistics-panel/statistics-panel.component.html`:

- `cdkDropList` is on the grid container
- `cdkDrag` is on each chart box
- `cdkDragHandle` is on the drag handle element

Common Problems

Symptom	Cause	Solution
`No provider for DragDropModule`	Module not imported	Add `DragDropModule` to imports in `FrameworkModule`
Charts don't reorder	`cdkDrag` not on correct element	Ensure `cdkDrag` is on `.chart-box` div
Drag preview looks wrong	Missing CDK styles	Ensure `.cdk-drag-preview` and `.cdk-drag-placeholder` styles are defined
Charts overlap during drag	Missing `position: relative`	Add `position: relative` to `.chart-box`
Click events fire during drag	CDK captures events	This is expected; CDK handles the separation

---

Key Takeaways

- **CDK provides powerful primitives** - Drag-drop without a full UI library
- **Composition over inheritance** - StatisticsPanel composes BaseChartComponents
- **URL updates flow through data sources** - The panel doesn't know how to update URLs; it delegates to data sources

---

Acceptance Criteria

- [ ] `@angular/cdk` is installed and listed in `package.json`
- [ ] `StatisticsPanelComponent` accepts `domainConfig` and optional `chartIds` inputs
- [ ] Charts render in a grid layout (2 per row on desktop, 1 per row on mobile)
- [ ] Charts can be reordered via drag-and-drop
- [ ] Drag handle appears on hover
- [ ] Chart clicks update URL state via data source's `toUrlParams()`
- [ ] Pop-out placeholder shows when a chart is in a separate window
- [ ] "No data" message shows when `statistics$` is null
- [ ] Component is registered in `FrameworkModule`
- [ ] `ng build` completes with no errors

Next Step

Proceed to `805-inline-filters-component.md` to build the compact filter chip display component.

805: Inline Filters Component

Status: Planning **Depends On:** 203-filter-definition-interface, 301-url-state-service, 306-resource-management-service **Blocks:** 904-automobile-discover

---

Learning Objectives

After completing this section, you will:

- Understand how to display active filters as interactive chips
- Know how to synchronize chip state with URL parameters
- Be able to implement click-to-edit and click-to-remove functionality

---

Objective

Build a compact inline filters component that displays active filters as chips. Each chip shows the filter label and value(s), can be clicked to edit, and has a remove button. This component provides a non-intrusive way to show what filters are currently active.

Why

Users need to see what filters are active without opening a panel. The **Inline Filters Component** provides:

- **Visibility**: Users immediately see which filters are applied
- **Quick removal**: One-click removal of any filter
- **Edit access**: Click a chip to open the filter editor
- **Compact display**: Uses minimal vertical space

This component complements the Query Panel (806) by showing the "output" of filter selections in a condensed format.

Angular Style Guide References

- Style 03-01: Use single responsibility principle
- Style 05-02: Use input properties for data binding

URL-First Architecture Reference

The inline filters component reads active filters from URL state (via `UrlStateService`) and displays them as chips. When a chip is removed, the component emits an event that clears the corresponding URL parameter. The component never stores filter state internally - it only reflects URL state.

What

Step 805.1: Create the Inline Filters Component TypeScript

Create the file `src/app/framework/components/inline-filters/inline-filters.component.ts`:

```
// src/app/framework/components/inline-filters/inline-filters.component.ts

// VERSION 1 (Section 805) - Active filter chip display



import { ChangeDetectionStrategy, ChangeDetectorRef, Component, EventEmitter, Input,
OnDestroy, OnInit, Output } from '@angular/core'; import { CommonModule } from
'@angular/common'; import { Subject } from 'rxjs'; import { takeUntil } from 'rxjs/
operators';



import { ChipModule } from 'primeng/chip'; import { TooltipModule } from 'primeng/
tooltip'; import { ButtonModule } from 'primeng/button';



import { FilterDefinition } from '../../models/filter-definition.interface'; import
{ UrlStateService } from '../../services/url-state.service';



/**



  • Active filter representation for display


 */

export interface ActiveFilterChip { /**



  • Filter definition from domain config


   */

definition: FilterDefinition;



/**



  • Display values (may be truncated)
```

```
   */
values: (string | number)[];


/**


  • URL parameter value for this filter

   */
urlValue: string; }


/**


  • Inline Filters Component

 *

  • Displays active filters as removable chips.


  • Reads filter state from URL and emits events for removal/editing.

 *

  • @example


  •
```

html

- <app-inline-filters
- [filterDefinitions]="domainConfig.queryControlFilters"
- (filterRemove)="onFilterRemove($event)"
- (filterEdit)="onFilterEdit($event)">
- </app-inline-filters>

```
 /
@Component({ selector: 'app-inline-filters', templateUrl: './inline-
filters.component.html', styleUrls: ['./inline-filters.component.scss'],
changeDetection: ChangeDetectionStrategy.OnPush }) export class InlineFiltersComponent
implements OnInit, OnDestroy {



private readonly destroy$ = new Subject<void>();



/**



  • Filter definitions from domain config


   */
@Input() filterDefinitions: FilterDefinition[] = [];



/**



  • Highlight filter definitions (optional)


   */
@Input() highlightDefinitions: FilterDefinition[] = [];



/**



  • Whether to show a "Clear All" button


   */
@Input() showClearAll = true;
```

```
/**


  • Emits when a filter chip is removed

   */
@Output() filterRemove = new EventEmitter<ActiveFilterChip>();


/**


  • Emits when a filter chip is clicked for editing

   */
@Output() filterEdit = new EventEmitter<ActiveFilterChip>();


/**


  • Emits when "Clear All" is clicked

   */
@Output() clearAll = new EventEmitter<void>();


/**


  • Active filters derived from URL state

   */
activeFilters: ActiveFilterChip[] = [];
```

```
/**


  • Active highlights derived from URL state


   */
activeHighlights: ActiveFilterChip[] = [];



constructor( private readonly urlState: UrlStateService, private readonly cdr:
ChangeDetectorRef ) {}



// ============================================================================ //
Lifecycle //
==============================================================================



ngOnInit(): void { // Subscribe to URL changes to rebuild active filter chips
this.urlState.params$ .pipe(takeUntil(this.destroy$)) .subscribe(params =>
{ this.buildActiveFilters(params); this.buildActiveHighlights(params);
this.cdr.markForCheck(); }); }



ngOnDestroy(): void { this.destroy$.next(); this.destroy$.complete(); }



// ============================================================================ //
Active Filter Management //
==============================================================================



/**


  • Build active filter chips from URL params


   */
```

```
private buildActiveFilters(params: Record<string, any>): void { this.activeFilters =
[];



for (const filterDef of this.filterDefinitions) { const chip =
this.buildChipFromParams(params, filterDef); if (chip)
{ this.activeFilters.push(chip); } } }



/**



  • Build active highlight chips from URL params


   */
private buildActiveHighlights(params: Record<string, any>): void
{ this.activeHighlights = [];



for (const filterDef of this.highlightDefinitions) { const chip =
this.buildChipFromParams(params, filterDef); if (chip)
{ this.activeHighlights.push(chip); } } }



/**



  • Build a chip from URL params for a given filter definition


   */
private buildChipFromParams( params: Record<string, any>, filterDef:
FilterDefinition ): ActiveFilterChip | null { if (filterDef.type === 'range') { //
Range filters have min/max params const urlParamsConfig = filterDef.urlParams as
{ min: string; max: string }; const minValue = params[urlParamsConfig.min]; const
maxValue = params[urlParamsConfig.max];



if (minValue || maxValue) { const values: (string | number)[] = []; if (minValue)
values.push(minValue); if (maxValue) values.push(maxValue);
```

```
return { definition: filterDef, values, urlValue: ${minValue || ''}-${maxValue || ''} }; } } else { // Other filters have a single param const paramName = filterDef.urlParams as string; const paramValue = params[paramName];



if (paramValue) { const values = Array.isArray(paramValue) ? paramValue : String(paramValue).split(',');



return { definition: filterDef, values, urlValue: Array.isArray(paramValue) ? paramValue.join(',') : paramValue }; } }



return null; }



// ========================================================================== // Event Handlers // ==========================================================================



/**



  • Handle chip remove click



   */

onChipRemove(chip: ActiveFilterChip, event: Event): void { event.stopPropagation(); // Prevent chip click (edit) from firing this.filterRemove.emit(chip); }



/**



  • Handle chip click (edit)



   */
```

```
onChipClick(chip: ActiveFilterChip): void { this.filterEdit.emit(chip); }



/**



  • Handle clear all click


   */
onClearAll(): void { this.clearAll.emit(); }



// ========================================================================== //
Display Helpers //
============================================================================



/**



  • Get display label for a chip


   */
getChipLabel(chip: ActiveFilterChip): string { if (chip.definition.type === 'range')
{ const values = chip.values; if (values.length === 2) { return ${chip.definition.label}: ${values[0]} - ${values[1]}; } else if (values.length
=== 1) { return ${chip.definition.label}: ${values[0]}+; } return
chip.definition.label; }



// For multiselect, truncate if too many values const displayValues =
chip.values.slice(0, 3).join(', '); const remaining = chip.values.length - 3; return
remaining > 0 ? ${chip.definition.label}: ${displayValues}... +${remaining} :
${chip.definition.label}: ${displayValues}; }



/**
```

```
   • Get tooltip for a chip

    */
getChipTooltip(chip: ActiveFilterChip): string { return ${chip.definition.label}: ${chip.values.join(', ')} (Click to edit); }



/**

   • Check if there are any active filters or highlights

    */
get hasActiveFilters(): boolean { return this.activeFilters.length > 0 ||
this.activeHighlights.length > 0; } }
```

Step 805.2: Create the Inline Filters Template

Create the file `src/app/framework/components/inline-filters/inline-filters.component.html`:

```
<!-- src/app/framework/components/inline-filters/inline-filters.component.html -->

<!-- VERSION 1 (Section 805) - Active filter chips -->



<div class="inline-filters-container" *ngIf="hasActiveFilters"> <!-- Filter Chips -->
<div class="filter-chips" *ngIf="activeFilters.length > 0"> <span class="chips-
label">Filters:</span> <p-chip *ngFor="let chip of
activeFilters" [label]="getChipLabel(chip)" [pTooltip]="getChipTooltip(chip)"
tooltipPosition="top" [removable]="true" (onRemove)="onChipRemove(chip,
$event)" (click)="onChipClick(chip)" styleClass="filter-chip clickable"> </p-chip> </
div>



<!-- Highlight Chips --> <div class="highlight-chips" *ngIf="activeHighlights.length >
0"> <span class="chips-label">Highlights:</span> <p-chip *ngFor="let chip of
activeHighlights" [label]="getChipLabel(chip)" [pTooltip]="getChipTooltip(chip)"
tooltipPosition="top" [removable]="true" (onRemove)="onChipRemove(chip,
$event)" (click)="onChipClick(chip)" styleClass="highlight-chip clickable"> </p-chip>
</div>



<!-- Clear All Button --> <button *ngIf="showClearAll && hasActiveFilters" pButton
type="button" label="Clear All" icon="pi pi-times" class="p-button-text p-button-sm
clear-all-button" (click)="onClearAll()"> </button> </div>
```

Step 805.3: Create the Inline Filters Styles

Create the file `src/app/framework/components/inline-filters/inline-filters.component.scss`:

```
// src/app/framework/components/inline-filters/inline-filters.component.scss

// VERSION 1 (Section 805) - Filter chip styles



.inline-filters-container { display: flex; flex-wrap: wrap; align-items: center; gap:
0.5rem; padding: 0.5rem; background: var(--surface-ground); border-radius: 6px;
border: 1px solid var(--surface-border); }



.filter-chips, .highlight-chips { display: flex; flex-wrap: wrap; align-items: center;
gap: 0.5rem; }



.chips-label { font-size: 0.875rem; font-weight: 600; color: var(--text-color-
secondary); margin-right: 0.25rem; }



// Filter chip styling ::ng-deep { .filter-chip { background: var(--primary-color);
color: var(--primary-color-text, #ffffff); border-radius: 16px; font-size: 0.875rem;



&.clickable { cursor: pointer; transition: background-color 0.2s, transform 0.1s;



&:hover { background: var(--primary-600, #4338ca); transform: translateY(-1px); } }



.p-chip-remove-icon { color: var(--primary-color-text, #ffffff); margin-left: 0.5rem;



&:hover { color: var(--red-400, #f87171); } } }



.highlight-chip { background: var(--yellow-500, #eab308); color: var(--gray-900,
#111827); border-radius: 16px; font-size: 0.875rem;



&.clickable { cursor: pointer; transition: background-color 0.2s, transform 0.1s;
```

```
&:hover { background: var(--yellow-600, #ca8a04); transform: translateY(-1px); } }


.p-chip-remove-icon { color: var(--gray-900, #111827); margin-left: 0.5rem;


&:hover { color: var(--red-600, #dc2626); } } } }


.clear-all-button { margin-left: auto; color: var(--text-color-secondary);


&:hover { color: var(--red-500, #ef4444); background: var(--red-50, #fef2f2); } }


// Responsive @media (max-width: 768px) { .inline-filters-container { flex-direction:
column; align-items: flex-start; }


.clear-all-button { margin-left: 0; margin-top: 0.5rem; } }
```

Step 805.4: Create the Module Export

Create the file `src/app/framework/components/inline-filters/index.ts`:

```
// src/app/framework/components/inline-filters/index.ts
// VERSION 1 (Section 805) - Barrel export


export { InlineFiltersComponent, ActiveFilterChip } from './inline-filters.component';
```

Step 805.5: Register in Framework Module

Open `src/app/framework/framework.module.ts` and add the component:

```
// src/app/framework/framework.module.ts

// VERSION 6 (Section 805) - Add InlineFiltersComponent // Replaces VERSION 5 from
Section 804



import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/
common'; import { FormsModule } from '@angular/forms'; import { DragDropModule } from
'@angular/cdk/drag-drop'; import { ButtonModule } from 'primeng/button'; import
{ TableModule } from 'primeng/table'; import { CheckboxModule } from 'primeng/
checkbox'; import { InputTextModule } from 'primeng/inputtext'; import
{ SkeletonModule } from 'primeng/skeleton'; import { MessageModule } from 'primeng/
message'; import { RippleModule } from 'primeng/ripple'; import { ChipModule } from
'primeng/chip'; import { TooltipModule } from 'primeng/tooltip';



import { BaseChartComponent } from './components/base-chart/base-chart.component';
import { BasePickerComponent } from './components/base-picker/base-picker.component';
import { BasicResultsTableComponent } from './components/basic-results-table/basic-
results-table.component'; import { StatisticsPanelComponent } from './components/
statistics-panel/statistics-panel.component'; import { InlineFiltersComponent } from
'./components/inline-filters/inline-filters.component';



@NgModule({ declarations: [ BaseChartComponent, BasePickerComponent,
BasicResultsTableComponent, StatisticsPanelComponent, InlineFiltersComponent ],
imports: [ CommonModule, FormsModule, DragDropModule, ButtonModule, TableModule,
CheckboxModule, InputTextModule, SkeletonModule, MessageModule, RippleModule,
ChipModule, TooltipModule ], exports: [ BaseChartComponent, BasePickerComponent,
BasicResultsTableComponent, StatisticsPanelComponent, InlineFiltersComponent ] })
export class FrameworkModule {}
```

Verification

1. Build the Application

```
$ cd ~/projects/vvroom
$ ng build
```

Expected: Build succeeds with no errors.

2. Check TypeScript Compilation

```
$ npx tsc --noEmit
```

Expected: No type errors.

3. Verify PrimeNG Chip Import

```
$ grep "ChipModule" src/app/framework/framework.module.ts
```

Expected: `ChipModule` is in imports array.

4. Manual Inspection

Check `src/app/framework/components/inline-filters/inline-filters.component.ts`:

- Subscribes to `urlState.params$`
- Builds chips from URL parameters
- Emits events for remove/edit/clear all

Common Problems

Symptom	Cause	Solution
Chips not showing	URL params not matching filter definitions	Verify `urlParams` in filter definitions match actual URL param names
Remove button doesn't work	Event propagation	Verify `event.stopPropagation()` is called in `onChipRemove`
Chip click and remove both fire	Missing stop propagation	Add `event.stopPropagation()` to remove handler
Highlights not distinguishable	Same styling as filters	Verify `.highlight-chip` has different background color
Chips wrap incorrectly	Missing `flex-wrap`	Ensure container has `flex-wrap: wrap`

Key Takeaways

- **Chips provide compact filter visibility** - Users see active filters at a glance
- **Events enable parent coordination** - The component doesn't modify URL directly; it emits events
- **Separate filter and highlight sections** - Different visual treatment for different purposes

---

Acceptance Criteria

- [ ] `InlineFiltersComponent` accepts filter and highlight definitions
- [ ] Active filters from URL display as chips
- [ ] Filter chips have primary color styling
- [ ] Highlight chips have yellow/warning color styling
- [ ] Clicking a chip emits `filterEdit` event
- [ ] Clicking remove icon emits `filterRemove` event
- [ ] "Clear All" button emits `clearAll` event
- [ ] Chips show truncated values with "+N more" for long lists
- [ ] Tooltips show full filter details
- [ ] Component hides when no active filters
- [ ] Component is registered in `FrameworkModule`

- [ ] `ng build` completes with no errors

Next Step

Proceed to `806-query-panel-component.md` to build the full filter management panel.

806: Query Panel Component

Status: Planning **Depends On:** 203-filter-definition-interface, 301-url-state-service, 306-resource-management-service, 802-base-picker-component **Blocks:** 904-automobile-discover

---

Learning Objectives

After completing this section, you will:

- Understand how to render form controls dynamically from configuration
- Know how to implement debounced input handling for performance
- Be able to coordinate multiple filter types in a single panel

---

Objective

Build a configuration-driven filter panel that renders filter controls dynamically based on `DomainConfig.filters`. This component supports multiple filter types (text, number, select, multiselect, autocomplete, range, boolean, date) and integrates with the ResourceManagementService for state updates.

Why

Every data discovery application needs a way for users to filter results. The challenge is that different domains have different filter requirements: automobiles need year ranges and manufacturer dropdowns; products need category hierarchies and price ranges.

The **Query Panel Component** solves this by:

- Reading filter definitions from `DomainConfig.filters`
- Rendering appropriate PrimeNG controls for each filter type

- Handling value changes with debouncing for performance
- Syncing all filter state with URL parameters

This is the Phase 8 pattern: **Generic components + specific configuration = infinite reusability**.

Angular Style Guide References

- Style 03-01: Use single responsibility principle
- Style 05-04: Put logic in the component class

URL-First Architecture Reference

When a user changes a filter value:

- The component updates its local model immediately (for UI responsiveness)
- After debounce delay (for text inputs), it calls `ResourceManagementService.updateFilters()`
- ResourceManagementService updates URL state
- URL change triggers new API request

This flow ensures filters are shareable via URL and survive page refreshes.

What

Step 806.1: Create the Query Panel Component TypeScript

Create the file `src/app/framework/components/query-panel/query-panel.component.ts`:

```
// src/app/framework/components/query-panel/query-panel.component.ts
// VERSION 1 (Section 806) - Configuration-driven filter panel

import { ChangeDetectionStrategy, ChangeDetectorRef, Component, EventEmitter, Input,
OnDestroy, OnInit, Output } from '@angular/core'; import { CommonModule } from
'@angular/common'; import { FormsModule } from '@angular/forms'; import { HttpClient }
from '@angular/common/http'; import { Observable, Subject } from 'rxjs'; import
{ takeUntil, filter, debounceTime } from 'rxjs/operators';

import { InputTextModule } from 'primeng/inputtext'; import { InputNumberModule } from
'primeng/inputnumber'; import { ButtonModule } from 'primeng/button'; import
{ DropdownModule } from 'primeng/dropdown'; import { MultiSelectModule } from
'primeng/multiselect'; import { AutoCompleteModule } from 'primeng/autocomplete';
import { CheckboxModule } from 'primeng/checkbox'; import { CalendarModule } from
'primeng/calendar';

import { DomainConfig, FilterDefinition, FilterOption } from '../../models/domain-
config.interface'; import { ResourceManagementService } from '../../services/resource-
management.service'; import { PopOutContextService } from '../../services/popout-
context.service'; import { PopOutMessageType } from '../../models/popout.interface';

/**

  • Query Panel Component

 *

  • Domain-agnostic filter panel that renders controls based on configuration.

 *

  • Supported filter types:

  • - text: Text input with clear button
```

```
  • - number: Numeric input with spinner

  • - range: Dual inputs for min/max values

  • - select: Single-select dropdown

  • - multiselect: Multi-select dropdown

  • - autocomplete: Text input with suggestions from API

  • - date: Date picker

  • - boolean: Checkbox
*
  • @template TFilters - Domain-specific filter model type

  • @template TData - Domain-specific data model type

  • @template TStatistics - Domain-specific statistics model type
*
  • @example
```

```
•
```

html

- <app-query-panel
- [domainConfig]="automobileDomainConfig">
- </app-query-panel>

```
 /
@Component({ selector: 'app-query-panel', templateUrl: './query-panel.component.html',
styleUrls: ['./query-panel.component.scss'], changeDetection:
ChangeDetectionStrategy.OnPush }) export class QueryPanelComponent<TFilters = any,
TData = any, TStatistics = any> implements OnInit, OnDestroy {



/**



  • Domain configuration with filter definitions


   */
@Input() domainConfig!: DomainConfig<TFilters, TData, TStatistics>;



/**



  • Emits when URL parameters should be updated (for pop-out sync)


   */
@Output() urlParamsChange = new EventEmitter<{ [key: string]: any }>();



/**



  • Emits when all filters should be cleared


   */
@Output() clearAllFilters = new EventEmitter<void>();



/**
```

```
  • Observable of current filter state

   */
filters$!: Observable<TFilters>;


/**

  • Local filter values (for form binding)

   */
currentFilters: Record<string, any> = {};


/**

  • Dynamic options loaded from API

   */
dynamicOptions: Record<string, FilterOption[]> = {};


/**

  • Autocomplete suggestions by filter ID

   */
autocompleteSuggestions: Record<string, string[]> = {};


private destroy$ = new Subject<void>(); private searchSubject = new Subject<{ field:
string; value: any }>();
```

```
constructor( private resourceService: ResourceManagementService<TFilters, TData,
TStatistics>, private cdr: ChangeDetectorRef, private http: HttpClient, private
popOutContext: PopOutContextService ) { // Setup debounced search for text inputs
this.searchSubject.pipe( debounceTime(300),
takeUntil(this.destroy$) ).subscribe(({ field, value }) =>
{ this.applyFilterChange(field, value); }); }



// ============================================================================ //
Lifecycle //
==============================================================================



ngOnInit(): void { if (!this.domainConfig) { throw new Error('QueryPanelComponent
requires domainConfig input'); }



// Load dynamic options for filters with endpoints this.loadDynamicFilterOptions();



// Subscribe to filter state this.filters$ = this.resourceService.filters$;



this.filters$ .pipe(takeUntil(this.destroy$)) .subscribe(filters =>
{ this.currentFilters = { ...filters as any }; this.cdr.markForCheck(); });



// Pop-out window support if (this.popOutContext.isInPopOut())
{ this.popOutContext .getMessages$() .pipe( filter(msg => msg.type ===
PopOutMessageType.STATE_UPDATE), takeUntil(this.destroy$) ) .subscribe(message => { if
(message.payload && message.payload.state)
{ this.resourceService.syncStateFromExternal(message.payload.state);
this.cdr.markForCheck(); } }); } }



ngOnDestroy(): void { this.destroy$.next(); this.destroy$.complete(); }
```

```
// ============================================================================ //
Filter Change Handling //
============================================================================



/**



  • Handle filter input change

   *

  • @param field - Filter field ID



  • @param value - New value



  • @param debounce - Whether to debounce this change

   */
onFilterChange(field: string, value: any, debounce = false): void { // Update local
model immediately this.currentFilters[field] = value;



if (debounce) { this.searchSubject.next({ field, value }); } else
{ this.applyFilterChange(field, value); } }



/**



  • Apply filter change to state management

   */
private applyFilterChange(field: string, value: any): void { const isEmpty = value ===
null || value === undefined || value === '' || (Array.isArray(value) && value.length
=== 0);
```

```
const paramValue = isEmpty ? null : value;



if (this.popOutContext.isInPopOut()) { // In pop-out: emit event for parent
this.urlParamsChange.emit({ [field]: paramValue, page: 1 }); } else { // In main
window: update ResourceManagementService const newFilters: Record<string, any> =
{ ...this.currentFilters, page: 1 };



if (isEmpty) { newFilters[field] = undefined; } else { newFilters[field] = value; }



this.resourceService.updateFilters(newFilters as unknown as TFilters); } }



/**



  • Clear all filters

   */
clearFilters(): void { if (this.popOutContext.isInPopOut())
{ this.clearAllFilters.emit(); } else { this.resourceService.updateFilters({ page: 1,
size: this.currentFilters['size'] || 20 } as unknown as TFilters); } }



// ========================================================================== //
Options Management //
============================================================================



/**



  • Get options for a filter

   */
```

```
getFilterOptions(filterId: string): FilterOption[] { if
(this.dynamicOptions[filterId]) { return this.dynamicOptions[filterId]; } const
filterDef = this.domainConfig.filters.find(f => f.id === filterId); return
filterDef?.options || []; }



/**



  • Load dynamic options from API


   */
private loadDynamicFilterOptions(): void { const filtersWithEndpoint =
this.domainConfig.filters.filter(f => f.optionsEndpoint);



filtersWithEndpoint.forEach(filterDef => { const endpoint = `$
{this.domainConfig.apiBaseUrl}/agg/${filterDef.optionsEndpoint}`;



this.http.get<{ field: string; values: Array<{ value: string; count: number }> }
>(endpoint) .pipe(takeUntil(this.destroy$)) .subscribe({ next: (response) =>
{ this.dynamicOptions[filterDef.id] = response.values.map(item => ({ value:
item.value, label: item.value })); this.cdr.markForCheck(); }, error: (err) =>
{ console.error(`Failed to load options for ${filterDef.id}:`, err); } }); }); }



// ============================================================================ //
Autocomplete //
==============================================================================



/**



  • Handle autocomplete search


   */
```

```
onAutocompleteSearch(event: { query: string }, filterDef: FilterDefinition): void { if
(!filterDef.autocompleteEndpoint) { return; }



const query = event.query; const limit = 10; const endpoint = $
{this.domainConfig.apiBaseUrl}/${filterDef.autocompleteEndpoint}?search=$
{encodeURIComponent(query)}&limit=${limit};



this.http.get<Record<string,
string[]>>(endpoint) .pipe(takeUntil(this.destroy$)) .subscribe({ next: (response) =>
{ const values = Object.values(response)[0] || [];
this.autocompleteSuggestions[filterDef.id] = values; this.cdr.detectChanges(); },
error: () => { this.autocompleteSuggestions[filterDef.id] = [];
this.cdr.detectChanges(); } }); }



/**



  • Handle autocomplete focus - load initial suggestions


   */
onAutocompleteFocus(filterDef: FilterDefinition): void { if
(this.autocompleteSuggestions[filterDef.id]?.length > 0) { return; }
this.onAutocompleteSearch({ query: '' }, filterDef); } }
```

Step 806.2: Create the Query Panel Template

Create the file `src/app/framework/components/query-panel/query-panel.component.html`:

```
<!-- src/app/framework/components/query-panel/query-panel.component.html -->

<!-- VERSION 1 (Section 806) - Dynamic filter controls -->



<div class="query-panel-container"> <div class="filter-panel-container"> <div
class="filter-grid"> <!-- Dynamic Filter Rendering --> <ng-container *ngFor="let
filterDef of domainConfig.filters"> <!-- Skip search filter (handled separately) -->
<ng-container *ngIf="filterDef.id !== 'search'"> <div class="filter-field"> <label
[for]="filterDef.id">{{ filterDef.label }}</label>



<!-- Text Input --> <ng-container *ngIf="filterDef.type === 'text'"> <span class="p-
input-icon-right" style="display: block;"> <i *ngIf="currentFilters[filterDef.id]"
class="pi pi-times" (click)="onFilterChange(filterDef.id, null)" style="cursor:
pointer;"></i> <input pInputText
[id]="filterDef.id" [(ngModel)]="currentFilters[filterDef.id]" (ngModelChange)="onFilt
erChange(filterDef.id, $event, true)" [placeholder]="filterDef.placeholder || ''"
style="width: 100%;"> </span> </ng-container>



<!-- Number Input --> <ng-container *ngIf="filterDef.type === 'number'"> <div
class="p-inputgroup"> <p-inputNumber
[id]="filterDef.id" [(ngModel)]="currentFilters[filterDef.id]" (ngModelChange)="onFilt
erChange(filterDef.id, $event)" [showButtons]="true" [min]="+(filterDef.min ??
0)" [max]="+(filterDef.max ?? 999999)" [step]="filterDef.step ??
1" [placeholder]="filterDef.placeholder || ''"> </p-inputNumber> <button
*ngIf="currentFilters[filterDef.id] != null" pButton type="button" icon="pi pi-times"
class="p-button-outlined p-button-secondary" (click)="onFilterChange(filterDef.id,
null)"> </button> </div> </ng-container>



<!-- Range Input (Min) --> <ng-container *ngIf="filterDef.type === 'range'"> <div
class="p-inputgroup"> <p-inputNumber [id]="filterDef.id +
'Min'" [(ngModel)]="currentFilters[filterDef.id +
'Min']" (ngModelChange)="onFilterChange(filterDef.id + 'Min',
$event)" [showButtons]="true" [min]="+(filterDef.min ?? 0)" [max]="+(filterDef.max ??
999999)" placeholder="Min"> </p-inputNumber> <button
*ngIf="currentFilters[filterDef.id + 'Min'] != null" pButton type="button" icon="pi
pi-times" class="p-button-outlined p-button-
secondary" (click)="onFilterChange(filterDef.id + 'Min', null)"> </button> </div> </
ng-container>
```

```
<!-- Select Dropdown --> <ng-container *ngIf="filterDef.type === 'select'"> <p-
dropdown
[id]="filterDef.id" [(ngModel)]="currentFilters[filterDef.id]" (ngModelChange)="onFilt
erChange(filterDef.id, $event)" [options]="getFilterOptions(filterDef.id)"
optionLabel="label" optionValue="value" [showClear]="true" [filter]="true"
filterBy="label" [placeholder]="filterDef.placeholder || 'Select...'"> </p-dropdown>
</ng-container>



<!-- Multi-Select --> <ng-container *ngIf="filterDef.type === 'multiselect'"> <p-
multiSelect
[id]="filterDef.id" [(ngModel)]="currentFilters[filterDef.id]" (ngModelChange)="onFilt
erChange(filterDef.id, $event)" [options]="getFilterOptions(filterDef.id)"
optionLabel="label"
optionValue="value" [filter]="true" [showClear]="true" [placeholder]="filterDef.placeh
older || 'Select...'"> </p-multiSelect> </ng-container>



<!-- Autocomplete --> <ng-container *ngIf="filterDef.type === 'autocomplete'"> <p-
autoComplete
[id]="filterDef.id" [(ngModel)]="currentFilters[filterDef.id]" (onSelect)="onFilterCha
nge(filterDef.id, $event)" (onClear)="onFilterChange(filterDef.id,
null)" [suggestions]="autocompleteSuggestions[filterDef.id] ||
[]" (completeMethod)="onAutocompleteSearch($event,
filterDef)" (onFocus)="onAutocompleteFocus(filterDef)" [minLength]="1" [forceSelection
]="false" [placeholder]="filterDef.placeholder || 'Type to
search...'" [showClear]="true" [delay]="300" appendTo="body" styleClass="w-full"> </p-
autoComplete> </ng-container>



<!-- Date Picker --> <ng-container *ngIf="filterDef.type === 'date'"> <div class="p-
inputgroup"> <p-calendar
[id]="filterDef.id" [(ngModel)]="currentFilters[filterDef.id]" (ngModelChange)="onFilt
erChange(filterDef.id, $event)" dateFormat="yy-mm-dd"> </p-calendar> <button
*ngIf="currentFilters[filterDef.id]" pButton type="button" icon="pi pi-times"
class="p-button-outlined p-button-secondary" (click)="onFilterChange(filterDef.id,
null)"> </button> </div> </ng-container>



<!-- Boolean Checkbox --> <ng-container *ngIf="filterDef.type === 'boolean'"> <p-
checkbox
[id]="filterDef.id" [(ngModel)]="currentFilters[filterDef.id]" (ngModelChange)="onFilt
erChange(filterDef.id, $event)" [binary]="true"> </p-checkbox> </ng-container> </div>
```

```
<!-- Range Max (second input) --> <div *ngIf="filterDef.type === 'range'"
class="filter-field"> <label [for]="filterDef.id + 'Max'">{{ filterDef.label }} Max</
label> <div class="p-inputgroup"> <p-inputNumber [id]="filterDef.id +
'Max'" [(ngModel)]="currentFilters[filterDef.id +
'Max']" (ngModelChange)="onFilterChange(filterDef.id + 'Max',
$event)" [showButtons]="true" [min]="+(filterDef.min ?? 0)" [max]="+(filterDef.max ??
999999)" placeholder="Max"> </p-inputNumber> <button
*ngIf="currentFilters[filterDef.id + 'Max'] != null" pButton type="button" icon="pi
pi-times" class="p-button-outlined p-button-
secondary" (click)="onFilterChange(filterDef.id + 'Max', null)"> </button> </div> </
div> </ng-container> </ng-container>



<!-- Clear Filters Button --> <div class="filter-field"> <label> </label> <button
pButton type="button" label="Clear Filters" icon="pi pi-filter-slash" class="p-button-
outlined" (click)="clearFilters()"> </button> </div> </div> </div>
```

Step 806.3: Create the Query Panel Styles

Create the file `src/app/framework/components/query-panel/query-panel.component.scss`:

```
// src/app/framework/components/query-panel/query-panel.component.scss

// VERSION 1 (Section 806) - Filter panel styles



.query-panel-container { display: flex; flex-direction: column; gap: 1rem; padding:
1rem;



.filter-panel-container { border: 1px solid var(--surface-border); border-radius: 6px;
overflow: hidden; background: var(--surface-card); }



.filter-grid { display: grid; grid-template-columns: repeat(auto-fit, minmax(250px,
1fr)); gap: 1rem; align-items: end; padding: 1rem;



.filter-field { display: flex; flex-direction: column; gap: 0.5rem;



label { font-weight: 600; font-size: 0.875rem; color: var(--text-color); }



input, p-inputNumber, p-dropdown, p-multiSelect, p-calendar, p-autoComplete, p-
checkbox { width: 100%; }



// Make PrimeNG components full width ::ng-deep { .p-dropdown, .p-multiselect, .p-
calendar, .p-autocomplete { width: 100%; }



.p-autocomplete-input { width: 100%; } } } } }
```

Step 806.4: Create the Module Export

Create the file `src/app/framework/components/query-panel/index.ts`:

```
// src/app/framework/components/query-panel/index.ts
// VERSION 1 (Section 806) - Barrel export


export { QueryPanelComponent } from './query-panel.component';
```

Step 806.5: Register in Framework Module

Open `src/app/framework/framework.module.ts` and add the component:

```
// src/app/framework/framework.module.ts

// VERSION 7 (Section 806) - Add QueryPanelComponent // Replaces VERSION 6 from
Section 805



import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/
common'; import { FormsModule } from '@angular/forms'; import { HttpClientModule }
from '@angular/common/http'; import { DragDropModule } from '@angular/cdk/drag-drop';
import { ButtonModule } from 'primeng/button'; import { TableModule } from 'primeng/
table'; import { CheckboxModule } from 'primeng/checkbox'; import { InputTextModule }
from 'primeng/inputtext'; import { InputNumberModule } from 'primeng/inputnumber';
import { SkeletonModule } from 'primeng/skeleton'; import { MessageModule } from
'primeng/message'; import { RippleModule } from 'primeng/ripple'; import
{ ChipModule } from 'primeng/chip'; import { TooltipModule } from 'primeng/tooltip';
import { DropdownModule } from 'primeng/dropdown'; import { MultiSelectModule } from
'primeng/multiselect'; import { AutoCompleteModule } from 'primeng/autocomplete';
import { CalendarModule } from 'primeng/calendar';



import { BaseChartComponent } from './components/base-chart/base-chart.component';
import { BasePickerComponent } from './components/base-picker/base-picker.component';
import { BasicResultsTableComponent } from './components/basic-results-table/basic-
results-table.component'; import { StatisticsPanelComponent } from './components/
statistics-panel/statistics-panel.component'; import { InlineFiltersComponent } from
'./components/inline-filters/inline-filters.component'; import { QueryPanelComponent }
from './components/query-panel/query-panel.component';



@NgModule({ declarations: [ BaseChartComponent, BasePickerComponent,
BasicResultsTableComponent, StatisticsPanelComponent, InlineFiltersComponent,
QueryPanelComponent ], imports: [ CommonModule, FormsModule, HttpClientModule,
DragDropModule, ButtonModule, TableModule, CheckboxModule, InputTextModule,
InputNumberModule, SkeletonModule, MessageModule, RippleModule, ChipModule,
TooltipModule, DropdownModule, MultiSelectModule, AutoCompleteModule,
CalendarModule ], exports: [ BaseChartComponent, BasePickerComponent,
BasicResultsTableComponent, StatisticsPanelComponent, InlineFiltersComponent,
QueryPanelComponent ] }) export class FrameworkModule {}
```

Verification

1. Build the Application

```
$ cd ~/projects/vvroom
$ ng build
```

Expected: Build succeeds with no errors.

2. Check TypeScript Compilation

```
$ npx tsc --noEmit
```

Expected: No type errors.

3. Verify All PrimeNG Imports

```
$ grep -E "DropdownModule|MultiSelectModule|AutoCompleteModule|CalendarModule" src/
app/framework/framework.module.ts
```

Expected: All modules are in imports array.

4. Manual Inspection

Check `src/app/framework/components/query-panel/query-panel.component.html`:

- Each filter type has its own `*ngIf` block
- All inputs use `[(ngModel)]` for two-way binding
- Text inputs use `debounce: true` in `onFilterChange`

Common Problems

Symptom	Cause	Solution
Filter controls not rendering	`filterDef.type` not matching template conditions	Verify filter type strings match exactly
Dropdown options empty	Dynamic options not loaded	Check `optionsEndpoint` in filter definition
Autocomplete not showing suggestions	API endpoint incorrect	Verify `autocompleteEndpoint` and API response format
Debounce not working	`searchSubject` not subscribed	Ensure constructor sets up debounce subscription
Clear button doesn't reset page	`page: 1` not included in update	Verify `applyFilterChange` includes page reset

---

Key Takeaways

- **Dynamic rendering from configuration** - One component handles all filter types
- **Debounce improves performance** - Text inputs wait before triggering API calls
- **Two-way binding with manual updates** - `ngModel` updates local state; `ngModelChange` triggers state management

---

Acceptance Criteria

- [ ] `QueryPanelComponent` accepts `domainConfig` input
- [ ] Text filters render with clear button
- [ ] Number filters render with spinner buttons
- [ ] Range filters render as two inputs (Min/Max)
- [ ] Select filters render as dropdowns
- [ ] Multiselect filters render as multi-select dropdowns
- [ ] Autocomplete filters load suggestions from API
- [ ] Date filters render as date pickers
- [ ] Boolean filters render as checkboxes
- [ ] Text inputs debounce for 300ms before applying

- [ ] "Clear Filters" button resets all filters
- [ ] Filter changes update URL state
- [ ] Pop-out window receives state updates
- [ ] Component is registered in `FrameworkModule`
- [ ] `ng build` completes with no errors

Next Step

Proceed to `807-column-manager-component.md` to build the table column visibility manager.

807: Column Manager Component

Status: Planning **Depends On:** 204-table-config-interface, 803-basic-results-table **Blocks:** 904-automobile-discover

---

Learning Objectives

After completing this section, you will:

- Understand how to manage table column visibility with user preferences
- Know how to persist column settings to localStorage
- Be able to implement drag-and-drop column reordering

Objective

Build a column manager component that allows users to show/hide and reorder table columns. The component displays a list of available columns with checkboxes and drag handles, persists preferences to localStorage, and emits column configuration changes to parent components.

Why

Data tables often have many columns, but users typically only need a subset for their current task. The **Column Manager Component** provides:

- **Column visibility toggle**: Show/hide columns without losing data
- **Column reordering**: Drag columns to customize display order
- **Persistence**: Settings survive browser refresh
- **Domain-agnostic**: Works with any table configuration

This enhances user experience by letting each user customize their view.

Angular Style Guide References

- Style 03-01: Use single responsibility principle
- Style 05-02: Use input properties for data binding

LocalStorage Persistence

Column preferences are stored in localStorage using a domain-specific key:

```
vvroom_columns_{domainId}
```

This ensures settings persist across sessions and don't conflict between domains.

What

Step 807.1: Create the Column Manager Interface

Create the file `src/app/framework/components/column-manager/column-manager.interface.ts`:

```
// src/app/framework/components/column-manager/column-manager.interface.ts
// VERSION 1 (Section 807) - Column manager types


import { TableColumn } from '../../models/table-config.interface';


/**


  • Column visibility state

 */
export interface ColumnState { /**


  • Column field name (unique identifier)

   */
field: string;


/**


  • Whether column is visible

   */
visible: boolean;


/**


  • Display order (lower = earlier)
```

```
   */
order: number; }


/**


  • Column manager output event

 */
export interface ColumnConfigEvent { /**


  • Visible columns in display order

   */
visibleColumns: TableColumn[];


/**


  • All column states (for persistence)

   */
allColumnStates: ColumnState[]; }
```

Step 807.2: Create the Column Manager Component TypeScript

Create the file `src/app/framework/components/column-manager/column-manager.component.ts`:

```
// src/app/framework/components/column-manager/column-manager.component.ts
// VERSION 1 (Section 807) - Table column visibility manager



import { ChangeDetectionStrategy, ChangeDetectorRef, Component, EventEmitter, Input,
OnDestroy, OnInit, Output } from '@angular/core'; import { CommonModule } from
'@angular/common'; import { FormsModule } from '@angular/forms';



import { DragDropModule, CdkDragDrop, moveItemInArray } from '@angular/cdk/drag-drop';
import { CheckboxModule } from 'primeng/checkbox'; import { ButtonModule } from
'primeng/button'; import { DialogModule } from 'primeng/dialog'; import
{ TooltipModule } from 'primeng/tooltip';



import { TableColumn } from '../../models/table-config.interface'; import
{ ColumnState, ColumnConfigEvent } from './column-manager.interface';



/**



  • Column Manager Component

 *

  • Provides UI for managing table column visibility and order.



  • Persists settings to localStorage.

 *

  • @example



  •
```

html

- <app-column-manager
- [columns]="tableConfig.columns"
- [domainId]="'automobile'"
- (columnConfigChange)="onColumnConfigChange($event)">
- </app-column-manager>

```
 /
@Component({ selector: 'app-column-manager', templateUrl: './column-
manager.component.html', styleUrls: ['./column-manager.component.scss'],
changeDetection: ChangeDetectionStrategy.OnPush }) export class ColumnManagerComponent
implements OnInit, OnDestroy {



/**



  • Available columns from table config


   */
@Input() columns: TableColumn[] = [];



/**



  • Domain ID for localStorage key


   */
@Input() domainId = 'default';



/**



  • Minimum number of visible columns required


   */
@Input() minVisibleColumns = 1;



/**
```

```
  • Emits when column configuration changes

   */
@Output() columnConfigChange = new EventEmitter<ColumnConfigEvent>();

/**

  • Whether the column manager dialog is visible

   */
dialogVisible = false;

/**

  • Column states for the manager UI

   */
columnStates: ColumnState[] = [];

private readonly STORAGE_KEY_PREFIX = 'vvroom_columns_';

constructor(private readonly cdr: ChangeDetectorRef) {}

// ============================================================================ //
Lifecycle //
============================================================================
```

```
ngOnInit(): void { this.loadColumnStates(); this.emitCurrentConfig(); }



ngOnDestroy(): void { // Save on destroy in case of unsaved changes
this.saveColumnStates(); }



// ========================================================================== //
Dialog Management //
============================================================================



/**



  • Open the column manager dialog


   */

openDialog(): void { this.dialogVisible = true; this.cdr.markForCheck(); }



/**



  • Close the dialog and save changes


   */

closeDialog(): void { this.saveColumnStates(); this.emitCurrentConfig();
this.dialogVisible = false; this.cdr.markForCheck(); }



// ========================================================================== //
Column State Management //
============================================================================



/**
```

```
  • Load column states from localStorage or initialize from input columns

   */
private loadColumnStates(): void { const storageKey = this.STORAGE_KEY_PREFIX +
this.domainId; const savedStates = localStorage.getItem(storageKey);



if (savedStates) { try { const parsed: ColumnState[] = JSON.parse(savedStates);



// Merge saved states with current columns // (handles new columns added since last
save) this.columnStates = this.mergeColumnStates(parsed); } catch { // Invalid JSON,
initialize fresh this.initializeColumnStates(); } } else
{ this.initializeColumnStates(); } }



/**



  • Initialize column states from input columns

   */
private initializeColumnStates(): void { this.columnStates = this.columns.map((col,
index) => ({ field: col.field, visible: col.hidden !== true, order: index })); }



/**



  • Merge saved states with current columns

   */
private mergeColumnStates(savedStates: ColumnState[]): ColumnState[] { const
savedByField = new Map(savedStates.map(s => [s.field, s])); const merged:
ColumnState[] = [];
```

```
// Add states for all current columns this.columns.forEach((col, index) => { const
saved = savedByField.get(col.field); if (saved) { merged.push({ ...saved });
savedByField.delete(col.field); } else { // New column not in saved states
merged.push({ field: col.field, visible: col.hidden !== true, order: index + 1000 //
Put new columns at end }); } });



// Sort by order merged.sort((a, b) => a.order - b.order);



// Normalize order values merged.forEach((state, index) => { state.order = index; });



return merged; }



/**



  • Save column states to localStorage


   */

private saveColumnStates(): void { const storageKey = this.STORAGE_KEY_PREFIX +
this.domainId; localStorage.setItem(storageKey, JSON.stringify(this.columnStates)); }



// ============================================================================ //
Event Handlers //
==============================================================================



/**



  • Handle visibility checkbox change


   */

onVisibilityChange(state: ColumnState): void { // Enforce minimum visible columns
const visibleCount = this.columnStates.filter(s => s.visible).length; if (!
```

```
state.visible && visibleCount <= this.minVisibleColumns) { // Can't hide, restore
visible state state.visible = true; this.cdr.markForCheck(); return; }



this.cdr.markForCheck(); }



/**



  • Handle drag-drop reordering


   */

onColumnDrop(event: CdkDragDrop<ColumnState[]>): void
{ moveItemInArray(this.columnStates, event.previousIndex, event.currentIndex);



// Update order values this.columnStates.forEach((state, index) => { state.order =
index; });



this.cdr.markForCheck(); }



/**



  • Show all columns


   */

showAll(): void { this.columnStates.forEach(state => { state.visible = true; });
this.cdr.markForCheck(); }



/**



  • Reset to default column configuration
```

```
   */
resetToDefault(): void { this.initializeColumnStates(); this.cdr.markForCheck(); }



// ============================================================================ //
Output // ============================================================================



/**



  • Emit current column configuration


   */
private emitCurrentConfig(): void { // Build visible columns in order const
visibleColumns: TableColumn[] = []; const columnsByField = new Map(this.columns.map(c
=> [c.field, c]));



this.columnStates .filter(state => state.visible) .forEach(state => { const column =
columnsByField.get(state.field); if (column) { visibleColumns.push(column); } });



const event: ColumnConfigEvent = { visibleColumns, allColumnStates:
[...this.columnStates] };



this.columnConfigChange.emit(event); }



// ============================================================================ //
Template Helpers //
==============================================================================



/**
```

```
  • Get column definition for a state

   */
getColumn(state: ColumnState): TableColumn | undefined { return this.columns.find(c =>
c.field === state.field); }



/**



  • Check if a column can be hidden

   */
canHide(state: ColumnState): boolean { const visibleCount = this.columnStates.filter(s
=> s.visible).length; return state.visible && visibleCount >
this.minVisibleColumns; } }
```

Step 807.3: Create the Column Manager Template

Create the file `src/app/framework/components/column-manager/column-manager.component.html`:

```
<!-- src/app/framework/components/column-manager/column-manager.component.html -->

<!-- VERSION 1 (Section 807) - Column visibility manager UI -->



<!-- Trigger Button --> <button pButton type="button" icon="pi pi-sliders-h" class="p-
button-text p-button-secondary" pTooltip="Manage Columns"
tooltipPosition="top" (click)="openDialog()"> </button>



<!-- Column Manager Dialog --> <p-dialog header="Manage
Columns" [(visible)]="dialogVisible" [modal]="true" [style]="{ width:
'400px' }" [closable]="true" [closeOnEscape]="true" [dismissableMask]="true" (onHide)=
"closeDialog()">



<!-- Dialog Content --> <div class="column-manager-content"> <p class="instructions">
Drag to reorder. Check/uncheck to show/hide columns. </p>



<!-- Column List with Drag-Drop --> <div cdkDropList
(cdkDropListDropped)="onColumnDrop($event)" class="column-list">



<div *ngFor="let state of columnStates" cdkDrag class="column-item">



<!-- Drag Handle --> <div class="drag-handle" cdkDragHandle> <i class="pi pi-bars"></
i> </div>



<!-- Visibility Checkbox --> <p-checkbox
[(ngModel)]="state.visible" [binary]="true" (onChange)="onVisibilityChange(state)" [pT
ooltip]="!canHide(state) ? 'At least one column must be visible' : ''"
tooltipPosition="right"> </p-checkbox>



<!-- Column Name --> <span class="column-name"> {{ getColumn(state)?.header ||
state.field }} </span> </div> </div> </div>
```

```
<!-- Dialog Footer --> <ng-template pTemplate="footer"> <div class="dialog-actions">
<button pButton type="button" label="Show All" icon="pi pi-eye" class="p-button-
text" (click)="showAll()"> </button> <button pButton type="button" label="Reset"
icon="pi pi-refresh" class="p-button-text" (click)="resetToDefault()"> </button>
<button pButton type="button" label="Done" icon="pi pi-check" class="p-button-
primary" (click)="closeDialog()"> </button> </div> </ng-template> </p-dialog>
```

Step 807.4: Create the Column Manager Styles

Create the file `src/app/framework/components/column-manager/column-manager.component.scss`:

```
// src/app/framework/components/column-manager/column-manager.component.scss

// VERSION 1 (Section 807) - Column manager styles



.column-manager-content { .instructions { margin: 0 0 1rem 0; font-size: 0.875rem;
color: var(--text-color-secondary); } }



.column-list { border: 1px solid var(--surface-border); border-radius: 6px; overflow:
hidden; }



.column-item { display: flex; align-items: center; gap: 0.75rem; padding: 0.75rem
1rem; background: var(--surface-card); border-bottom: 1px solid var(--surface-border);
cursor: default;



&:last-child { border-bottom: none; }



.drag-handle { cursor: move; color: var(--text-color-secondary); padding: 0.25rem;
transition: color 0.2s;



&:hover { color: var(--text-color); }



i { font-size: 1rem; } }



.column-name { flex: 1; font-size: 0.875rem; color: var(--text-color); }



// CDK Drag states &.cdk-drag-preview { box-shadow: 0 4px 12px rgba(0, 0, 0, 0.2);
background: var(--surface-card); border: 1px solid var(--primary-color); border-
radius: 4px; }
```

```
&.cdk-drag-placeholder { opacity: 0.4; background: var(--surface-hover); } }


.dialog-actions { display: flex; justify-content: flex-end; gap: 0.5rem; }
```

Step 807.5: Create the Module Export

Create the file `src/app/framework/components/column-manager/index.ts`:

```
// src/app/framework/components/column-manager/index.ts
// VERSION 1 (Section 807) - Barrel export


export { ColumnManagerComponent } from './column-manager.component'; export
{ ColumnState, ColumnConfigEvent } from './column-manager.interface';
```

Step 807.6: Register in Framework Module

Open `src/app/framework/framework.module.ts` and add the component:

```
// src/app/framework/framework.module.ts

// VERSION 8 (Section 807) - Add ColumnManagerComponent // Replaces VERSION 7 from
Section 806



import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/
common'; import { FormsModule } from '@angular/forms'; import { HttpClientModule }
from '@angular/common/http'; import { DragDropModule } from '@angular/cdk/drag-drop';
import { ButtonModule } from 'primeng/button'; import { TableModule } from 'primeng/
table'; import { CheckboxModule } from 'primeng/checkbox'; import { InputTextModule }
from 'primeng/inputtext'; import { InputNumberModule } from 'primeng/inputnumber';
import { SkeletonModule } from 'primeng/skeleton'; import { MessageModule } from
'primeng/message'; import { RippleModule } from 'primeng/ripple'; import
{ ChipModule } from 'primeng/chip'; import { TooltipModule } from 'primeng/tooltip';
import { DropdownModule } from 'primeng/dropdown'; import { MultiSelectModule } from
'primeng/multiselect'; import { AutoCompleteModule } from 'primeng/autocomplete';
import { CalendarModule } from 'primeng/calendar'; import { DialogModule } from
'primeng/dialog';



import { BaseChartComponent } from './components/base-chart/base-chart.component';
import { BasePickerComponent } from './components/base-picker/base-picker.component';
import { BasicResultsTableComponent } from './components/basic-results-table/basic-
results-table.component'; import { StatisticsPanelComponent } from './components/
statistics-panel/statistics-panel.component'; import { InlineFiltersComponent } from
'./components/inline-filters/inline-filters.component'; import { QueryPanelComponent }
from './components/query-panel/query-panel.component'; import
{ ColumnManagerComponent } from './components/column-manager/column-
manager.component';



@NgModule({ declarations: [ BaseChartComponent, BasePickerComponent,
BasicResultsTableComponent, StatisticsPanelComponent, InlineFiltersComponent,
QueryPanelComponent, ColumnManagerComponent ], imports: [ CommonModule, FormsModule,
HttpClientModule, DragDropModule, ButtonModule, TableModule, CheckboxModule,
InputTextModule, InputNumberModule, SkeletonModule, MessageModule, RippleModule,
ChipModule, TooltipModule, DropdownModule, MultiSelectModule, AutoCompleteModule,
CalendarModule, DialogModule ], exports: [ BaseChartComponent, BasePickerComponent,
BasicResultsTableComponent, StatisticsPanelComponent, InlineFiltersComponent,
QueryPanelComponent, ColumnManagerComponent ] }) export class FrameworkModule {}
```

Verification

1. Build the Application

```
$ cd ~/projects/vvroom
$ ng build
```

Expected: Build succeeds with no errors.

2. Check TypeScript Compilation

```
$ npx tsc --noEmit
```

Expected: No type errors.

3. Verify LocalStorage Key

Check `src/app/framework/components/column-manager/column-manager.component.ts`:

- `STORAGE_KEY_PREFIX` is defined
- `loadColumnStates` reads from localStorage
- `saveColumnStates` writes to localStorage

4. Verify Drag-Drop Integration

Check the template:

- `cdkDropList` on the column list
- `cdkDrag` on each column item
- `cdkDragHandle` on the drag handle

Common Problems

Symptom	Cause	Solution
Column order not persisting	localStorage not saving	Verify `saveColumnStates` is called in `closeDialog`
New columns not appearing	Merge logic not handling new columns	Check `mergeColumnStates` adds missing columns
Can't hide last column	Minimum columns check working	This is expected behavior; verify `minVisibleColumns` input
Drag preview looks wrong	Missing CDK styles	Ensure `.cdk-drag-preview` styles are defined
Dialog doesn't close	`dialogVisible` not updating	Verify `closeDialog` sets `dialogVisible = false`

Key Takeaways

- **LocalStorage provides simple persistence** - No backend required for user preferences
- **Merge handles schema evolution** - New columns appear even if user has saved preferences
- **CDK Drag-Drop is reusable** - Same pattern as StatisticsPanel (Section 804)

Acceptance Criteria

- [ ] `ColumnManagerComponent` accepts `columns` and `domainId` inputs
- [ ] Clicking the button opens a dialog
- [ ] Columns display with checkboxes and drag handles
- [ ] Checking/unchecking toggles column visibility
- [ ] Dragging reorders columns
- [ ] At least one column must remain visible
- [ ] "Show All" button shows all columns
- [ ] "Reset" button restores default order and visibility
- [ ] "Done" button closes dialog and saves settings
- [ ] Settings persist to localStorage
- [ ] Settings survive page refresh

- [ ] `columnConfigChange` emits visible columns in correct order
- [ ] Component is registered in `FrameworkModule`
- [ ] `ng build` completes with no errors

Next Step

Proceed to `808-statistics-panel-2.md` to build the refined CDK horizontal chart grid.

808: Statistics Panel 2 Component

Status: Planning **Depends On:** 801-base-chart-component, 606-chart-configs, 651-654 (Chart Data Sources) **Blocks:** 809-dockview-statistics-panel, 903-discover-page-component

---

Learning Objectives

After completing this section, you will:

- Understand how to build a CDK drag-drop chart grid with horizontal orientation
- Know how to coordinate chart clicks with URL state updates
- Be able to implement pop-out placeholder patterns for multi-window applications

---

Objective

Build a statistics panel component that renders multiple charts in a draggable grid layout using Angular CDK. This component composes `BaseChartComponent` instances and handles chart ordering, pop-out functionality, and click event coordination with URL state.

Why

The original `StatisticsPanelComponent` (document 804) introduced the concept of a chart grid. `StatisticsPanel2Component` refines this pattern with:

- **CDK horizontal orientation** — Charts drag horizontally, wrapping to rows
- **Pop-out awareness** — Shows placeholder when a chart is in a separate window
- **Pop-out window support** — Works correctly when rendered inside a pop-out window
- **URL state coordination** — Chart clicks update URL via data sources

Why a Second Statistics Panel?

The application needs different chart layouts in different contexts:

- **Discover page**: Full statistics panel with all charts
- **Pop-out windows**: Subset of charts (e.g., just manufacturer and models)

`StatisticsPanel2Component` supports both via the optional `chartIds` input.

Angular Style Guide References

- Style 03-02: Use delegation over inheritance
- Style 05-02: Use input properties for data binding

URL-First Architecture Reference

When a user clicks on a chart element:

- `BaseChartComponent` emits `chartClick` event with value and highlight mode
- `StatisticsPanel2Component` receives the event
- Data source's `toUrlParams()` converts the click to URL parameters
- URL state is updated (directly or via pop-out message)

This maintains the URL-First principle: all state flows through the URL.

What

Step 808.1: Create the Statistics Panel 2 Component TypeScript

Create the file `src/app/framework/components/statistics-panel-2/statistics-panel-2.component.ts`:

```
// src/app/framework/components/statistics-panel-2/statistics-panel-2.component.ts
// VERSION 1 (Section 808) - CDK horizontal chart grid



import { ChangeDetectionStrategy, ChangeDetectorRef, Component, EventEmitter, Inject,
Input, OnDestroy, OnInit, Optional, Output } from '@angular/core'; import
{ ActivatedRoute } from '@angular/router'; import { Observable, Subject } from 'rxjs';
import { takeUntil } from 'rxjs/operators'; import { CdkDragDrop, moveItemInArray }
from '@angular/cdk/drag-drop';



import { DomainConfig } from '../../models/domain-config.interface'; import
{ PopOutMessageType } from '../../models/popout.interface'; import
{ PopOutContextService } from '../../services/popout-context.service'; import
{ ResourceManagementService } from '../../services/resource-management.service';
import { UrlStateService } from '../../services/url-state.service'; import
{ DomainConfigRegistry } from '../../services/domain-config-registry.service'; import
{ ChartDataSource } from '../base-chart/chart-data.interface'; import
{ IS_POPOUT_TOKEN } from '../../tokens/popout.token';



/**



  • Statistics Panel 2 Component


 *


  • Renders statistical charts in a CDK horizontal drag-drop grid.



  • Charts can be reordered by dragging.


 *


  • @example



  •
```

html

- <app-statistics-panel-2
- [domainConfig]="domainConfig"
- [chartIds]="['manufacturer', 'top-models']"
- (chartPopOut)="onChartPopOut($event)"
- (chartClicked)="onChartClick($event)">
- </app-statistics-panel-2>

```
 /
@Component({ selector: 'app-statistics-panel-2', templateUrl: './statistics-
panel-2.component.html', styleUrls: ['./statistics-panel-2.component.scss'],
changeDetection: ChangeDetectionStrategy.OnPush }) export class
StatisticsPanel2Component implements OnInit, OnDestroy {



private readonly destroy$ = new Subject<void>();



// ============================================================================ //
Inputs // ============================================================================



/**



  • Domain configuration with chart data sources


   */
@Input() domainConfig!: DomainConfig<any, any, any>;



/**



  • Optional subset of chart IDs to display



  • If not provided, all charts from domainConfig.chartDataSources are shown


   */
@Input() chartIds?: string[];



/**
```

```
   • Function to check if a chart is popped out



   • Provided by parent component (DiscoverComponent)

    */
@Input() isPanelPoppedOut: (panelId: string) => boolean = () => false;



// ============================================================================ //
Outputs //
==============================================================================



/**



   • Emits when user clicks the pop-out button on a chart

    */
@Output() chartPopOut = new EventEmitter<string>();



/**



   • Emits when user clicks on a chart element

    */
@Output() chartClicked = new EventEmitter<{ event: { value: string; isHighlightMode:
boolean }; dataSource: ChartDataSource; }>();



// ============================================================================ //
State // ==============================================================================
```

```
/**


  • Ordered list of chart IDs for the grid


   */
chartOrder: string[] = [];



constructor( private readonly resourceService: ResourceManagementService<any, any,
any>, private readonly urlState: UrlStateService, private readonly popOutContext:
PopOutContextService, private readonly cdr: ChangeDetectorRef, private readonly
domainRegistry: DomainConfigRegistry, @Optional() private readonly route:
ActivatedRoute, @Optional() @Inject(IS_POPOUT_TOKEN) private readonly isPopout:
boolean ) {}



// ============================================================================ //
Observable Streams //
==========================================================================



get statistics$(): Observable<any | undefined> { return
this.resourceService.statistics$; }



get highlights$(): Observable<any> { return this.resourceService.highlights$; }



/**


  • Check if running in a pop-out window


   */
get isInPopOut(): boolean { return this.popOutContext.isInPopOut(); }
```

```
// ============================================================================ //
Lifecycle //
==============================================================================



ngOnInit(): void { // If domainConfig not provided via @Input (e.g., in popout), get
from registry if (!this.domainConfig) { this.domainConfig =
this.domainRegistry.getActive(); }



// Initialize chart order from chartIds input or domain config if (this.chartIds &&
this.chartIds.length > 0) { this.chartOrder = this.chartIds; } else if (this.isPopout
&& this.route) { // In popout: extract componentId from URL and map to chart IDs const
componentId = this.route.parent?.snapshot.paramMap.get('componentId') ?? null;
this.chartOrder = this.getChartIdsForStatisticsPanel(componentId); } else if
(this.domainConfig.chartDataSources) { this.chartOrder =
Object.keys(this.domainConfig.chartDataSources); } }



ngOnDestroy(): void { this.destroy$.next(); this.destroy$.complete(); }



// ============================================================================ //
Private Methods //
==============================================================================



/**



  • Map statistics panel ID to chart IDs

   *

  • This mapping determines which charts appear in which pop-out panel.



  • The default configuration splits charts into two panels.
```

```
   */
private getChartIdsForStatisticsPanel(panelId: string | null): string[] { const
chartIdMap: { [key: string]: string[] } = { 'statistics-1': ['manufacturer', 'top-
models'], 'statistics-2': ['body-class', 'year'] };



if (panelId && chartIdMap[panelId]) { return chartIdMap[panelId]; }



// Fallback to all charts if panel ID not recognized if
(this.domainConfig.chartDataSources) { return
Object.keys(this.domainConfig.chartDataSources); } return []; }



// ============================================================================ //
Event Handlers //
==============================================================================



/**



  • Handle chart drag-drop to reorder


   */
onChartDrop(event: CdkDragDrop<string[]>): void { moveItemInArray(this.chartOrder,
event.previousIndex, event.currentIndex); this.cdr.markForCheck(); }



/**



  • Handle chart pop-out request


   */
onChartPopOut(chartId: string): void { this.chartPopOut.emit(chartId); }
```

```
/**


  • Handle chart click


   *


  • Gets URL params from the data source and updates URL state.



  • When in a pop-out window, sends the update via message to parent.


   */
onChartClick(event: { value: string; isHighlightMode: boolean }, chartId: string):
void { const dataSource = this.domainConfig.chartDataSources?.[chartId]; if (!
dataSource) return;



// Delegate URL param generation to the data source const newParams =
dataSource.toUrlParams(event.value, event.isHighlightMode);



// Update URL (either directly or via pop-out message) if
(this.popOutContext.isInPopOut()) { this.popOutContext.sendMessage({ type:
PopOutMessageType.URL_PARAMS_CHANGED, payload: { params: newParams }, timestamp:
Date.now() }); } else { this.urlState.setParams(newParams); } }



/**


  • Get data source for a chart ID


   */
getDataSource(chartId: string): ChartDataSource | undefined { return
this.domainConfig.chartDataSources?.[chartId]; } }
```

Step 808.2: Create the Statistics Panel 2 Template

Create the file `src/app/framework/components/statistics-panel-2/statistics-panel-2.component.html`:

```
<!-- src/app/framework/components/statistics-panel-2/statistics-panel-2.component.html
-->
<!-- VERSION 1 (Section 808) - CDK horizontal chart grid -->



<div class="statistics-content"> <!-- No data message --> <div *ngIf="!(statistics$ |
async)" class="no-data-message"> <i class="pi pi-info-circle"></i> <p>No statistics
available. Add filters or select data to view distributions.</p> </div>



<!-- Chart Grid with CDK Drag-Drop --> <div *ngIf="statistics$ | async" cdkDropList
cdkDropListOrientation="horizontal" (cdkDropListDropped)="onChartDrop($event)"
class="chart-grid">



<ng-container *ngFor="let chartId of chartOrder"> <div *ngIf="getDataSource(chartId)
as dataSource" class="chart-box" cdkDrag> <!-- Drag Handle --> <div class="chart-drag-
handle" cdkDragHandle> <i class="pi pi-bars"></i> </div>



<!-- Chart or Placeholder --> <ng-container *ngIf="isInPopOut || !
isPanelPoppedOut('chart-' + chartId); else chartPoppedOut"> <app-base-chart
[dataSource]="dataSource" [statistics]="statistics$ | async" [highlights]="highlights$
| async" [selectedValue]="null" [canPopOut]="!
isInPopOut" (popOutClick)="onChartPopOut(chartId)" (chartClick)="onChartClick($event,
chartId)"> </app-base-chart> </ng-container>



<ng-template #chartPoppedOut> <div class="popout-placeholder"> <i class="pi pi-
external-link"></i> <span>Chart is open in a separate window</span> </div> </ng-
template> </div> </ng-container> </div> </div>
```

Step 808.3: Create the Statistics Panel 2 Styles

Create the file `src/app/framework/components/statistics-panel-2/statistics-panel-2.component.scss`:

```
// src/app/framework/components/statistics-panel-2/statistics-panel-2.component.scss

// VERSION 1 (Section 808) - CDK horizontal chart grid styles



// When in popout context, hide overflow to prevent scrollbars :host-
context(.statistics-2-popout) { display: block; overflow: hidden; height: 100%;



.statistics-content { overflow: hidden; height: 100%; }



.chart-grid { overflow: hidden; max-height: 100%; } }



.statistics-content { padding: 0.5rem; }



.no-data-message { display: flex; flex-direction: column; align-items: center;
justify-content: center; padding: 3rem 1rem; text-align: center; color: var(--text-
color-secondary);



i.pi { font-size: 3rem; margin-bottom: 1rem; opacity: 0.5; }



p { margin: 0; font-size: 1rem; } }



// Chart Grid with CDK Horizontal Drag-Drop .chart-grid { display: flex; flex-wrap:
wrap; gap: 1rem; padding: 0.5rem; background: var(--surface-ground); border-radius:
6px; border: 1px solid var(--surface-border); min-height: 200px;



.chart-box { // Each chart takes ~50% width minus gap (2 per row) flex: 0 0 calc(50% -
0.5rem); background: var(--surface-card); border-radius: 4px; padding: 1rem; position:
relative; box-sizing: border-box;
```

```
// Drag handle positioned in top-left .chart-drag-handle { position: absolute; top:
0.5rem; left: 0.5rem; cursor: move; padding: 0.25rem 0.5rem; color: var(--text-color-
secondary); opacity: 0.6; transition: opacity 0.2s, color 0.2s; z-index: 10;



&:hover { opacity: 1; color: var(--text-color); }



i { font-size: 1rem; } }



app-base-chart { display: block; height: 300px; }



.popout-placeholder { display: flex; align-items: center; justify-content: center;
gap: 0.75rem; height: 300px; background-color: var(--surface-ground); border: 2px
dashed var(--surface-border); border-radius: 4px; color: var(--text-color-secondary);
font-style: italic;



i { font-size: 1.5rem; color: var(--primary-color); } }



// CDK Drag states for chart boxes &.cdk-drag-preview { box-shadow: 0 5px 15px rgba(0,
0, 0, 0.3); opacity: 0.95; border: 2px solid var(--primary-color); }



&.cdk-drag-placeholder { opacity: 0.3; background: var(--surface-hover); border: 2px
dashed var(--surface-border); } } }



// Responsive adjustments @media (max-width: 1200px) { .chart-grid { .chart-box
{ flex: 0 0 100%; } } }
```

Step 808.4: Create the Module Export

Create the file `src/app/framework/components/statistics-panel-2/index.ts`:

```
// src/app/framework/components/statistics-panel-2/index.ts

// VERSION 1 (Section 808) - Barrel export



export { StatisticsPanel2Component } from './statistics-panel-2.component';
```

Step 808.5: Register in Framework Module

Open `src/app/framework/framework.module.ts` and add the component:

```
// src/app/framework/framework.module.ts

// VERSION 9 (Section 808) - Add StatisticsPanel2Component // Replaces VERSION 8 from
Section 807



import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/
common'; import { FormsModule } from '@angular/forms'; import { HttpClientModule }
from '@angular/common/http'; import { DragDropModule } from '@angular/cdk/drag-
drop'; // ... other imports



import { StatisticsPanel2Component } from './components/statistics-panel-2/statistics-
panel-2.component';



@NgModule({ declarations: [ // ... existing components StatisticsPanel2Component ],
imports: [ CommonModule, FormsModule, HttpClientModule, DragDropModule, // ... other
imports ], exports: [ // ... existing exports StatisticsPanel2Component ] }) export
class FrameworkModule {}
```

Verification

1. Build the Application

```
$ cd ~/projects/vvroom
$ ng build
```

Expected: Build succeeds with no errors.

2. Check TypeScript Compilation

```
$ npx tsc --noEmit
```

Expected: No type errors.

3. Verify CDK Integration

Check the template for proper CDK attributes:

- `cdkDropList` on the grid container
- `cdkDropListOrientation="horizontal"` for row-based dragging
- `cdkDrag` on each chart box
- `cdkDragHandle` on the drag handle

4. Verify Pop-out Integration

Check the component:

- `isInPopOut` getter uses `PopOutContextService`
- `onChartClick` sends messages via `PopOutContextService` when in pop-out
- `isPanelPoppedOut` input allows parent to track pop-out state

Common Problems

Symptom	Cause	Solution
Charts don't render	`domainConfig` not provided	Verify @Input or registry fallback
Drag not working	CDK module not imported	Add `DragDropModule` to imports
Pop-out placeholder not showing	`isPanelPoppedOut` not bound	Verify parent passes the function
Click not updating URL	Data source not found	Verify chartId matches key in `chartDataSources`
Charts overlap during drag	Missing CSS position	Ensure `.chart-box` has `position: relative`

Key Takeaways

- **Horizontal CDK orientation with wrap** — Flexbox handles row wrapping automatically
- **Pop-out awareness is bidirectional** — Component knows when it's in a pop-out AND when its charts are popped out
- **URL updates delegate to data sources** — The panel doesn't know URL structure; data sources handle that

---

Acceptance Criteria

- [ ] `StatisticsPanel2Component` accepts `domainConfig`, `chartIds`, and `isPanelPoppedOut` inputs
- [ ] Charts render in a 2-column grid (1 column on mobile)
- [ ] Charts can be reordered via horizontal drag-and-drop
- [ ] Drag handle appears in top-left corner of each chart
- [ ] Chart clicks update URL state via data source's `toUrlParams()`
- [ ] When in pop-out window, clicks send messages to parent
- [ ] Pop-out placeholder shows when a chart is in a separate window
- [ ] "No data" message shows when `statistics$` is null
- [ ] Component is registered in `FrameworkModule`

- [ ] `ng build` completes with no errors

Next Step

Proceed to `809-dockview-statistics-panel.md` to build the tabbed/dockable statistics panel using Dockview.

809: Dockview Statistics Panel Component

Status: Planning **Depends On:** 808-statistics-panel-2, 801-base-chart-component **Blocks:** 903-discover-page-component

---

Learning Objectives

After completing this section, you will:

- Understand how to integrate Dockview with Angular components
- Know how to move Angular-rendered elements into Dockview panels via DOM manipulation
- Be able to create tabbed, resizable panel layouts for data visualization

---

Objective

Build a statistics panel component that renders charts inside Dockview containers with tabbed, resizable, and draggable panels. This provides an alternative layout to the CDK grid, allowing users to arrange charts in a more flexible workspace.

Why

Different users prefer different layouts for data exploration:

- **Fixed grid** (StatisticsPanel2) — Simple, predictable, two columns
- **Dockable panels** (DockviewStatisticsPanel) — Flexible, tabbed, resizable

Dockview provides:

- **Tabbed panels** — Multiple charts in the same space, switch via tabs

- **Resizable splits** — Drag borders to adjust chart sizes
- **Drag to rearrange** — Drag tabs to reorder or split panels
- **Consistent dark theme** — Matches the application's visual style

Why Dockview Instead of Building Custom Panels?

Building a full-featured docking system is complex:

- Tab management with close/reorder
- Resizable split panes
- Drop zones for drag operations
- Serialization/restoration of layouts

Dockview provides all of this out of the box. We integrate it with Angular via a DOM manipulation pattern.

The DOM Manipulation Pattern

Dockview is framework-agnostic and expects to manage its own DOM. Angular components, however, are rendered by Angular's change detection. The pattern:

- Render Angular chart components in a hidden container
- When Dockview creates a panel, move the chart element into the panel
- When Dockview disposes a panel, move the chart back to the hidden container

This keeps Angular happy while letting Dockview manage the layout.

Angular Style Guide References

- Style 03-01: Use single responsibility principle
- Style 09-01: Use lifecycle hooks for setup/teardown

What

Step 809.1: Install Dockview

Install the Dockview core library:

```
$ cd ~/projects/vvroom
$ npm install dockview-core --save
```

Verify installation:

```
$ grep "dockview-core" package.json
"dockview-core": "^1.8.0",
```

Step 809.2: Add Dockview Styles to Angular.json

Dockview requires its CSS to be loaded globally. Open `angular.json` and add to the styles array:

```
{
"projects": { "vvroom": { "architect": { "build": { "options": { "styles":
[ "node_modules/dockview-core/dist/styles/dockview.css", "src/
styles.scss" ] } } } } } }
```

Step 809.3: Create the Dockview Statistics Panel Component TypeScript

Create the file `src/app/framework/components/dockview-statistics-panel/dockview-statistics-panel.component.ts`:

```
// src/app/framework/components/dockview-statistics-panel/dockview-statistics-
panel.component.ts

// VERSION 1 (Section 809) - Dockview tabbed chart panels



import { AfterViewInit, ChangeDetectionStrategy, ChangeDetectorRef, Component,
ElementRef, EventEmitter, Input, NgZone, OnDestroy, OnInit, Output, QueryList,
ViewChild, ViewChildren, ViewEncapsulation } from '@angular/core'; import { Subject }
from 'rxjs'; import { takeUntil } from 'rxjs/operators'; import { createDockview,
DockviewApi, IContentRenderer, themeDark } from 'dockview-core';



import { DomainConfig } from '../../models/domain-config.interface'; import
{ PopOutMessageType } from '../../models/popout.interface'; import
{ PopOutContextService } from '../../services/popout-context.service'; import
{ ResourceManagementService } from '../../services/resource-management.service';
import { UrlStateService } from '../../services/url-state.service'; import
{ DomainConfigRegistry } from '../../services/domain-config-registry.service'; import
{ ChartDataSource } from '../base-chart/chart-data.interface';



/**



  • Dockview Statistics Panel Component

 *

  • Renders charts in a dockview container with tabbed/split panel support.



  • Charts are rendered by Angular then moved into Dockview panels via DOM.

 *

  • @example



  •
```

html

- <app-dockview-statistics-panel
- [domainConfig]="domainConfig"
- [chartIds]="['manufacturer', 'top-models']">
- </app-dockview-statistics-panel>

```
 /
@Component({ selector: 'app-dockview-statistics-panel', templateUrl: './dockview-
statistics-panel.component.html', styleUrls: ['./dockview-statistics-
panel.component.scss'], changeDetection: ChangeDetectionStrategy.OnPush,
encapsulation: ViewEncapsulation.None // Required for Dockview CSS overrides }) export
class DockviewStatisticsPanelComponent implements OnInit, AfterViewInit, OnDestroy {



private readonly destroy$ = new Subject<void>(); private dockviewApi: DockviewApi |
null = null;



@ViewChild('dockviewContainer', { static: true }) dockviewContainer!:
ElementRef<HTMLDivElement>;



@ViewChildren('chartElement') chartElements!: QueryList<ElementRef<HTMLDivElement>>;



// ============================================================================= //
Inputs // ==============================================================================



/**



  • Domain configuration with chart data sources


   */
@Input() domainConfig!: DomainConfig<any, any, any>;



/**



  • Chart IDs to display in dockview panels


   */
```

```
@Input() chartIds: string[] = [];



/**



  • Function to check if a panel is popped out



  • Uses dockview-chart- prefix for chart panel IDs

   */
@Input() isPanelPoppedOut: (panelId: string) => boolean = () => false;



// ============================================================================ //
Outputs //
==============================================================================



/**



  • Emits when user clicks on a chart element

   */
@Output() chartClicked = new EventEmitter<{ event: { value: string; isHighlightMode:
boolean }; dataSource: ChartDataSource; }>();



/**



  • Emits when user clicks the pop-out button on a chart

   */
@Output() chartPopOut = new EventEmitter<string>();
```

```
// ============================================================================ //
State // ================================================================================



/**



  • Current statistics from resource service

   */
statistics: any = null;



/**



  • Current highlights from resource service

   */
highlights: any = {};



/**



  • Map of chartId to title (from data source)

   */
chartTitles: Map<string, string> = new Map();



constructor( private readonly resourceService: ResourceManagementService<any, any,
any>, private readonly urlState: UrlStateService, private readonly popOutContext:
PopOutContextService, private readonly cdr: ChangeDetectorRef, private readonly
ngZone: NgZone, private readonly domainRegistry: DomainConfigRegistry ) {}
```

```
// ============================================================================ //
Lifecycle //
==============================================================================



ngOnInit(): void { // If domainConfig not provided via @Input, get from registry if (!
this.domainConfig) { this.domainConfig = this.domainRegistry.getActive(); }



// Initialize chart titles from data sources this.chartIds.forEach(chartId => { const
dataSource = this.domainConfig.chartDataSources?.[chartId]; if (dataSource)
{ this.chartTitles.set(chartId, dataSource.getTitle()); } });



// Subscribe to statistics updates
this.resourceService.statistics$ .pipe(takeUntil(this.destroy$)) .subscribe(stats =>
{ this.statistics = stats; this.cdr.markForCheck(); });



// Subscribe to highlights updates
this.resourceService.highlights$ .pipe(takeUntil(this.destroy$)) .subscribe(highlights
=> { this.highlights = highlights; this.cdr.markForCheck(); }); }



ngAfterViewInit(): void { // Give Angular time to render chart components before
initializing Dockview setTimeout(() => { this.initializeDockview(); }, 100); }



ngOnDestroy(): void { this.destroy$.next(); this.destroy$.complete();



// Dispose dockview to prevent memory leaks if (this.dockviewApi)
{ this.dockviewApi.dispose(); } }



// ============================================================================ //
Dockview Setup //
==============================================================================
```

```
/**


  • Initialize the Dockview instance and populate panels

   *

  • This method:



  • 1. Creates a map of chart elements by their data-chart-id attribute



  • 2. Creates a Dockview instance with a custom component factory



  • 3. Adds panels for each chart, arranged side-by-side

   */
private initializeDockview(): void { const container =
this.dockviewContainer.nativeElement;



// Map to store chart elements by their ID const chartElementsMap = new Map<string,
HTMLElement>();



// Find all chart wrapper elements this.chartElements.forEach(el => { const chartId =
el.nativeElement.getAttribute('data-chart-id'); if (chartId)
{ chartElementsMap.set(chartId, el.nativeElement); } });



// Create dockview instance // Note: disableFloatingGroups prevents conflicts with
PopOutManagerService this.dockviewApi = createDockview(container,
{ disableFloatingGroups: true, theme: themeDark, createComponent: (options):
IContentRenderer => { const chartId = options.id; const chartElement =
chartElementsMap.get(chartId);
```

```
// Create a wrapper element for the panel content const wrapper =
document.createElement('div'); wrapper.className = 'dockview-chart-content';
wrapper.style.width = '100%'; wrapper.style.height = '100%'; wrapper.style.overflow =
'hidden';



// Move the chart element into the wrapper if (chartElement)
{ chartElement.style.display = 'block'; wrapper.appendChild(chartElement); }



return { element: wrapper, init: () => {}, dispose: () => { // Move chart element back
to hidden container on dispose if (chartElement) { const hiddenContainer =
document.querySelector('.charts-hidden-container'); if (hiddenContainer)
{ hiddenContainer.appendChild(chartElement); chartElement.style.display =
'none'; } } } }; } });



// Add panels for each chart - side by side layout this.chartIds.forEach((chartId,
index) => { const title = this.chartTitles.get(chartId) || chartId;



if (index === 0) { // First panel - add normally this.dockviewApi!.addPanel({ id:
chartId, title: title, component: 'chart' }); } else { // Subsequent panels - add to
the right of the first panel this.dockviewApi!.addPanel({ id: chartId, title: title,
component: 'chart', position: { referencePanel: this.chartIds[0], direction:
'right' } }); } });



// Force layout calculation after panels are added const rect =
container.getBoundingClientRect(); if (rect.width > 0 && rect.height > 0)
{ this.dockviewApi!.layout(rect.width, rect.height); }



this.cdr.markForCheck(); }



// ============================================================================ //
Event Handlers //
============================================================================
```

```
/**


  • Handle chart click


   *


  • Gets URL params from the data source and updates URL state.


   */
onChartClick(event: { value: string; isHighlightMode: boolean }, chartId: string):
void { const dataSource = this.domainConfig.chartDataSources?.[chartId]; if (!
dataSource) return;



// Delegate URL param generation to the data source const newParams =
dataSource.toUrlParams(event.value, event.isHighlightMode);



// Update URL (either directly or via pop-out message) if
(this.popOutContext.isInPopOut()) { this.popOutContext.sendMessage({ type:
PopOutMessageType.URL_PARAMS_CHANGED, payload: { params: newParams }, timestamp:
Date.now() }); } else { this.urlState.setParams(newParams); } }



/**


  • Handle chart pop-out request


   */
onChartPopOut(chartId: string): void { this.chartPopOut.emit(chartId); }



/**
```

```
   • Get data source for a chart ID

    */
getDataSource(chartId: string): ChartDataSource | undefined { return
this.domainConfig.chartDataSources?.[chartId]; }



/**



   • Check if a chart is popped out



   • Uses dockview-chart- prefix for panel IDs

    */
isChartPoppedOut(chartId: string): boolean { return this.isPanelPoppedOut(`dockview-
chart-${chartId}`); } }
```

Step 809.4: Create the Dockview Statistics Panel Template

Create the file `src/app/framework/components/dockview-statistics-panel/dockview-statistics-panel.component.html`:

```
<!-- src/app/framework/components/dockview-statistics-panel/dockview-statistics-
panel.component.html -->

<!-- VERSION 1 (Section 809) - Dockview tabbed chart panels -->



<div class="dockview-statistics-container"> <!-- Dockview container --> <div
#dockviewContainer class="dockview-wrapper"></div>



<!-- Chart components rendered in hidden container, moved into dockview panels via DOM
--> <div class="charts-hidden-container"> <ng-container *ngFor="let chartId of
chartIds"> <div #chartElement [attr.data-chart-id]="chartId" class="chart-wrapper"
style="display: none;"> <!-- Show chart or placeholder based on pop-out state --> <ng-
container *ngIf="!isChartPoppedOut(chartId); else chartPoppedOutPlaceholder"> <app-
base-chart *ngIf="getDataSource(chartId) as
dataSource" [dataSource]="dataSource" [statistics]="statistics" [highlights]="highligh
ts" [selectedValue]="null" [hideTitle]="true" [canPopOut]="true" (chartClick)="onChart
Click($event, chartId)" (popOutClick)="onChartPopOut(chartId)"> </app-base-chart> </
ng-container> <ng-template #chartPoppedOutPlaceholder> <div class="popout-
placeholder"> <i class="pi pi-external-link"></i> <span>Chart is open in a separate
window</span> </div> </ng-template> </div> </ng-container> </div> </div>
```

Step 809.5: Create the Dockview Statistics Panel Styles

Create the file `src/app/framework/components/dockview-statistics-panel/dockview-statistics-panel.component.scss`:

```
// src/app/framework/components/dockview-statistics-panel/dockview-statistics-
panel.component.scss

// VERSION 1 (Section 809) - Dockview styles with dark theme



// Dockview CSS is imported globally in angular.json



.dockview-statistics-container { width: 100%; height: 400px; position: relative; }



.dockview-wrapper { width: 100%; height: 100%; min-height: 400px;



// Ensure dockview takes full space > div { width: 100%; height: 100%; } }



// Hidden container keeps Angular components alive while Dockview manages
layout .charts-hidden-container { position: absolute; left: -9999px; top: -9999px;
visibility: hidden; pointer-events: none; }



.chart-wrapper { width: 100%; height: 100%; }



// Dockview dark theme overrides :host ::ng-deep { .dockview-theme-dark { --dv-
background-color: var(--surface-ground, #1a1a1a); --dv-paneview-header-border-color:
var(--surface-border, #333333); --dv-tabs-and-actions-container-background-color:
var(--surface-card, #252525); --dv-activegroup-visiblepanel-tab-background-color:
var(--surface-ground, #1a1a1a); --dv-activegroup-hiddenpanel-tab-background-color:
var(--surface-hover, #2d2d2d); --dv-inactivegroup-visiblepanel-tab-background-color:
var(--surface-hover, #2d2d2d); --dv-inactivegroup-hiddenpanel-tab-background-color:
var(--surface-card, #252525); --dv-tab-divider-color: var(--surface-border, #404040);
--dv-activegroup-visiblepanel-tab-color: var(--text-color, #ffffff); --dv-activegroup-
hiddenpanel-tab-color: var(--text-color-secondary, #999999); --dv-inactivegroup-
visiblepanel-tab-color: var(--text-color, #cccccc); --dv-inactivegroup-hiddenpanel-
tab-color: var(--text-color-secondary, #999999); --dv-separator-border: var(--surface-
border, #404040); --dv-paneview-header-background-color: var(--surface-card,
#252525); }
```

```
.dockview-vue, .dockview-react, .dockview { background-color: var(--surface-ground,
#1a1a1a); }



.groupview { background-color: var(--surface-ground, #1a1a1a); }



.tab { font-size: 12px; padding: 4px 12px; }



.dv-resize-container { background-color: var(--surface-ground, #1a1a1a); }



.content-container { background-color: var(--surface-ground, #1a1a1a); } }



.dockview-chart-content { width: 100%; height: 100%; background: var(--surface-ground,
#1a1a1a); overflow: hidden;



app-base-chart { display: block; width: 100%; height: 100%; } }



.popout-placeholder { display: flex; flex-direction: column; align-items: center;
justify-content: center; height: 100%; min-height: 200px; background: var(--surface-
ground, #1a1a1a); color: var(--text-color-secondary, #999999); gap: 12px;



i { font-size: 2rem; color: var(--text-color-secondary, #666666); }



span { font-size: 14px; } }
```

Step 809.6: Create the Module Export

Create the file `src/app/framework/components/dockview-statistics-panel/index.ts`:

```
// src/app/framework/components/dockview-statistics-panel/index.ts

// VERSION 1 (Section 809) - Barrel export



export { DockviewStatisticsPanelComponent } from './dockview-statistics-
panel.component';
```

Step 809.7: Register in Framework Module

Open `src/app/framework/framework.module.ts` and add the component:

```
// src/app/framework/framework.module.ts

// VERSION 10 (Section 809) - Add DockviewStatisticsPanelComponent // Replaces VERSION
9 from Section 808



import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/
common'; // ... other imports



import { DockviewStatisticsPanelComponent } from './components/dockview-statistics-
panel/dockview-statistics-panel.component';



@NgModule({ declarations: [ // ... existing components
DockviewStatisticsPanelComponent ], imports: [ CommonModule, // ... other imports ],
exports: [ // ... existing exports DockviewStatisticsPanelComponent ] }) export class
FrameworkModule {}
```

Understanding the DOM Manipulation Pattern

The hidden container pattern deserves special attention:

```
<!-- Charts render here (hidden from view) -->

<div class="charts-hidden-container"> <div #chartElement data-chart-id="manufacturer">
<app-base-chart [dataSource]="..." ...></app-base-chart> </div> </div>
```

When Dockview's `createComponent` is called:

```
createComponent: (options): IContentRenderer => {

const chartId = options.id; const chartElement = chartElementsMap.get(chartId);



// Move Angular component into Dockview panel wrapper.appendChild(chartElement);



return { element: wrapper, dispose: () => { // Move back to hidden container
hiddenContainer.appendChild(chartElement); } }; }
```

Why this works:

- Angular renders the chart components normally
- We physically move the DOM elements into Dockview panels
- Angular's change detection continues to work because the component instance is unchanged
- When Dockview disposes a panel, we move the element back to keep it alive

---

Verification

1. Build the Application

```
$ cd ~/projects/vvroom

$ ng build
```

Expected: Build succeeds with no errors.

2. Check Dockview Installation

```
$ npm list dockview-core
```

Expected: Shows dockview-core version installed.

3. Verify Global Styles

```
$ grep "dockview" angular.json
```

Expected: Shows `node_modules/dockview-core/dist/styles/dockview.css` in styles array.

4. Manual Inspection

Check the component template:

- `#dockviewContainer` is present
- `#chartElement` template references are correct
- `data-chart-id` attributes are bound

Common Problems

Symptom	Cause	Solution
Dockview not rendering	Container has zero height	Add `min-height: 400px` to `.dockview-wrapper`
Charts not visible in panels	DOM elements not moved	Check `chartElementsMap` population in console
Tabs not styled	Global CSS not loaded	Verify `angular.json` styles array
TypeError on dispose	Element already removed	Check for null in dispose callback
Floating panels appearing	`disableFloatingGroups` not set	Verify Dockview options

---

Key Takeaways

- **Dockview is framework-agnostic** — It manages DOM; we bridge to Angular via element moves
- **Hidden container pattern** — Keeps Angular components alive while Dockview manages layout
- **Disable floating groups** — Prevents conflicts with our custom PopOutManagerService

---

Acceptance Criteria

- [ ] `dockview-core` package is installed and listed in `package.json`
- [ ] Dockview CSS is loaded globally via `angular.json`
- [ ] `DockviewStatisticsPanelComponent` accepts `domainConfig` and `chartIds` inputs
- [ ] Charts render inside Dockview panels with tabs
- [ ] Panels can be resized by dragging borders
- [ ] Tabs can be dragged to reorder
- [ ] Chart clicks update URL state
- [ ] Pop-out placeholder shows when a chart is in a separate window
- [ ] Dark theme matches application styling
- [ ] Component is registered in `FrameworkModule`
- [ ] `ng build` completes with no errors

Next Step

Phase 8 (Framework Components) is now complete. Proceed to Phase 9 to build the feature components that wire everything together.

901: Home Component

Status: Planning **Depends On:** 102-app-shell, 103-routing **Blocks:** 907-final-integration

---

Learning Objectives

After completing this section, you will:

- Understand how to create a landing page component that serves as a domain hub
- Know how to use Angular's `RouterModule` for in-app navigation links
- Be able to apply CSS Grid to create responsive card layouts

---

Objective

Build the Home component — the application's main entry point that welcomes users and provides navigation to the automobile discovery features. This component establishes the visual identity of vvroom and guides users to explore automobile data.

Why

Every application needs a home base. The Home component serves several critical purposes:

- **First Impression** — Users land here first; it sets expectations for the entire application
- **Navigation Hub** — Provides clear pathways to key features without overwhelming users
- **Domain Introduction** — Explains what the application does before diving into specifics

Angular Style Guide References

- Style 02-01: Use consistent naming conventions (`home.component.ts`)

- Style 04-07: Create a folder for each feature component

Design Principles

The Home component is intentionally simple. It contains no business logic, no API calls, and no complex state management. It's pure presentation:

- Static content
- Navigation links
- Visual styling

This simplicity is deliberate. The Home component should load instantly and never fail. It's the fallback when something goes wrong elsewhere.

What

Step 901.1: Create the Home Component Directory

First, ensure the home feature directory exists:

```
$ cd ~/projects/vvroom
$ mkdir -p src/app/features/home
```

Step 901.2: Create the Home Component

Create `src/app/features/home/home.component.ts`:

```
// src/app/features/home/home.component.ts

// VERSION 1 (Section 901) - Complete home component with navigation



import { Component } from '@angular/core'; import { RouterModule } from '@angular/
router'; import { CommonModule } from '@angular/common';



/**



  • Home Component - Landing Page

 *

  • Serves as the main entry point for the vvroom application.



  • Provides navigation to the automobile discovery features.

 *

  • This component is intentionally simple - pure presentation



  • with no business logic or state management.

 */

@Component({ selector: 'app-home', templateUrl: './home.component.html', styleUrls:
['./home.component.scss'] }) export class HomeComponent {}
```

What this code does:

Element	Purpose
`selector: 'app-home'`	The HTML tag used to render this component
`templateUrl`	External HTML file for better separation of concerns
`styleUrls`	External SCSS file for component-specific styles
Empty class body	No logic needed — this is pure presentation

Step 901.3: Create the Home Component Template

Create `src/app/features/home/home.component.html`:

```
<!-- src/app/features/home/home.component.html -->

<!-- VERSION 1 (Section 901) - Home page with domain navigation -->



<div class="home-container"> <!-- Header Section --> <div class="header"> <h1>Vvroom</
h1> <p class="subtitle">Automobile Discovery Platform</p> </div>



<!-- Domain Cards Section --> <div class="domains-section"> <h2>Explore Data</h2> <div
class="domain-grid"> <a routerLink="/automobiles" class="domain-card"> <div
class="card-icon">🚗</div> <h3>Automobiles</h3> <p>Browse and analyze vehicle data
from thousands of manufacturers</p> </a> </div> </div>



<!-- Quick Start Section --> <div class="quickstart-section"> <h2>Quick Start</h2>
<div class="quickstart-grid"> <div class="quickstart-card"> <div class="step-
number">1</div> <h4>Select a Domain</h4> <p>Click on Automobiles above to begin
exploring vehicle data</p> </div> <div class="quickstart-card"> <div class="step-
number">2</div> <h4>Apply Filters</h4> <p>Use the query panel to narrow down results
by manufacturer, year, and more</p> </div> <div class="quickstart-card"> <div
class="step-number">3</div> <h4>Analyze Results</h4> <p>View charts, statistics, and
detailed data tables</p> </div> </div> </div> </div>
```

Template structure explained:

Section	Purpose
`.header`	Application branding and tagline
`.domains-section`	Navigation cards to feature areas
`.quickstart-section`	User guidance for first-time visitors

Note on `routerLink`: The `routerLink` directive from `RouterModule` creates navigation links that work with Angular's router. Unlike regular `href` attributes, `routerLink` doesn't cause a full page reload — it performs client-side navigation.

Step 901.4: Create the Home Component Styles

Create `src/app/features/home/home.component.scss`:

```
// src/app/features/home/home.component.scss
// VERSION 1 (Section 901) - Home page styles

.home-container { max-width: 1200px; margin: 0 auto; padding: 2rem; }

// Header styles .header { text-align: center; margin-bottom: 3rem; padding: 2rem 0;

h1 { font-size: 3rem; font-weight: 700; color: #1976d2; margin-bottom: 0.5rem; }

.subtitle { font-size: 1.25rem; color: #666; margin: 0; } }

// Domain cards section .domains-section { margin-bottom: 3rem;

h2 { font-size: 1.5rem; color: #333; margin-bottom: 1.5rem; text-align: center; } }

.domain-grid { display: grid; grid-template-columns: repeat(auto-fit, minmax(280px, 1fr)); gap: 1.5rem; justify-items: center; }

.domain-card { display: block; background: white; border-radius: 12px; padding: 2rem; text-decoration: none; color: inherit; box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1); transition: transform 0.2s, box-shadow 0.2s; max-width: 320px; width: 100%; text-align: center;

&:hover { transform: translateY(-4px); box-shadow: 0 8px 24px rgba(0, 0, 0, 0.15); text-decoration: none; }

.card-icon { font-size: 3rem; margin-bottom: 1rem; }
```

```
h3 { font-size: 1.25rem; color: #1976d2; margin-bottom: 0.5rem; }

p { font-size: 0.9rem; color: #666; margin: 0; line-height: 1.5; } }

// Quick start section .quickstart-section { background: #f8f9fa; border-radius: 12px;
padding: 2rem;

h2 { font-size: 1.5rem; color: #333; margin-bottom: 1.5rem; text-align: center; } }

.quickstart-grid { display: grid; grid-template-columns: repeat(auto-fit,
minmax(220px, 1fr)); gap: 1.5rem; }

.quickstart-card { background: white; border-radius: 8px; padding: 1.5rem; text-align:
center;

.step-number { width: 40px; height: 40px; background: #1976d2; color: white; border-
radius: 50%; display: flex; align-items: center; justify-content: center; font-size:
1.25rem; font-weight: 600; margin: 0 auto 1rem; }

h4 { font-size: 1rem; color: #333; margin-bottom: 0.5rem; }

p { font-size: 0.875rem; color: #666; margin: 0; line-height: 1.4; } }
```

CSS techniques used:

Technique	Purpose
`max-width` + `margin: auto`	Centers content with a maximum width
CSS Grid with `auto-fit`	Creates responsive columns that adapt to screen size
`translateY` on hover	Provides visual lift effect for interactive cards
`box-shadow` transitions	Smooth shadow changes on interaction
SCSS nesting	Keeps related styles organized and readable

Step 901.5: Register the Home Component in a Feature Module

For Angular 13 with NgModules, create a module for the home feature.

Create `src/app/features/home/home.module.ts`:

```
// src/app/features/home/home.module.ts

// VERSION 1 (Section 901) - Home feature module



import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/
common'; import { RouterModule } from '@angular/router';



import { HomeComponent } from './home.component';



@NgModule({ declarations: [ HomeComponent ], imports: [ CommonModule, RouterModule ],
exports: [ HomeComponent ] }) export class HomeModule {}
```

Module structure explained:

Property	Purpose
`declarations`	Components that belong to this module
`imports`	Other modules this module depends on
`exports`	Components available to other modules that import HomeModule

---

Step 901.6: Create the Index Barrel Export

Create `src/app/features/home/index.ts`:

```
// src/app/features/home/index.ts
// VERSION 1 (Section 901) - Barrel export for home feature


export * from './home.component'; export * from './home.module';
```

This barrel export allows clean imports elsewhere in the application:

```
import { HomeComponent, HomeModule } from './features/home';
```

Verification

1. Check File Structure

```
$ find src/app/features/home -type f | sort
```

Expected output:

```
src/app/features/home/home.component.html

src/app/features/home/home.component.scss src/app/features/home/home.component.ts src/
app/features/home/home.module.ts src/app/features/home/index.ts
```

2. Build the Application

```
$ ng build
```

Expected: Build succeeds with no errors. If you see errors about missing routes, that's expected — we'll configure routing in document 905.

3. Visual Verification (After Routing)

Once routing is configured (document 905), navigate to `http://localhost:4200/home`:

- Large "Vvroom" heading with blue color
- "Automobile Discovery Platform" subtitle
- Single domain card for Automobiles
- Three quick-start steps at the bottom
- Card lifts on hover with shadow effect

Common Problems

Symptom	Cause	Solution
Component not found error	Module not imported in AppModule	Add `HomeModule` to AppModule imports (done in document 906)
`routerLink` not working	RouterModule not imported	Ensure `RouterModule` is in the HomeModule imports
Styles not applying	Wrong file path in `styleUrls`	Verify the path matches the actual file location
Emoji not displaying	System font issue	Use an SVG icon instead, or accept platform variation
Grid layout broken	Browser doesn't support CSS Grid	Add fallback styles or use flexbox for older browsers

---

Key Takeaways

- **Feature components live in feature folders** — `src/app/features/home/` keeps all home-related files together
- **Simple components are powerful** — The Home component has no logic but serves a critical role
- **CSS Grid enables responsive layouts** — `auto-fit` and `minmax` create layouts that adapt to any screen size

---

Acceptance Criteria

- [ ] `src/app/features/home/home.component.ts` exists with proper decorator configuration
- [ ] `src/app/features/home/home.component.html` contains header, domain card, and quickstart sections
- [ ] `src/app/features/home/home.component.scss` contains responsive grid layout styles
- [ ] `src/app/features/home/home.module.ts` declares and exports HomeComponent
- [ ] `src/app/features/home/index.ts` provides barrel exports
- [ ] Component uses `routerLink` for navigation (not `href`)
- [ ] `ng build` completes without errors

Architecture Note

The Home component exemplifies the separation between **feature components** and **framework components**:

- **Feature components** (like Home) are page-level, application-specific
- **Framework components** (from Phase 8) are reusable across any domain

The Home component contains no domain logic — it's pure navigation and branding. This means:

- It loads instantly (no API calls)
- It never fails (no dependencies)
- It can be tested in isolation

This pattern will repeat for other feature components: they orchestrate framework components and provide page-level structure.

Next Step

Proceed to `902-automobile-landing-component.md` to build the automobile domain's landing page.

902: Automobile Landing Component

Status: Planning **Depends On:** 901-home-component, 607-domain-config-assembly **Blocks:** 903-discover-page-component

---

Learning Objectives

After completing this section, you will:

- Understand how to create a domain-specific landing page within a feature module
- Know how to provide context and navigation for a domain's features
- Be able to structure informational content that guides users to key functionality

---

Objective

Build the Automobile Landing component — the entry point to the automobile domain that introduces users to available features and guides them toward the discovery interface. This component bridges the gap between the general Home page and the feature-rich Discover page.

Why

When users click "Automobiles" from the Home page, they need orientation before diving into complex data exploration. The Automobile Landing component serves as this orientation layer:

- **Domain Context** — Tells users what kind of data they'll explore
- **Feature Preview** — Shows available capabilities before committing
- **Clear Call-to-Action** — Guides users to the primary feature (Discover)
- **Supporting Information** — Provides dataset stats and usage tips

The Landing Page Pattern

Many successful applications use a two-level navigation pattern:

```
Home (all domains) → Domain Landing (one domain) → Feature Page (specific task)
```

This progressive disclosure prevents cognitive overload. Users make one decision at a time:

- "I want to explore automobiles" (Home → Automobile Landing)
- "I want to search and analyze data" (Landing → Discover)

Angular Style Guide References

- Style 04-07: Feature areas should have their own folder
- Style 02-01: Name files with their feature name (`automobile`)

What

Step 902.1: Create the Automobile Feature Directory

```
$ cd ~/projects/vvroom
$ mkdir -p src/app/features/automobile
```

Step 902.2: Create the Automobile Landing Component

Create `src/app/features/automobile/automobile.component.ts`:

```
// src/app/features/automobile/automobile.component.ts

// VERSION 1 (Section 902) - Automobile domain landing page



import { Component } from '@angular/core'; import { RouterModule } from '@angular/
router';



/**



  • Automobile Component - Domain Landing Page

 *

  • Feature component serving as the entry point for the Automobile domain.



  • Provides navigation and context for automobile-related data exploration.

 *

  • This component:



  • - Introduces the automobile domain to users



  • - Previews available features (search, analytics, specs)



  • - Guides users to the Discover page for data exploration



  • - Displays dataset information and usage tips
```

```
 *

  • This is a pure presentation component with no business logic.



  • All data exploration happens in the Discover component.


 */
@Component({ selector: 'app-automobile', templateUrl: './automobile.component.html',
styleUrls: ['./automobile.component.scss'] }) export class AutomobileComponent {}
```

Step 902.3: Create the Automobile Landing Template

Create `src/app/features/automobile/automobile.component.html`:

```
<!-- src/app/features/automobile/automobile.component.html -->

<!-- VERSION 1 (Section 902) - Automobile landing page with feature preview -->



<div class="automobile-container"> <!-- Features Grid --> <div class="features-
section"> <h2><span class="section-icon">🚗</span> Explore Automobile Data</h2> <div
class="features-grid"> <!-- Primary Feature Card (Linked) --> <a routerLink="/
automobiles/discover" class="feature-card feature-card-primary"> <div class="feature-
icon">🔍</div> <h3>Advanced Search</h3> <p>Filter and search across thousands of
vehicles using advanced criteria</p> <span class="card-action">Start Exploring →</
span> </a>



<!-- Preview Feature Cards (Not Yet Linked) --> <div class="feature-card"> <div
class="feature-icon">📊</div> <h3>Analytics</h3> <p>View comprehensive statistics and
visualizations of vehicle data</p> </div> <div class="feature-card"> <div
class="feature-icon">🛠️</div> <h3>Detailed Specs</h3> <p>Access detailed
specifications for every vehicle in the database</p> </div> <div class="feature-card">
<div class="feature-icon">⚡</div> <h3>Performance</h3> <p>Compare performance metrics
and find the perfect vehicle match</p> </div> </div> </div>



<!-- Info Section --> <div class="info-section"> <div class="info-card info-card-
highlight"> <h3>Ready to Explore?</h3> <p>Click "Advanced Search" above to dive into
the full discovery experience with filters, charts, and data tables.</p> </div> <div
class="info-card"> <h3>Dataset Information</h3> <ul> <li><strong>Vehicle Count:</
strong> 55,000+ vehicles</li> <li><strong>Data Fields:</strong> Manufacturer, model,
year, body class, and more</li> <li><strong>Coverage:</strong> Comprehensive
automobile market data</li> </ul> </div> <div class="info-card"> <h3>Quick Tips</h3>
<ul> <li>Use filters to narrow down search results</li> <li>Hover over charts for
detailed information</li> <li>Pop-out panels for multi-monitor workflows</li> <li>URL
state persists across page reloads</li> </ul> </div> </div> </div>
```

Template structure explained:

Section	Purpose
`.features-section`	Grid of feature cards previewing capabilities
`.feature-card-primary`	The main call-to-action linking to Discover
Other `.feature-card` elements	Preview features (some may not be implemented yet)
`.info-section`	Supporting information cards

Design decision: Only the "Advanced Search" card links to a page. Other feature cards show what's possible but don't have links yet. This is intentional — it shows users the application's scope while guiding them to the implemented feature.

Step 902.4: Create the Automobile Landing Styles

Create `src/app/features/automobile/automobile.component.scss`:

```
// src/app/features/automobile/automobile.component.scss

// VERSION 1 (Section 902) - Automobile landing page styles



.automobile-container { max-width: 1200px; margin: 0 auto; padding: 2rem; }



// Features section .features-section { margin-bottom: 3rem;



h2 { font-size: 1.75rem; color: #333; margin-bottom: 1.5rem; display: flex; align-
items: center; gap: 0.5rem;



.section-icon { font-size: 1.5rem; } } }



.features-grid { display: grid; grid-template-columns: repeat(auto-fit, minmax(250px,
1fr)); gap: 1.5rem; }



.feature-card { background: white; border-radius: 12px; padding: 1.5rem; box-shadow: 0
2px 8px rgba(0, 0, 0, 0.08); transition: transform 0.2s, box-shadow 0.2s;



.feature-icon { font-size: 2.5rem; margin-bottom: 1rem; }



h3 { font-size: 1.125rem; color: #333; margin-bottom: 0.5rem; }



p { font-size: 0.9rem; color: #666; margin: 0; line-height: 1.5; } }



// Primary feature card (linked) .feature-card-primary { display: block; text-
decoration: none; color: inherit; border: 2px solid #1976d2; position: relative;
```

```
&:hover { transform: translateY(-4px); box-shadow: 0 8px 24px rgba(25, 118, 210, 0.2);
text-decoration: none; }



.card-action { display: block; margin-top: 1rem; color: #1976d2; font-weight: 600;
font-size: 0.9rem; } }



// Info section .info-section { display: grid; grid-template-columns: repeat(auto-fit,
minmax(280px, 1fr)); gap: 1.5rem; }



.info-card { background: white; border-radius: 12px; padding: 1.5rem; box-shadow: 0
2px 8px rgba(0, 0, 0, 0.08);



h3 { font-size: 1rem; color: #333; margin-bottom: 1rem; padding-bottom: 0.5rem;
border-bottom: 1px solid #eee; }



p { font-size: 0.9rem; color: #666; margin: 0; line-height: 1.6; }



ul { margin: 0; padding-left: 1.25rem;



li { font-size: 0.9rem; color: #666; margin-bottom: 0.5rem; line-height: 1.4;



strong { color: #333; }



&:last-child { margin-bottom: 0; } } } }



.info-card-highlight { background: linear-gradient(135deg, #1976d2 0%, #1565c0 100%);
color: white;
```

```
h3 { color: white; border-bottom-color: rgba(255, 255, 255, 0.2); }


p { color: rgba(255, 255, 255, 0.9); } }
```

Key styling patterns:

Pattern	Purpose
`.feature-card-primary` with border	Visual distinction for actionable card
`.info-card-highlight` with gradient	Draws attention to the call-to-action
Consistent `border-radius: 12px`	Establishes visual consistency across cards
`translateY` hover effect	Indicates clickability on interactive elements

---

Step 902.5: Create the Automobile Feature Module

Create `src/app/features/automobile/automobile.module.ts`:

```
// src/app/features/automobile/automobile.module.ts
// VERSION 1 (Section 902) - Automobile feature module


import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/
common'; import { RouterModule } from '@angular/router';


import { AutomobileComponent } from './automobile.component';


@NgModule({ declarations: [ AutomobileComponent ], imports: [ CommonModule,
RouterModule ], exports: [ AutomobileComponent ] }) export class AutomobileModule {}
```

Note: This module will grow in document 903 when we add the Discover component. For now, it only contains the landing component.

Step 902.6: Create the Index Barrel Export

Create `src/app/features/automobile/index.ts`:

```
// src/app/features/automobile/index.ts
// VERSION 1 (Section 902) - Barrel export for automobile feature


export * from './automobile.component'; export * from './automobile.module';
```

Verification

1. Check File Structure

```
$ find src/app/features/automobile -type f | sort
```

Expected output:

```
src/app/features/automobile/automobile.component.html
src/app/features/automobile/automobile.component.scss src/app/features/automobile/
automobile.component.ts src/app/features/automobile/automobile.module.ts src/app/
features/automobile/index.ts
```

2. Build the Application

```
$ ng build
```

Expected: Build succeeds with no errors.

3. Visual Verification (After Routing)

Once routing is configured (document 905), navigate to `http://localhost:4200/automobiles`:

- Section header with car emoji: "Explore Automobile Data"
- Four feature cards in a responsive grid
- "Advanced Search" card has blue border and "Start Exploring →" link
- Three info cards at the bottom
- Blue highlighted card inviting users to explore

4. Navigation Test

Click "Start Exploring →" on the Advanced Search card:

- URL should change to `/automobiles/discover`
- (The Discover component will be created in document 903)

Common Problems

Symptom	Cause	Solution
Grid layout shows one column	Container too narrow	Check `max-width` on `.automobile-container`
Primary card not visually distinct	Missing `.feature-card-primary` class	Verify HTML has both `feature-card` and `feature-card-primary` classes
Hover effect not working	CSS transition not applied	Check that `.feature-card-primary` has `transition` property
Link not navigating	Route not configured	Routes will be set up in document 905
Styles bleeding to other components	Missing component encapsulation	Ensure styles are in component-specific SCSS file

---

Key Takeaways

- **Landing pages provide context** — They orient users before complex features
- **Progressive disclosure reduces overwhelm** — Show features one level at a time
- **Visual hierarchy guides attention** — Use borders, colors, and hover effects to indicate primary actions

---

Acceptance Criteria

- [ ] `src/app/features/automobile/automobile.component.ts` exists with proper decorator
- [ ] `src/app/features/automobile/automobile.component.html` contains feature cards and info sections
- [ ] `src/app/features/automobile/automobile.component.scss` provides responsive grid styles
- [ ] `src/app/features/automobile/automobile.module.ts` declares and exports the component
- [ ] Primary feature card (Advanced Search) has visual distinction and links to `/automobiles/discover`
- [ ] Info cards display dataset information and quick tips
- [ ] `ng build` completes without errors

Architecture Note

The Automobile Landing component is the first **domain-specific feature component** we've built. Notice the pattern:

```
features/
├── home/                    # Application-level (no domain) │   └── home.component.ts
└── automobile/              # Domain-specific └── automobile.component.ts
```

The `home` feature is domain-agnostic — it navigates to domains but contains no domain logic. The `automobile` feature is domain-specific — it lives entirely within the automobile context.

This separation matters because:

- Home component never changes when automobile features change

- Automobile module can be lazy-loaded independently
- Future domains (if any) follow the same pattern

Next Step

Proceed to `903-discover-page-component.md` to build the main discovery interface where users explore automobile data with filters, charts, and tables.

903: Discover Page Component

Status: Planning **Depends On:** 902-automobile-landing-component, 306-resource-management-service, 801-809 (Framework Components) **Blocks:** 907-final-integration

---

Learning Objectives

After completing this section, you will:

- Understand how to orchestrate multiple framework components in a feature page
- Know how to use `ResourceManagementService` for URL-first state management
- Be able to implement drag-and-drop panel reordering with Angular CDK

Objective

Build the Automobile Discover component — the main data exploration interface that combines all framework components (query control, charts, tables, pickers) into a cohesive discovery experience. This is where users spend most of their time: filtering data, viewing statistics, and exploring results.

Why

The Discover component is the heart of vvroom. It demonstrates the power of the URL-First architecture by:

- **Orchestrating Components** — Combines query panel, charts, tables, and pickers
- **Managing State** — Uses `ResourceManagementService` with URL as single source of truth
- **Enabling Workflows** — Supports pop-out windows for multi-monitor setups
- **Providing Flexibility** — Allows users to reorder and collapse panels

The Orchestrator Pattern

Feature components like Discover don't contain UI logic themselves. Instead, they:

```
Feature Component (Orchestrator)

├── Injects services (ResourceManagementService, DomainConfig) ├── Coordinates
framework components ├── Handles cross-component communication └── Manages page-level
concerns (panel order, collapse state)
```

Each framework component (QueryControl, BaseChart, DynamicResultsTable) is self-contained. The Discover component wires them together and handles their outputs.

Angular Style Guide References

- Style 05-04: Delegate complex component logic to services
- Style 07-04: Use input/output properties for component communication

---

What

Step 903.1: Create the Discover Component Directory

```
$ cd ~/projects/vvroom

$ mkdir -p src/app/features/automobile/automobile-discover
```

Step 903.2: Create the Discover Component

Create `src/app/features/automobile/automobile-discover/automobile-discover.component.ts`:

```
// src/app/features/automobile/automobile-discover/automobile-discover.component.ts

// VERSION 1 (Section 903) - Main discovery page with framework component
orchestration



import { ChangeDetectionStrategy, ChangeDetectorRef, Component, Inject, Injector,
OnDestroy, OnInit } from '@angular/core'; import { Params } from '@angular/router';
import { Subject } from 'rxjs'; import { takeUntil } from 'rxjs/operators';



import { DomainConfig } from '../../../../framework/models/domain-config.interface';
import { PopOutMessageType } from '../../../../framework/models/popout.interface';
import { DOMAIN_CONFIG } from '../../../../framework/services/domain-config-
registry.service'; import { FilterOptionsService } from '../../../../framework/
services/filter-options.service'; import { PickerConfigRegistry } from '../../../../
framework/services/picker-config-registry.service'; import { PopOutManagerService }
from '../../../../framework/services/popout-manager.service'; import
{ ResourceManagementService } from '../../../../framework/services/resource-
management.service'; import { UrlStateService } from '../../../../framework/services/
url-state.service'; import { UserPreferencesService } from '../../../../framework/
services/user-preferences.service'; import { ChartDataSource } from '../../../../
framework/components/base-chart/base-chart.component'; import
{ createAutomobilePickerConfigs } from '../../../../domain-config/automobile/configs/
automobile.picker-configs';



/**



  • Automobile Discover Component

 *

  • Main discovery page for the Automobile domain. This component orchestrates



  • all framework components to provide a comprehensive data exploration experience.

 *

  • Responsibilities:
```

```
  • - Provides ResourceManagementService instance for URL-first state management

  • - Registers automobile-specific picker configurations

  • - Manages pop-out window communication

  • - Handles panel ordering and collapse state

  • - Coordinates events between child components
*
  • Child Components:

  • - QueryControlComponent: Filter management UI

  • - BasePickerComponent: Manufacturer-model hierarchical picker

  • - StatisticsPanel2Component: Multi-chart statistics display

  • - DynamicResultsTableComponent: Paginated data table

  • - BaseChartComponent: Individual chart visualizations
*/
```

```
@Component({ selector: 'app-automobile-discover', templateUrl: './automobile-
discover.component.html', styleUrls: ['./automobile-discover.component.scss'],
changeDetection: ChangeDetectionStrategy.OnPush, providers:
[ResourceManagementService, PopOutManagerService] }) export class
AutomobileDiscoverComponent implements OnInit, OnDestroy {



/* Domain configuration injected from registry */ domainConfig: DomainConfig<any, any,
any>;



/* Track which panels are collapsed */ collapsedPanels = new Map<string,
boolean>([ ['manufacturer-model-picker', true]  // Start picker collapsed ]);



/* Order of panels in the UI (user can reorder via drag-
drop) */ panelOrder: string[] = [ 'query-control', 'statistics-1', 'chart-body-class',
'chart-year', 'manufacturer-model-picker', 'results-table' ];



/* Unique picker configuration ID for this page instance */ readonly pickerConfigId =
'automobile-discover-manufacturer-model-picker';



private destroy$ = new Subject<void>(); private readonly gridId = 'automobile-
discover';



constructor( @Inject(DOMAIN_CONFIG) domainConfig: DomainConfig<any, any, any>, public
resourceService: ResourceManagementService<any, any, any>, private pickerRegistry:
PickerConfigRegistry, private injector: Injector, private popOutManager:
PopOutManagerService, private cdr: ChangeDetectorRef, private urlStateService:
UrlStateService, private userPreferences: UserPreferencesService, private
filterOptionsService: FilterOptionsService ) { this.domainConfig = domainConfig; }



ngOnInit(): void { // Register picker configurations for this page const pickerConfigs
= createAutomobilePickerConfigs(this.injector, 'automobile-discover');
this.pickerRegistry.registerMultiple(pickerConfigs);
```

```
// Initialize pop-out manager for this grid
this.popOutManager.initialize(this.gridId);



// Handle messages from pop-out windows
this.popOutManager.messages$ .pipe(takeUntil(this.destroy$)) .subscribe(({ panelId,
message }) => { this.handlePopOutMessage(panelId, message); });



// Update UI when pop-out windows close
this.popOutManager.closed$ .pipe(takeUntil(this.destroy$)) .subscribe(() =>
{ this.cdr.markForCheck(); });



// Broadcast state changes to pop-out windows
this.resourceService.state$ .pipe(takeUntil(this.destroy$)) .subscribe(state =>
{ const filterOptionsCache = this.filterOptionsService.getCache();
this.popOutManager.broadcastState(state, filterOptionsCache); });



// Sync filter options cache to pop-outs when it changes
this.filterOptionsService.getCache$() .pipe(takeUntil(this.destroy$)) .subscribe(cache
=> { if (this.popOutManager.getPoppedOutPanels().length > 0) { const state =
this.resourceService.getCurrentState(); this.popOutManager.broadcastState(state,
cache); } }); }



// ============================================================================ //
Panel State Management //
==============================================================================



/**



  • Check if a panel is currently popped out to a separate window


   */

isPanelPoppedOut(panelId: string): boolean { return
this.popOutManager.isPoppedOut(panelId); }
```

```
/**


  • Check if a panel is collapsed


   */
isPanelCollapsed(panelId: string): boolean { return
this.collapsedPanels.get(panelId) ?? false; }



/**


  • Toggle panel collapse state


   */
togglePanelCollapse(panelId: string): void { const currentState =
this.collapsedPanels.get(panelId) ?? false; this.collapsedPanels.set(panelId, !
currentState);



// Persist collapsed state const collapsedPanels =
Array.from(this.collapsedPanels.entries()) .filter(([_, isCollapsed]) =>
isCollapsed) .map(([id]) => id);
this.userPreferences.saveCollapsedPanels(collapsedPanels);



this.cdr.markForCheck(); }



/**


  • Handle panel drag-drop reordering


   */
```

```
onPanelDrop(event: { previousIndex: number; currentIndex: number }): void { const item
= this.panelOrder.splice(event.previousIndex, 1)[0];
this.panelOrder.splice(event.currentIndex, 0, item);
this.userPreferences.savePanelOrder(this.panelOrder); this.cdr.markForCheck(); }



/**



  • TrackBy function for panel ngFor


   */

trackByPanelId(index: number, panelId: string): string { return panelId; }



// ============================================================================ //
Panel Configuration //
==============================================================================



/**



  • Get human-readable title for a panel


   */

getPanelTitle(panelId: string): string { const titleMap: { [key: string]: string } =
{ 'query-control': 'Query Control', 'manufacturer-model-picker': 'Manufacturer-Model
Picker', 'statistics-1': 'Statistics', 'chart-body-class': 'Vehicles by Body Class',
'chart-year': 'Vehicles by Year', 'results-table': 'Results Table' }; return
titleMap[panelId] || panelId; }



/**



  • Get component type for a panel (used for pop-out routing)


   */
```

```
getPanelType(panelId: string): string { const typeMap: { [key: string]: string } =
{ 'query-control': 'query-control', 'manufacturer-model-picker': 'picker',
'statistics-1': 'statistics-2', 'chart-body-class': 'chart', 'chart-year': 'chart',
'results-table': 'basic-results' }; return typeMap[panelId] || panelId; }



/**



  • Get chart IDs for a statistics panel


   */

getChartIdsForPanel(panelId: string): string[] { const chartIdMap: { [key: string]:
string[] } = { 'statistics-1': ['manufacturer', 'top-models'] }; return
chartIdMap[panelId] || []; }



/**



  • Get chart data source by ID


   */

getChartDataSource(chartId: string): ChartDataSource | undefined { return
this.domainConfig.chartDataSources?.[chartId]; }



// ============================================================================ //
Pop-Out Management //
==============================================================================



/**



  • Open a panel in a pop-out window


   */
```

```
popOutPanel(panelId: string, panelType: string): void
{ this.popOutManager.openPopOut(panelId, panelType); this.cdr.markForCheck(); }



/**



  • Handle chart pop-out request


   */

onChartPopOut(chartId: string): void { const panelId = chart-${chartId};
this.popOutManager.openPopOut(panelId, 'chart'); this.cdr.markForCheck(); }



/**



  • Handle messages from pop-out windows


   */

private async handlePopOutMessage(_panelId: string, message: any): Promise<void>
{ switch (message.type) { case PopOutMessageType.PANEL_READY: // Pop-out is ready,
send current state const currentState = this.resourceService.getCurrentState(); const
currentCache = this.filterOptionsService.getCache();
this.popOutManager.broadcastState(currentState, currentCache); break;



case PopOutMessageType.URL_PARAMS_CHANGED: // Pop-out changed filters, update URL if
(message.payload?.params) { await
this.urlStateService.setParams(message.payload.params); } break;



case PopOutMessageType.CLEAR_ALL_FILTERS: await this.urlStateService.clearParams();
break;



case PopOutMessageType.PICKER_SELECTION_CHANGE: if (message.payload) { await
this.onPickerSelectionChangeAndUpdateUrl(message.payload); } break;
```

```
case PopOutMessageType.FILTER_ADD: if (message.payload?.params) { await
this.urlStateService.setParams({ ...message.payload.params, page: 1 }); } break;



case PopOutMessageType.FILTER_REMOVE: if (message.payload?.field) { await
this.urlStateService.setParams({ [message.payload.field]: null, page: 1 }); } break;



case PopOutMessageType.CHART_CLICK: if (message.payload) { const dataSource =
this.domainConfig.chartDataSources?.[message.payload.chartId]; await
this.onStandaloneChartClick( { value: message.payload.value, isHighlightMode:
message.payload.isHighlightMode }, dataSource ); } break; } }



// ============================================================================ //
Event Handlers //
==============================================================================



/**



  • Handle URL parameter changes from child components


   */
async onUrlParamsChange(params: Params): Promise<void> { await
this.urlStateService.setParams(params); }



/**



  • Handle clear all filters request


   */
async onClearAllFilters(): Promise<void> { await this.urlStateService.clearParams(); }



/**
```

```
  • Handle chart click events

   */
async onStandaloneChartClick( event: { value: string; isHighlightMode: boolean },
dataSource: ChartDataSource | undefined ): Promise<void> { if (!dataSource) return;



const newParams = dataSource.toUrlParams(event.value, event.isHighlightMode); if (!
event.isHighlightMode) { newParams['page'] = 1; }



if (Object.keys(newParams).length > 0) { await
this.urlStateService.setParams(newParams); } }



/**



  • Handle picker selection changes

   */
async onPickerSelectionChangeAndUpdateUrl(event: any): Promise<void> { const paramName
= 'modelCombos'; await this.urlStateService.setParams({ [paramName]: event.urlValue ||
null, page: 1 }); }



// ============================================================================ //
Lifecycle //
============================================================================



ngOnDestroy(): void { this.destroy$.next(); this.destroy$.complete(); } }
```

Key architectural points:

Aspect	Implementation
State Management	`ResourceManagementService` provided at component level
Pop-Out Support	`PopOutManagerService` coordinates window communication
Change Detection	`OnPush` strategy for optimal performance
Panel Ordering	Array-based order with drag-drop support
Event Handling	Async URL state updates for all user interactions

---

Step 903.3: Create the Discover Component Template

Create `src/app/features/automobile/automobile-discover/automobile-discover.component.html`:

```
<!-- src/app/features/automobile/automobile-discover/automobile-
discover.component.html -->

<!-- VERSION 1 (Section 903) - Discovery page with draggable panels -->



<div class="discover-container"> <!-- Header with Results Count --> <div
class="discover-header"> <h1>Automobile Discovery</h1> <span class="results-count">
{{ resourceService.totalResults$ | async }} {{ (resourceService.totalResults$ | async)
=== 1 ? 'result' : 'results' }} </span> </div>



<!-- Panels Container --> <div class="panels-container"> <div *ngFor="let panelId of
panelOrder; trackBy: trackByPanelId" class="panel-wrapper" [attr.id]="'panel-' +
panelId">



<!-- Panel Header --> <div class="panel-header"> <div class="panel-title-group"> <span
class="drag-handle">☰</span> <h3 class="panel-title">{{ getPanelTitle(panelId) }}</h3>
</div> <div class="panel-actions"> <button type="button" class="btn-
icon" (click)="togglePanelCollapse(panelId)" [attr.aria-
label]="isPanelCollapsed(panelId) ? 'Expand' :
'Collapse'" [title]="isPanelCollapsed(panelId) ? 'Expand' : 'Collapse'">
{{ isPanelCollapsed(panelId) ? '▶' : '▼' }} </button> <button *ngIf="!
isPanelPoppedOut(panelId)" type="button" class="btn-
icon" (click)="popOutPanel(panelId, getPanelType(panelId))" aria-label="Pop out to
separate window" title="Pop out to separate window"> ↗ </button> </div> </div>



<!-- Panel Content (shown when not collapsed) --> <div class="panel-content" *ngIf="!
isPanelCollapsed(panelId)">



<!-- Query Control Panel --> <ng-container *ngIf="panelId === 'query-control'"> <div
*ngIf="!isPanelPoppedOut('query-control'); else queryControlPoppedOut"> <app-query-
control
[domainConfig]="domainConfig" (urlParamsChange)="onUrlParamsChange($event)" (clearAllF
ilters)="onClearAllFilters()"> </app-query-control> </div> <ng-template
#queryControlPoppedOut> <div class="popout-placeholder"> <span class="popout-icon">↗</
span> <span>Query Control is open in a separate window</span> </div> </ng-template> </
ng-container>
```

```
<!-- Manufacturer-Model Picker Panel --> <ng-container *ngIf="panelId ===
'manufacturer-model-picker'"> <div *ngIf="!isPanelPoppedOut('manufacturer-model-
picker'); else pickerPoppedOut"> <app-base-picker
[configId]="pickerConfigId" (selectionChange)="onPickerSelectionChangeAndUpdateUrl($ev
ent)"> </app-base-picker> </div> <ng-template #pickerPoppedOut> <div class="popout-
placeholder"> <span class="popout-icon">↗</span> <span>Manufacturer-Model Picker is
open in a separate window</span> </div> </ng-template> </ng-container>



<!-- Statistics Panel --> <ng-container *ngIf="panelId === 'statistics-1'"> <div
*ngIf="!isPanelPoppedOut('statistics-1'); else statsPoppedOut"> <app-statistics-
panel-2
[domainConfig]="domainConfig" [chartIds]="getChartIdsForPanel('statistics-1')" [isPane
lPoppedOut]="isPanelPoppedOut.bind(this)" (chartPopOut)="onChartPopOut($event)"> </
app-statistics-panel-2> </div> <ng-template #statsPoppedOut> <div class="popout-
placeholder"> <span class="popout-icon">↗</span> <span>Statistics is open in a
separate window</span> </div> </ng-template> </ng-container>



<!-- Body Class Chart (Standalone) --> <ng-container *ngIf="panelId === 'chart-body-
class'"> <div *ngIf="!isPanelPoppedOut('chart-body-class') &&
getChartDataSource('body-class') as bodyClassDataSource; else bodyClassPoppedOut">
<app-base-chart
[dataSource]="bodyClassDataSource" [statistics]="resourceService.statistics$ |
async" [highlights]="resourceService.highlights$ |
async" [selectedValue]="null" (chartClick)="onStandaloneChartClick($event,
bodyClassDataSource)"> </app-base-chart> </div> <ng-template #bodyClassPoppedOut> <div
class="popout-placeholder"> <span class="popout-icon">↗</span> <span>Vehicles by Body
Class is open in a separate window</span> </div> </ng-template> </ng-container>



<!-- Year Chart (Standalone) --> <ng-container *ngIf="panelId === 'chart-year'"> <div
*ngIf="!isPanelPoppedOut('chart-year') && getChartDataSource('year') as
yearDataSource; else yearPoppedOut"> <app-base-chart
[dataSource]="yearDataSource" [statistics]="resourceService.statistics$ |
async" [highlights]="resourceService.highlights$ |
async" [selectedValue]="null" (chartClick)="onStandaloneChartClick($event,
yearDataSource)"> </app-base-chart> </div> <ng-template #yearPoppedOut> <div
class="popout-placeholder"> <span class="popout-icon">↗</span> <span>Vehicles by Year
is open in a separate window</span> </div> </ng-template> </ng-container>



<!-- Results Table Panel --> <ng-container *ngIf="panelId === 'results-table'"> <div
*ngIf="!isPanelPoppedOut('results-table'); else resultsTablePoppedOut"> <app-dynamic-
results-table [domainConfig]="domainConfig"> </app-dynamic-results-table> </div> <ng-
template #resultsTablePoppedOut> <div class="popout-placeholder"> <span class="popout-
```

```
icon">↗</span> <span>Results Table is open in a separate window</span> </div> </ng-
template> </ng-container>



</div> </div> </div> </div>
```

Template patterns explained:

Pattern	Purpose
`async` pipe	Subscribes to observables and handles cleanup automatically
`ng-template` with `#reference`	Defines placeholder content for popped-out panels
`*ngIf...else`	Conditionally shows component or placeholder
`trackBy` function	Optimizes `*ngFor` rendering performance
`bind(this)` on callback	Preserves `this` context when passing method as input

---

Step 903.4: Create the Discover Component Styles

Create `src/app/features/automobile/automobile-discover/automobile-discover.component.scss`:

```
// src/app/features/automobile/automobile-discover/automobile-discover.component.scss

// VERSION 1 (Section 903) - Discovery page layout and panel styles



.discover-container { max-width: 1400px; margin: 0 auto; padding: 1rem; }



// Header .discover-header { display: flex; align-items: baseline; gap: 1rem; margin-
bottom: 1.5rem; padding-bottom: 1rem; border-bottom: 1px solid #e0e0e0;



h1 { font-size: 1.5rem; font-weight: 600; color: #333; margin: 0; }



.results-count { font-size: 0.9rem; color: #666; } }



// Panels container .panels-container { display: flex; flex-direction: column; gap:
1rem; }



// Individual panel .panel-wrapper { background: white; border-radius: 8px; box-
shadow: 0 1px 3px rgba(0, 0, 0, 0.1); overflow: hidden; }



.panel-header { display: flex; align-items: center; justify-content: space-between;
padding: 0.75rem 1rem; background: #f5f5f5; border-bottom: 1px solid #e0e0e0; }



.panel-title-group { display: flex; align-items: center; gap: 0.75rem;



.drag-handle { cursor: grab; color: #999; font-size: 1rem; user-select: none;



&:hover { color: #666; } }
```

```
.panel-title { font-size: 0.95rem; font-weight: 600; color: #333; margin: 0; } }



.panel-actions { display: flex; gap: 0.25rem; }



.btn-icon { display: flex; align-items: center; justify-content: center; width: 28px;
height: 28px; border: none; background: transparent; border-radius: 4px; cursor:
pointer; font-size: 0.8rem; color: #666; transition: background-color 0.15s, color
0.15s;



&:hover { background: rgba(0, 0, 0, 0.08); color: #333; }



&:focus { outline: 2px solid #1976d2; outline-offset: 1px; } }



.panel-content { padding: 1rem; }



// Pop-out placeholder .popout-placeholder { display: flex; align-items: center;
justify-content: center; gap: 0.75rem; padding: 3rem 1rem; background: #f9f9f9;
border: 2px dashed #ddd; border-radius: 4px; color: #666; font-size: 0.9rem;



.popout-icon { font-size: 1.25rem; opacity: 0.6; } }



// Loading state :host ::ng-deep .loading-overlay { display: flex; align-items:
center; justify-content: center; padding: 2rem; color: #666; }



// Responsive adjustments @media (max-width: 768px) { .discover-container { padding:
0.5rem; }



.discover-header { flex-direction: column; gap: 0.25rem;
```

```
h1 { font-size: 1.25rem; } }


.panel-content { padding: 0.75rem; } }
```

Step 903.5: Update the Automobile Module

Update `src/app/features/automobile/automobile.module.ts` to include the Discover component:

```
// src/app/features/automobile/automobile.module.ts

// VERSION 2 (Section 903) - Add AutomobileDiscoverComponent // Replaces VERSION 1
from Section 902



import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/
common'; import { RouterModule } from '@angular/router';



import { AutomobileComponent } from './automobile.component'; import
{ AutomobileDiscoverComponent } from './automobile-discover/automobile-
discover.component';



// Framework component imports import { QueryControlComponent } from '../../../
framework/components/query-control/query-control.component'; import
{ BasePickerComponent } from '../../../framework/components/base-picker/base-
picker.component'; import { StatisticsPanel2Component } from '../../../framework/
components/statistics-panel-2/statistics-panel-2.component'; import
{ BaseChartComponent } from '../../../framework/components/base-chart/base-
chart.component'; import { DynamicResultsTableComponent } from '../../../framework/
components/dynamic-results-table/dynamic-results-table.component';



@NgModule({ declarations: [ AutomobileComponent, AutomobileDiscoverComponent ],
imports: [ CommonModule, RouterModule, // Framework components QueryControlComponent,
BasePickerComponent, StatisticsPanel2Component, BaseChartComponent,
DynamicResultsTableComponent ], exports: [ AutomobileComponent,
AutomobileDiscoverComponent ] }) export class AutomobileModule {}
```

Note: Framework components from Phase 8 are imported here. They should already be created as standalone components or have their own modules.

Step 903.6: Update Barrel Export

Update `src/app/features/automobile/index.ts`:

```
// src/app/features/automobile/index.ts

// VERSION 2 (Section 903) - Add discover component export



export * from './automobile.component'; export * from './automobile-discover/
automobile-discover.component'; export * from './automobile.module';
```

Verification

1. Check File Structure

```
$ find src/app/features/automobile -type f | sort
```

Expected output:

```
src/app/features/automobile/automobile-discover/automobile-discover.component.html

src/app/features/automobile/automobile-discover/automobile-discover.component.scss
src/app/features/automobile/automobile-discover/automobile-discover.component.ts src/
app/features/automobile/automobile.component.html src/app/features/automobile/
automobile.component.scss src/app/features/automobile/automobile.component.ts src/app/
features/automobile/automobile.module.ts src/app/features/automobile/index.ts
```

2. Build the Application

```
$ ng build
```

Expected: Build succeeds. If there are import errors for framework components, ensure Phase 8 is complete.

3. Visual Verification (After Routing)

Navigate to `http://localhost:4200/automobiles/discover`:

- Page header shows "Automobile Discovery" with results count

- Multiple panels visible (Query Control, Statistics, Charts, Table)
- Each panel has header with collapse/expand and pop-out buttons
- Clicking collapse button hides panel content
- Clicking pop-out button opens new window (requires pop-out routes)

4. URL State Test

- Apply a filter (e.g., select a manufacturer)
- Observe URL changes (e.g., `?manufacturer=Toyota`)
- Refresh the page
- Filter should persist (URL-first architecture in action)

Common Problems

Symptom	Cause	Solution
"No provider for ResourceManagementService"	Service not in providers array	Add to component's `providers` array
"No provider for DOMAIN_CONFIG"	Domain config not registered	Ensure domain providers are set up in AppModule
Framework components not found	Phase 8 components missing	Complete Phase 8 before this section
Pop-out not working	Pop-out routes not configured	Configure pop-out routes in document 904-905
Charts not rendering	Chart data sources not configured	Verify domain config has `chartDataSources` property
Filters not persisting	URL state service not working	Check UrlStateService initialization

---

Key Takeaways

- **Feature components orchestrate, not implement** — They wire together framework components

- **Component-level providers create isolated instances** — `ResourceManagementService` is fresh for each Discover instance
- **The async pipe is your friend** — It handles subscription management automatically

---

Acceptance Criteria

- [ ] `AutomobileDiscoverComponent` is created with proper providers
- [ ] Component template renders all panel types (query control, charts, table, picker)
- [ ] Panels can be collapsed and expanded
- [ ] Pop-out placeholder is shown when a panel is popped out
- [ ] Results count updates reactively from `ResourceManagementService`
- [ ] All child component inputs are properly bound
- [ ] Event handlers update URL state through `UrlStateService`
- [ ] `ng build` completes without errors

Architecture Note: The Flow of Data

The Discover component demonstrates the complete URL-First data flow:

```
User Action (click filter)

↓ Event Handler (onUrlParamsChange) ↓ UrlStateService.setParams() ↓ URL Updates
(browser address bar) ↓ ResourceManagementService watches URL ↓ Filters extracted →
API call made ↓ Response updates BehaviorSubject ↓ Components receive new data via
async pipe ↓ UI updates automatically
```

Every piece of state flows through the URL. This is the "Aha moment" of Phase 9: you've built an application where the URL is the single source of truth, and every component reacts to URL changes.

Next Step

Proceed to `904-popout-component.md` to build the pop-out window component that enables multi-monitor workflows.

904: Popout Component

Status: Planning **Depends On:** 307-popout-context-service, 308-popout-manager-service, 315-popout-token **Blocks:** 905-app-routing-module

---

Learning Objectives

After completing this section, you will:

- Understand how to create a layout component for pop-out windows
- Know how to use injection tokens to identify pop-out context
- Be able to synchronize state between main window and pop-out windows using BroadcastChannel

---

Objective

Build the Popout component — a minimal layout shell that hosts framework components in separate browser windows. This enables multi-monitor workflows where users can pop out charts, tables, or query panels to secondary displays while the main window remains focused on other tasks.

Why

Professional data analysis workflows often span multiple monitors. A user might want:

- **Primary monitor:** Main discovery interface with filters
- **Secondary monitor:** Large chart for detailed visualization
- **Third monitor:** Results table for data review

The Popout component makes this possible by:

- **Providing a Shell** — A minimal container for the popped-out component

- **Managing Context** — Setting up domain configuration for child components
- **Synchronizing State** — Receiving state updates from the main window
- **Forwarding Actions** — Sending user interactions back to the main window

The Pop-Out Architecture

Pop-out windows don't fetch data independently. Instead:

```
Main Window (source of truth)
```

```
├── Makes API calls ├── Manages URL state ├── Broadcasts state updates | └──
[BroadcastChannel] ──→ Pop-Out Window ├── Receives state ├── Renders UI └── Sends
actions back
```

This design ensures consistency: all data flows from one source, and pop-outs are essentially "remote views" of the main window's state.

Angular Style Guide References

- Style 05-02: Use services for shared functionality
- Style 07-01: Define small, focused components

What

Step 904.1: Create the Popout Feature Directory

```
$ cd ~/projects/vvroom
```

```
$ mkdir -p src/app/features/popout
```

Step 904.2: Create the Popout Component

Create `src/app/features/popout/popout.component.ts`:

```
// src/app/features/popout/popout.component.ts

// VERSION 1 (Section 904) - Pop-out window layout component



import { ChangeDetectionStrategy, ChangeDetectorRef, Component, OnDestroy, OnInit }
from '@angular/core'; import { ActivatedRoute } from '@angular/router'; import
{ Subject } from 'rxjs'; import { takeUntil } from 'rxjs/operators';



import { PopOutMessage, PopOutMessageType } from '../../../framework/models/
popout.interface'; import { DomainConfigRegistry } from '../../../framework/services/
domain-config-registry.service'; import { PopOutContextService } from '../../../
framework/services/popout-context.service'; import { ResourceManagementService } from
'../../../framework/services/resource-management.service'; import { IS_POPOUT_TOKEN }
from '../../../framework/tokens/popout.token';



/**



  • Popout Component - Pop-out Window Layout Shell

 *

  • A minimal layout component for pop-out windows. This component:



  • 1. Provides ResourceManagementService and IS_POPOUT_TOKEN to children



  • 2. Sets the active domain so child components can inject domain config



  • 3. Syncs state from main window via BroadcastChannel



  • 4. Renders child components via router-outlet
```

```
 *

  • The popout has NO knowledge of specific component types. Child routes



  • determine which component is rendered. Each child component is responsible



  • for its own behavior based on injected services.

 *

  • URL structure: /popout/:gridId/:componentId/:type



  • - gridId: Identifies the source grid (e.g., 'automobile-discover')



  • - componentId: Unique ID of the panel being popped out



  • - type: Component type for routing (e.g., 'chart', 'picker', 'query-control')

 */
@Component({ selector: 'app-popout', templateUrl: './popout.component.html',
styleUrls: ['./popout.component.scss'], changeDetection:
ChangeDetectionStrategy.OnPush, providers: [ ResourceManagementService, { provide:
IS_POPOUT_TOKEN, useValue: true } ] }) export class PopoutComponent implements OnInit,
OnDestroy { private destroy$ = new Subject<void>();



constructor( private route: ActivatedRoute, private popOutContext:
PopOutContextService, private cdr: ChangeDetectorRef, private resourceService:
ResourceManagementService<any, any, any>, private domainRegistry:
DomainConfigRegistry ) {}
```

```
ngOnInit(): void { // Extract route parameters and initialize
this.route.params .pipe(takeUntil(this.destroy$)) .subscribe(params => { const gridId
= params['gridId']; const componentId = params['componentId'];



// Extract domain name from gridId (e.g., 'automobile-discover' -> 'automobile') const
domainName = gridId.split('-')[0] || 'automobile';



// Set active domain for child components this.domainRegistry.setActive(domainName);



// Initialize BroadcastChannel communication
this.popOutContext.initializeAsPopOut(componentId);



// Add pop-out styling classes to document
document.documentElement.classList.add('popout-html');
document.body.classList.add('popout-body'); });



// Handle messages from the main window this.popOutContext .getMessages$
() .pipe(takeUntil(this.destroy$)) .subscribe(message =>
{ this.handleMessage(message); }); }



/**



  • Handle incoming messages from the main window


   */
private handleMessage(message: PopOutMessage): void { switch (message.type) { case
PopOutMessageType.CLOSE_POPOUT: // Main window requested this pop-out to close
window.close(); break;



case PopOutMessageType.STATE_UPDATE: // Main window sent updated state if
(message.payload?.state)
```

```
{ this.resourceService.syncStateFromExternal(message.payload.state);
this.cdr.detectChanges(); } break; } }



ngOnDestroy(): void { this.destroy$.next(); this.destroy$.complete(); } }
```

Key design decisions:

Decision	Rationale
`IS_POPOUT_TOKEN` provided as `true`	Child components can check if they're in a pop-out
`ResourceManagementService` provided here	Creates isolated instance that receives state from main
Domain extracted from `gridId`	Enables domain-agnostic pop-out support
`OnPush` change detection	Requires explicit `detectChanges()` after state sync

Step 904.3: Create the Popout Component Template

Create `src/app/features/popout/popout.component.html`:

```
<!-- src/app/features/popout/popout.component.html -->
<!-- VERSION 1 (Section 904) - Minimal pop-out layout -->



<div class="popout-container"> <router-outlet></router-outlet> </div>
```

The template is intentionally minimal. The `router-outlet` renders whatever child component is specified by the route (chart, table, picker, etc.).

Step 904.4: Create the Popout Component Styles

Create `src/app/features/popout/popout.component.scss`:

```
// src/app/features/popout/popout.component.scss

// VERSION 1 (Section 904) - Pop-out window styles



// Full-height container .popout-container { display: flex; flex-direction: column;
min-height: 100vh; padding: 1rem; background: #f5f5f5; }



// Global styles applied via class on html/body :host-context(.popout-html) { //
Remove default margins margin: 0; padding: 0; }



:host-context(.popout-body) { margin: 0; padding: 0; overflow: auto; }



// Ensure child components take full space ::ng-deep { .popout-container > * { flex:
1; display: flex; flex-direction: column; }



// Charts should fill available space app-base-chart { flex: 1; min-height: 400px; }



// Tables should scroll internally app-dynamic-results-table, app-basic-results-table
{ flex: 1; overflow: auto; }



// Query control takes its natural height app-query-control { flex: none; } }
```

Step 904.5: Create the Popout Routes

Create `src/app/features/popout/popout.routes.ts`:

```
// src/app/features/popout/popout.routes.ts
// VERSION 1 (Section 904) - Child routes for pop-out window

import { Routes } from '@angular/router';

/**

  • Popout Child Routes
 *
  • Defines the child routes for the pop-out layout. Each route maps a component

  • type (from the URL) to the actual component that should be rendered.
 *
  • URL structure: /popout/:gridId/:componentId/:type

  • Where :type is one of these child route paths.
 *
  • Note: For Angular 13 with NgModules, components are referenced directly

  • rather than using dynamic imports (loadComponent).
 */
export const POPOUT_ROUTES: Routes = [ { path: 'query-control', loadChildren: () =>
import('../../../framework/components/query-control/query-control.module') .then(m =>
m.QueryControlModule) }, { path: 'picker', loadChildren: () => import('../../../
```

```
framework/components/base-picker/base-picker.module') .then(m =>
m.BasePickerModule) }, { path: 'chart', loadChildren: () => import('../../../
framework/components/base-chart/base-chart.module') .then(m => m.BaseChartModule) },
{ path: 'basic-results', loadChildren: () => import('../../../framework/components/
dynamic-results-table/dynamic-results-table.module') .then(m =>
m.DynamicResultsTableModule) }, { path: 'statistics-2', loadChildren: () =>
import('../../../framework/components/statistics-panel-2/statistics-
panel-2.module') .then(m => m.StatisticsPanel2Module) } ];
```

Note on Angular 13 patterns: This uses `loadChildren` with module imports, which is the NgModule pattern. The exact implementation depends on how framework components are structured in Phase 8. If components are standalone, use `loadComponent` instead.

Step 904.6: Create the Popout Module

Create `src/app/features/popout/popout.module.ts`:

```
// src/app/features/popout/popout.module.ts

// VERSION 1 (Section 904) - Pop-out feature module



import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/
common'; import { RouterModule } from '@angular/router';



import { PopoutComponent } from './popout.component'; import { POPOUT_ROUTES } from
'./popout.routes';



@NgModule({ declarations: [ PopoutComponent ], imports: [ CommonModule,
RouterModule.forChild([ { path: '', component: PopoutComponent, children:
POPOUT_ROUTES } ]) ] }) export class PopoutModule {}
```

Step 904.7: Create the Barrel Export

Create `src/app/features/popout/index.ts`:

```
// src/app/features/popout/index.ts

// VERSION 1 (Section 904) - Barrel export for popout feature



export * from './popout.component'; export * from './popout.routes'; export * from './
popout.module';
```

Step 904.8: Add Global Pop-out Styles (Optional)

For consistent pop-out appearance, add these global styles to `src/styles.scss`:

```
// src/styles.scss

// ADD these styles for pop-out window support



// Pop-out window specific styles .popout-html { height: 100%; margin: 0; padding:
0; }



.popout-body { height: 100%; margin: 0; padding: 0; background: #f5f5f5; font-family:
-apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Oxygen, Ubuntu, sans-serif; }
```

Verification

1. Check File Structure

```
$ find src/app/features/popout -type f | sort
```

Expected output:

```
src/app/features/popout/index.ts

src/app/features/popout/popout.component.html src/app/features/popout/
popout.component.scss src/app/features/popout/popout.component.ts src/app/features/
popout/popout.module.ts src/app/features/popout/popout.routes.ts
```

2. Build the Application

```
$ ng build
```

Expected: Build succeeds.

3. Manual Pop-out Test (After Full Integration)

- Navigate to `/automobiles/discover`
- Click the pop-out button on any panel
- A new browser window should open with URL like:

`/popout/automobile-discover/chart-year/chart`

- The component should render in the new window
- State changes in main window should reflect in pop-out

4. Message Flow Test

In browser console of main window:

```
// After pop-out is open, check BroadcastChannel

console.log('Pop-out should receive state updates');
```

In pop-out window console:

```
// State should sync from main

console.log('Check that data appears');
```

Common Problems

Symptom	Cause	Solution
Pop-out window is blank	Router-outlet not finding child route	Verify route path matches child routes
"No provider for DOMAIN_CONFIG"	Domain not set before component loads	Ensure `domainRegistry.setActive()` called in ngOnInit
State not syncing	BroadcastChannel not initialized	Check `popOutContext.initializeAsPopOut()` called
Change detection not updating	OnPush strategy blocking updates	Call `cdr.detectChanges()` after state sync
Window.close() not working	Browser security restrictions	User must have opened the window (not blocked)

Key Takeaways

- **Pop-outs are state receivers, not sources** — They display data but don't fetch it
- **BroadcastChannel enables cross-window communication** — Built into modern browsers
- **Injection tokens differentiate contexts** — `IS_POPOUT_TOKEN` tells components where they're running

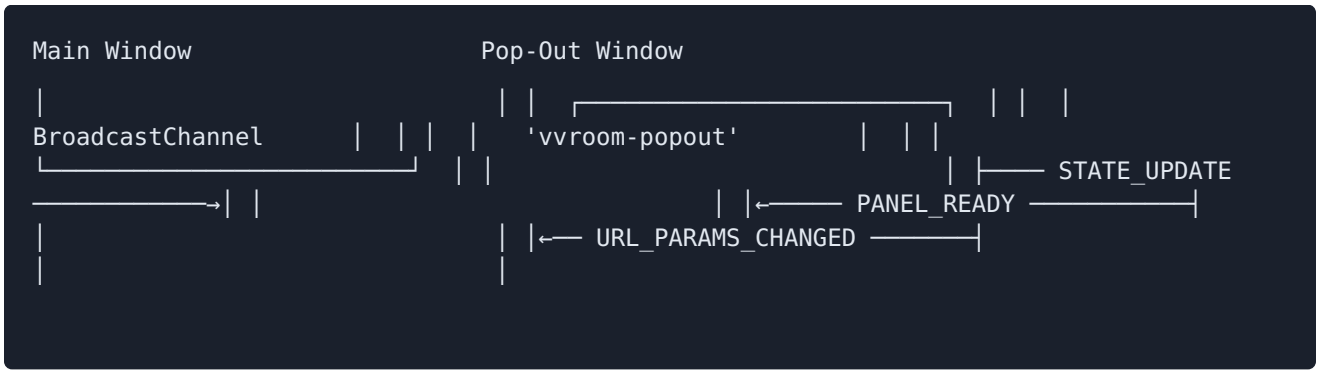
Acceptance Criteria

- [ ] `PopoutComponent` is created with `IS_POPOUT_TOKEN` provider
- [ ] Component extracts `gridId` and `componentId` from route params
- [ ] Domain is set via `DomainConfigRegistry.setActive()`
- [ ] BroadcastChannel communication initialized via `PopOutContextService`
- [ ] State updates from main window sync to `ResourceManagementService`
- [ ] Child routes defined for all pop-out-able component types

- [ ] `ng build` completes without errors

Architecture Note: Cross-Window Communication

The pop-out system uses the browser's `BroadcastChannel` API for communication:



Message types:

Type	Direction	Purpose
`STATE_UPDATE`	Main → Pop-out	Send current state (filters, results, statistics)
`PANEL_READY`	Pop-out → Main	Pop-out initialized, ready for state
`URL_PARAMS_CHANGED`	Pop-out → Main	User changed filter in pop-out
`CLOSE_POPOUT`	Main → Pop-out	Request pop-out to close

This architecture ensures the main window remains the source of truth while pop-outs provide additional viewport space.

Next Step

Proceed to `905-app-routing-module.md` to configure the application's routing, connecting all feature components into a navigable application.

905: App Routing Module

Status: Planning **Depends On:** 901-home-component, 902-automobile-landing-component, 903-discover-page-component, 904-popout-component **Blocks:** 906-app-module

---

Learning Objectives

After completing this section, you will:

- Understand how to configure Angular Router with lazy loading for feature modules
- Know how to structure routes with nested children for complex navigation
- Be able to implement route guards and redirects for improved UX

Objective

Configure the application's routing module to connect all feature components into a cohesive, navigable application. This is where the individual pieces we've built come together as a unified experience.

Why

The routing module is the skeleton of your application. It defines:

- **Navigation Paths** — What URLs map to which components
- **Loading Strategy** — Which features load immediately vs. on demand
- **User Experience** — Default routes, redirects, and error handling

Angular Router Key Concepts

Concept	Purpose
`Routes`	Array of route configurations
`RouterModule.forRoot()`	Configures router at application level
`loadChildren`	Lazy-loads a feature module on navigation
`redirectTo`	Redirects one path to another
`pathMatch`	Determines how path is matched ('full' or 'prefix')

Angular Style Guide References

- Style 04-10: Use redirects for default routes
- Style 04-11: Consider lazy loading for large features

URL-First Architecture

The router is central to URL-First architecture. Every view in the application corresponds to a URL, and the URL is the source of truth for application state. The routing module defines what those URLs are.

What

Step 905.1: Create the App Routing Module

Create `src/app/app-routing.module.ts`:

```
// src/app/app-routing.module.ts

// VERSION 1 (Section 905) - Complete application routing configuration



import { NgModule } from '@angular/core'; import { RouterModule, Routes } from
'@angular/router';



/**



  • Application Routes Configuration


 *


  • Defines all navigation paths for the vvroom application.



  • Uses lazy loading for feature modules to improve initial load time.


 *


  • Route Structure:



  • - / (root): Redirects to /home



  • - /home: Home landing page



  • - /automobiles: Automobile domain landing



  • - /automobiles/discover: Main discovery interface
```

```
  • - /popout/:gridId/:componentId/:type: Pop-out window routes


 */
const routes: Routes = [ // Default route - redirect to home { path: '', redirectTo:
'home', pathMatch: 'full' },



// Home page (eager loaded for instant landing) { path: 'home', loadChildren: () =>
import('./features/home/home.module') .then(m => m.HomeModule) },



// Automobile domain routes (lazy loaded) { path: 'automobiles', loadChildren: () =>
import('./features/automobile/automobile.module') .then(m => m.AutomobileModule) },



// Pop-out window routes (lazy loaded) { path: 'popout/:gridId/:componentId',
loadChildren: () => import('./features/popout/popout.module') .then(m =>
m.PopoutModule) },



// Wildcard route - redirect unknown paths to home { path: '**', redirectTo:
'home' } ];



@NgModule({ imports: [ RouterModule.forRoot(routes, { // Enable URL fragment scrolling
anchorScrolling: 'enabled', // Scroll to top on navigation scrollPositionRestoration:
'enabled', // Preserve query params on redirect (important for URL-First)
paramsInheritanceStrategy: 'always' }) ], exports: [RouterModule] }) export class
AppRoutingModule {}
```

Route configuration explained:

Route	Purpose	Loading Strategy
`''`	Root path, redirects to home	Immediate redirect
`'home'`	Home landing page	Lazy loaded
`'automobiles'`	Automobile feature area	Lazy loaded
`'popout/:gridId/:componentId'`	Pop-out windows	Lazy loaded
`'**'`	Catch-all for unknown routes	Redirect to home

Step 905.2: Update Feature Modules for Routing

Each feature module needs internal routes. Update the modules created earlier.

Update Home Module

Update `src/app/features/home/home.module.ts`:

```
// src/app/features/home/home.module.ts

// VERSION 2 (Section 905) - Add routing configuration // Replaces VERSION 1 from
Section 901



import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/
common'; import { RouterModule, Routes } from '@angular/router';



import { HomeComponent } from './home.component';



const routes: Routes = [ { path: '', component: HomeComponent } ];



@NgModule({ declarations: [ HomeComponent ], imports: [ CommonModule,
RouterModule.forChild(routes) ] }) export class HomeModule {}
```

Update Automobile Module

Update `src/app/features/automobile/automobile.module.ts`:

```
// src/app/features/automobile/automobile.module.ts

// VERSION 3 (Section 905) - Add routing configuration // Replaces VERSION 2 from
Section 903



import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/
common'; import { RouterModule, Routes } from '@angular/router';



import { AutomobileComponent } from './automobile.component'; import
{ AutomobileDiscoverComponent } from './automobile-discover/automobile-
discover.component';



// Framework component imports (from Phase 8) import { QueryControlModule } from
'../../framework/components/query-control/query-control.module'; import
{ BasePickerModule } from '../../framework/components/base-picker/base-picker.module';
import { StatisticsPanel2Module } from '../../framework/components/statistics-panel-2/
statistics-panel-2.module'; import { BaseChartModule } from '../../framework/
components/base-chart/base-chart.module'; import { DynamicResultsTableModule } from
'../../framework/components/dynamic-results-table/dynamic-results-table.module';



const routes: Routes = [ { path: '', component: AutomobileComponent }, { path:
'discover', component: AutomobileDiscoverComponent } ];



@NgModule({ declarations: [ AutomobileComponent, AutomobileDiscoverComponent ],
imports: [ CommonModule, RouterModule.forChild(routes), // Framework component modules
QueryControlModule, BasePickerModule, StatisticsPanel2Module, BaseChartModule,
DynamicResultsTableModule ] }) export class AutomobileModule {}
```

Step 905.3: Understanding Route Parameters

The pop-out route uses parameters to identify what component to show:

```
/popout/:gridId/:componentId/:type
|       |            | |       |            └── Component type (chart, picker, etc.)
|       └── Unique panel identifier └── Source grid identifier (e.g., automobile-
discover)
```

Example URLs:

URL	Meaning
`/popout/automobile-discover/chart-year/chart`	Year chart from automobile discover
`/popout/automobile-discover/query-control/query-control`	Query control panel
`/popout/automobile-discover/results-table/basic-results`	Results table

The PopoutComponent extracts these parameters to:

- Set the correct domain (`automobile`)
- Initialize BroadcastChannel with the right ID (`chart-year`)
- Load the appropriate child component via router

Step 905.4: Update AppComponent for Router Outlet

Ensure `src/app/app.component.ts` includes the router outlet:

```
// src/app/app.component.ts

// VERSION 3 (Section 905) - Router outlet support // Replaces VERSION 2 from Section
102



import { Component, OnInit, OnDestroy } from '@angular/core'; import { Router,
NavigationEnd } from '@angular/router'; import { Subject } from 'rxjs'; import
{ filter, takeUntil } from 'rxjs/operators';



/**



  • Root Application Component

 *

  • Main container for the vvroom application. Provides:



  • - Application shell with header navigation



  • - Router outlet for feature components



  • - Pop-out detection to hide header in pop-out windows

 */
@Component({ selector: 'app-root', template:  <!-- Navigation Header (hidden in
pop-outs) --> <header class="app-header" *ngIf="!isPopOut"> <div class="app-
header-brand"> <a routerLink="/home" class="brand-link"> <span class="app-
header-logo">🚗</span> <span class="app-header-title">vvroom</span> </a> </
div> <nav class="app-header-nav"> <a routerLink="/home"
routerLinkActive="active" class="nav-link">Home</a> <a routerLink="/
automobiles" routerLinkActive="active" class="nav-link">Automobiles</a> </nav>
</header>
```

```
<!-- Main Content Area --> <main class="app-content" [class.popout-content]="isPopOut"> <router-outlet></router-outlet> </main>
,
styles: [ :host { display: flex; flex-direction: column; min-height: 100vh; }
.app-header { display: flex; align-items: center; justify-content: space-between; padding: 0 1.5rem; height: 56px; background-color: #1976d2; color: white; box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1); }
.app-header-brand { display: flex; align-items: center; }
.brand-link { display: flex; align-items: center; gap: 0.5rem; color: white; text-decoration: none; }
.app-header-logo { font-size: 1.5rem; }
.app-header-title { font-size: 1.25rem; font-weight: 600; letter-spacing: -0.5px; }
.app-header-nav { display: flex; gap: 0.5rem; }
.nav-link { padding: 0.5rem 1rem; color: rgba(255, 255, 255, 0.9); text-decoration: none; border-radius: 4px; transition: background-color 0.2s; }
.nav-link:hover { background-color: rgba(255, 255, 255, 0.1); }
```

```
[

.nav-link.active { background-color: rgba(255, 255, 255, 0.2); }

[

.app-content { flex: 1; background: #f5f5f5; }

[

.popout-content { padding: 0; }

[
]
}) export class AppComponent implements OnInit, OnDestroy { isPopOut = false; private
destroy$ = new Subject<void>();



constructor(private router: Router) {}



ngOnInit(): void { // Check if current route is a pop-out this.isPopOut =
this.router.url.startsWith('/popout');



// Watch for route changes this.router.events.pipe( filter((event): event is
NavigationEnd => event instanceof NavigationEnd),
takeUntil(this.destroy$) ).subscribe(event => { this.isPopOut =
event.urlAfterRedirects.startsWith('/popout'); }); }



ngOnDestroy(): void { this.destroy$.next(); this.destroy$.complete(); } }
```

Key features:

Feature	Purpose
`*ngIf="!isPopOut"`	Hides header in pop-out windows
`routerLinkActive="active"`	Highlights current route in navigation
`[class.popout-content]`	Removes padding for pop-out content
Route change detection	Updates `isPopOut` on navigation

Step 905.5: Router Configuration Options

The `RouterModule.forRoot()` options we used:

```
RouterModule.forRoot(routes, {

anchorScrolling: 'enabled', scrollPositionRestoration: 'enabled',
paramsInheritanceStrategy: 'always' })
```

Option	Purpose
`anchorScrolling`	Enables scrolling to `#fragment` anchors
`scrollPositionRestoration`	Scrolls to top when navigating between routes
`paramsInheritanceStrategy: 'always'`	Query params available in all child routes

The `paramsInheritanceStrategy` is particularly important for URL-First architecture — it ensures query parameters are accessible throughout the component tree.

Verification

1. Build the Application

```
$ ng build
```

Expected: Build succeeds with no errors.

2. Serve and Navigate

```
$ ng serve --open
```

Test these navigation paths:

Action	Expected Result
Open `http://localhost:4200`	Redirects to `/home`
Click "Automobiles" in header	Navigates to `/automobiles`
Click "Advanced Search" card	Navigates to `/automobiles/discover`
Click "Home" in header	Returns to `/home`
Enter unknown URL like `/foo`	Redirects to `/home`

3. Verify Lazy Loading

Open browser DevTools (Network tab) and observe:

- Initial load: Only main bundle and home module
- Navigate to `/automobiles`: Automobile module loads
- Navigate to `/automobiles/discover`: No additional load (same module)

4. Check Router Link Active

The navigation link for the current route should have the `active` class applied, showing a slightly different background color.

5. Verify Query Param Persistence

- Navigate to `/automobiles/discover?manufacturer=Toyota`
- Note the query parameter in URL
- Refresh the page
- Query parameter should persist

Common Problems

Symptom	Cause	Solution
"Cannot find module" error	Import path wrong	Verify feature module paths are correct
Blank page on navigation	Component not declared in module	Add component to module's declarations
Query params lost on navigation	Wrong `paramsInheritanceStrategy`	Set to `'always'` in RouterModule.forRoot
Pop-out routes not loading	Missing route parameters	Verify URL includes gridId and componentId
"No provider" errors	Services not provided	Add providers to feature module or app module
Redirect loops	Conflicting route definitions	Check for multiple matching paths

---

Key Takeaways

- **Lazy loading improves performance** — Feature modules load only when needed
- **Route parameters enable dynamic content** — The pop-out route uses params to determine what to show
- **RouterModule.forRoot() vs forChild()** — `forRoot()` at app level, `forChild()` in feature modules

---

Acceptance Criteria

- [ ] `AppRoutingModule` configures all application routes
- [ ] Root path (`/`) redirects to `/home`
- [ ] Home, Automobile, and Popout routes are lazy loaded
- [ ] Unknown routes (`**`) redirect to home
- [ ] Feature modules define their internal routes with `RouterModule.forChild()`
- [ ] `AppComponent` hides header for pop-out routes
- [ ] `routerLinkActive` highlights current navigation item

- [ ] Query parameters persist across navigation
- [ ] `ng build` completes without errors
- [ ] Navigation between all routes works correctly

Architecture Note: Route Hierarchy

The complete route structure:

```
/
├── home                          → HomeComponent ├── automobiles                  →
AutomobileComponent │   └── discover                 → AutomobileDiscoverComponent
└── popout/:gridId/:componentId ├── query-control            → QueryControlComponent
├── picker                    → BasePickerComponent ├── chart                    →
BaseChartComponent ├── basic-results            → DynamicResultsTableComponent └──
statistics-2             → StatisticsPanel2Component
```

This hierarchy reflects the application's information architecture:

- Top level: Application areas (home, automobiles, popout)
- Second level: Features within areas (landing, discover)
- Pop-out level: Individual components in isolation

Next Step

Proceed to `906-app-module.md` to configure the root AppModule with all necessary imports, providers, and bootstrap configuration.

906: App Module

Status: Planning **Depends On:** 905-app-routing-module, 312-error-notification-service, 313-http-error-interceptor, 314-global-error-handler, 608-domain-providers **Blocks:** 907-final-integration

---

Learning Objectives

After completing this section, you will:

- Understand how to configure the root Angular module with all necessary providers
- Know how to set up HTTP interceptors for global error handling
- Be able to register domain configuration providers at the application level

---

Objective

Configure the root `AppModule` — the central hub that bootstraps the application and registers all global services, providers, and configurations. This module ties together everything we've built across all phases.

Why

The `AppModule` is the entry point for your Angular application. It:

- **Bootstraps the Application** — Tells Angular which component to load first
- **Registers Global Providers** — Services available throughout the app
- **Configures HTTP Layer** — Sets up interceptors for error handling
- **Imports Core Modules** — BrowserModule, HttpClient, etc.
- **Registers Domain Configuration** — Makes domain config available via DI

Angular Module Types

Type	Purpose	Example
Root Module	Bootstraps app, global config	`AppModule`
Feature Module	Encapsulates a feature	`AutomobileModule`
Shared Module	Commonly used components/pipes	`SharedModule`
Core Module	Singleton services	`CoreModule`

For vvroom, we use a simplified structure:

- `AppModule` handles root and core concerns
- Feature modules are lazy-loaded
- Framework services are provided at root level via `@Injectable({ providedIn: 'root' })`

Angular Style Guide References

- Style 04-12: Do not export the root module
- Style 04-13: Use the root module for configuration

What

Step 906.1: Create the Complete AppModule

Replace the contents of `src/app/app.module.ts`:

```
// src/app/app.module.ts

// VERSION 1 (Section 906) - Complete application module configuration



import { ErrorHandler, Injector, NgModule } from '@angular/core'; import
{ BrowserModule } from '@angular/platform-browser'; import { BrowserAnimationsModule }
from '@angular/platform-browser/animations'; import { HttpClientModule,
HTTP_INTERCEPTORS } from '@angular/common/http';



import { AppComponent } from './app.component'; import { AppRoutingModule } from './
app-routing.module';



// Framework services for global configuration import { HttpErrorInterceptor } from
'./framework/services/http-error.interceptor'; import { GlobalErrorHandler } from './
framework/services/global-error.handler'; import { DOMAIN_CONFIG } from './framework/
services/domain-config-registry.service';



// Domain configuration import { createAutomobileDomainConfig } from './domain-config/
automobile';



/**



  • Root Application Module

 *

  • Central configuration hub for the vvroom application.

 *

  • Responsibilities:



  • - Bootstrap the AppComponent
```

```
  • - Configure HTTP client with error interceptor

  • - Set up global error handling

  • - Provide domain configuration for dependency injection
 *
  • Module Import Order:

  • 1. BrowserModule (must be first for browser apps)

  • 2. BrowserAnimationsModule (enables Angular animations)

  • 3. HttpClientModule (enables HTTP requests)

  • 4. AppRoutingModule (application routes)
 *
  • Note: Feature modules (Home, Automobile, Popout) are lazy-loaded

  • via the router, not imported here.
 */
@NgModule({ declarations: [ AppComponent ], imports: [ // Angular core modules
BrowserModule, BrowserAnimationsModule, HttpClientModule,
```

```
// Application routing AppRoutingModule ], providers: [ // HTTP Error Interceptor - catches all HTTP errors { provide: HTTP_INTERCEPTORS, useClass: HttpErrorInterceptor, multi: true },



// Global Error Handler - catches all unhandled errors { provide: ErrorHandler, useClass: GlobalErrorHandler },



// Domain Configuration - provides automobile config to all components { provide: DOMAIN_CONFIG, useFactory: createAutomobileDomainConfig, deps: [Injector] } ], bootstrap: [AppComponent] }) export class AppModule {}
```

Module configuration explained:

Section	Purpose
`declarations`	Components owned by this module (only AppComponent)
`imports`	Other modules this module depends on
`providers`	Services and tokens available globally
`bootstrap`	The root component to start the application

Step 906.2: Understanding the Provider Configuration

Let's examine each provider in detail:

HTTP Interceptor

```
{
provide: HTTP_INTERCEPTORS, useClass: HttpErrorInterceptor, multi: true }
```

- `HTTP_INTERCEPTORS` is an Angular token for HTTP middleware

- `useClass` instantiates `HttpErrorInterceptor`
- `multi: true` allows multiple interceptors (they chain together)

The interceptor catches HTTP errors before they reach components, enabling centralized error handling.

Global Error Handler

```
{
provide: ErrorHandler, useClass: GlobalErrorHandler }
```

- `ErrorHandler` is Angular's built-in error handling token
- Replacing it with `GlobalErrorHandler` catches all unhandled errors
- Useful for logging, error reporting, and user notification

Domain Configuration

```
{
provide: DOMAIN_CONFIG, useFactory: createAutomobileDomainConfig, deps: [Injector] }
```

- `DOMAIN_CONFIG` is our custom injection token (from Phase 3)
- `useFactory` calls a function to create the config
- `deps: [Injector]` provides the Injector to the factory
- Components can inject `DOMAIN_CONFIG` to access domain configuration

Step 906.3: Domain Configuration Factory

The factory function creates the automobile domain configuration. This should be defined in `src/app/domain-config/automobile/index.ts`:

```
// src/app/domain-config/automobile/index.ts

// VERSION 1 (Section 906) - Domain config factory



import { Injector } from '@angular/core'; import { DomainConfig } from '../../
framework/models/domain-config.interface'; import { automobileDomainConfig } from './
automobile.domain-config';



/**



  • Factory function for creating automobile domain configuration


 *

  • This factory is called by Angular's dependency injection system



  • when DOMAIN_CONFIG is first requested.


 *

  • @param injector - Angular's Injector for creating service instances



  • @returns Complete domain configuration for automobiles


 */

export function createAutomobileDomainConfig(injector: Injector): DomainConfig<any,
any, any> { // The base config is defined in automobile.domain-config.ts // The
factory allows us to inject dependencies if needed return automobileDomainConfig; }



// Re-export everything from the domain config export * from './automobile.domain-
config'; export * from './models'; export * from './adapters'; export * from './
configs'; export * from './chart-sources';
```

Step 906.4: Main Entry Point

Verify the `main.ts` file bootstraps the AppModule:

```
// src/main.ts

// VERSION 1 (Section 906) - Application bootstrap



import { platformBrowserDynamic } from '@angular/platform-browser-dynamic'; import
{ AppModule } from './app/app.module';



platformBrowserDynamic() .bootstrapModule(AppModule) .catch(err =>
console.error('Bootstrap error:', err));
```

This is the entry point that Angular CLI uses to start the application.

Step 906.5: Module Import Order

The order of imports in `AppModule` matters:

```
imports: [

BrowserModule,           // 1. Browser platform (must be first)
BrowserAnimationsModule, // 2. Animation support HttpClientModule,        // 3. HTTP
client AppRoutingModule         // 4. Routes (last, so it can use all imported
modules) ]
```

Why this order?

- `BrowserModule` provides browser-specific services and must be imported first in the root module
- `BrowserAnimationsModule` must come after `BrowserModule`
- `HttpClientModule` sets up the HTTP client that services will use
- `AppRoutingModule` should be last so routes can reference any components from other imports

Step 906.6: Production Considerations

For production, you may want additional configuration. Create/update `src/environments/environment.prod.ts`:

```
// src/environments/environment.prod.ts

// VERSION 1 (Section 906) - Production environment



export const environment = { production: true, apiBaseUrl: 'http://generic-
prime.minilab/api/specs/v1' };
```

And in `main.ts`, you can enable production mode:

```
// src/main.ts

// VERSION 2 (Section 906) - Production mode support



import { enableProdMode } from '@angular/core'; import { platformBrowserDynamic } from
'@angular/platform-browser-dynamic'; import { AppModule } from './app/app.module';
import { environment } from './environments/environment';



if (environment.production) { enableProdMode(); }



platformBrowserDynamic() .bootstrapModule(AppModule) .catch(err =>
console.error('Bootstrap error:', err));
```

Verification

1. Build the Application

```
$ ng build
```

Expected: Build succeeds with no errors.

2. Serve the Application

```
$ ng serve
```

Expected: Application starts without errors. Console should show:

```
✔ Compiled successfully.
```

3. Check Provider Registration

Open browser DevTools and verify:

- **HTTP Interceptor:** Make a request and check Network tab. Errors should be handled gracefully.
- **Global Error Handler:** In console, run:

```
  throw new Error('Test error');
```

The error should be caught by `GlobalErrorHandler`.

- **Domain Config:** Components should receive domain configuration. If the discover page loads and shows data, the config is working.

4. Verify Lazy Loading

In the Network tab, observe chunk files loading:

- Initial load: `main.js`, `vendor.js`, `polyfills.js`
- Navigate to `/automobiles`: `automobile-module.js` loads

- Navigate to `/automobiles/discover`: No new chunk (same module)

5. Full Application Test

Complete this workflow:

- Open `http://localhost:4200` → Redirects to `/home`
- Click "Automobiles" → Navigates to automobile landing
- Click "Advanced Search" → Navigates to discover page
- Apply filters → Data loads, URL updates
- Refresh page → Filters persist
- Pop out a panel → New window opens with synced state

Common Problems

Symptom	Cause	Solution
"No provider for HttpClient"	HttpClientModule not imported	Add `HttpClientModule` to imports
"NullInjectorError: No provider for DOMAIN_CONFIG"	Factory not registered	Add DOMAIN_CONFIG provider with factory
Animations not working	BrowserAnimationsModule missing	Add `BrowserAnimationsModule` to imports
Multiple instances of services	Wrong providedIn setting	Use `providedIn: 'root'` for singletons
HTTP interceptor not firing	Not registered properly	Use `multi: true` with HTTP_INTERCEPTORS
"AppComponent is not known element"	Not in declarations	Add AppComponent to declarations array

---

Key Takeaways

- **AppModule is the configuration hub** — It wires together all parts of the application
- **Provider order doesn't matter, import order does** — Especially for BrowserModule
- **Use factories for complex configuration** — The DOMAIN_CONFIG factory pattern is powerful

---

Acceptance Criteria

- [ ] `AppModule` imports BrowserModule, BrowserAnimationsModule, HttpClientModule, and AppRoutingModule
- [ ] `AppComponent` is declared and bootstrapped
- [ ] `HTTP_INTERCEPTORS` provider registers `HttpErrorInterceptor`
- [ ] `ErrorHandler` provider registers `GlobalErrorHandler`
- [ ] `DOMAIN_CONFIG` provider uses factory function with Injector
- [ ] `main.ts` bootstraps `AppModule`
- [ ] Application builds without errors
- [ ] Application runs and all routes are accessible
- [ ] HTTP errors are caught by the interceptor
- [ ] Domain configuration is available throughout the app

Architecture Note: Provider Scopes

Angular's dependency injection has different scopes:

Scope	How to Achieve	Lifetime
Application (singleton)	`@Injectable({ providedIn: 'root' })`	Entire app lifetime
Module	`providers: []` in module	Module lifetime
Component	`providers: []` in component	Component lifetime

In vvroom:

- **Root-level singletons:** `UrlStateService`, `DomainConfigRegistry`, `ErrorNotificationService`

- **Module-provided:** None (all framework services are root)
- **Component-provided:** `ResourceManagementService`, `PopOutManagerService` (new instance per component)

This design ensures:

- Shared services (URL state) are consistent across the app
- Per-page services (resource management) are isolated to their component

---

Module Relationship Diagram

```
AppModule (root)
│ ├── BrowserModule ├── BrowserAnimationsModule ├── HttpClientModule ├──
AppRoutingModule ──────────────────┐ │                                    │ │   Lazy-
loaded via routes:           │ │   ┌────────────────────────────────────┘ │   │ │   ├──
HomeModule │   │   └── HomeComponent │   │ │   ├── AutomobileModule │   │   ├──
AutomobileComponent │   │   ├── AutomobileDiscoverComponent │   │   └── Framework
Component Modules │   │ │   └── PopoutModule │       └── PopoutComponent │ └──
Providers: ├── HTTP_INTERCEPTORS → HttpErrorInterceptor ├── ErrorHandler →
GlobalErrorHandler └── DOMAIN_CONFIG → createAutomobileDomainConfig()
```

Next Step

Proceed to `907-final-integration.md` to complete the integration, perform final testing, and celebrate completing the vvroom application!

907: Final Integration

Status: Planning **Depends On:** 901-906 (All Phase 9 components) **Blocks:** None (Final document)

Learning Objectives

After completing this section, you will:

- Understand how all the pieces of the vvroom application fit together
- Be able to perform comprehensive integration testing
- Have a fully functional production-ready Angular application

---

Objective

Complete the final integration of the vvroom application by verifying all components work together, performing comprehensive testing, and ensuring the application is ready for production use. This is the culmination of everything you've built throughout this book.

Congratulations!

You've made it. From an empty Angular project to a fully-featured data exploration application, you've typed every line of code and understood every decision. Let's take a moment to appreciate what you've built.

What You've Accomplished

Over the course of this book, you have:

Phase 0: API Contract & Naming

- Defined the API contract that drives the entire application

- Established naming conventions separating framework from domain code

Phase 1: Foundation

- Created the project structure from scratch
- Configured routing and environment settings
- Built the application shell

Phase 2: Framework Models

- Defined TypeScript interfaces for type safety
- Created generic models that work with any domain
- Learned how interfaces serve as executable documentation

Phase 3: Framework Services

- Built the URL-First state management system
- Created services for API communication, error handling, and pop-out windows
- Implemented the service layer that powers the entire application

Phase 4-6: Automobile Domain

- Defined automobile-specific models, adapters, and configurations
- Created the bridge between framework and domain
- Configured filters, tables, and charts for automobile data

Phase 7: Chart Data Sources

- Implemented visualization data transformations
- Created reusable chart source patterns

Phase 8: Framework Components

- Built reusable UI components for queries, charts, tables, and pickers
- Created components that work with any domain configuration
- Implemented the visual layer of the application

Phase 9: Feature Components

- Orchestrated framework components into complete pages
- Built the Home, Automobile Landing, and Discover pages
- Implemented pop-out window support for multi-monitor workflows
- Configured routing and the root module

The Aha Moment

"I just built a production application. I understand every line."

This is what sets you apart. You didn't just follow a tutorial — you understand:

- Why the URL is the single source of truth
- How dependency injection wires everything together
- What makes a component reusable vs. domain-specific
- When to use BehaviorSubject vs. ReplaySubject
- How to debug RxJS streams
- Why TypeScript generics enable type-safe reuse

What

Step 907.1: Complete File Structure Verification

Let's verify the complete file structure of the application:

```
$ cd ~/projects/vvroom

$ find src/app -type f -name "*.ts" | wc -l
```

You should have approximately 60-70 TypeScript files across:

```
src/app/

├── app.component.ts ├── app.module.ts ├── app-routing.module.ts ├── domain-config/
│   └── automobile/ │       ├── adapters/          (3 files) │       ├── chart-
sources/     (4 files) │       ├── configs/           (6 files) │       ├──
models/            (3 files) │       └── automobile.domain-config.ts ├── features/ │
├── home/                  (3 files) │   ├── automobile/ │   │   ├── automobile-
discover/  (4 files) │   │   └── (4 files) │   └── popout/               (4 files)
└── framework/ ├── components/           (~27 files across 9 components) ├──
models/                (8 files) ├── services/              (12 files) └──
tokens/                (1 file)
```

Step 907.2: Build Verification

Perform a production build to verify everything compiles correctly:

```
$ ng build --configuration production
```

Expected output:

```
✔ Browser application bundle generation complete.

✔ Copying assets complete. ✔ Index html generation complete.



Initial Chunk Files           | Names         |  Raw Size | Estimated Transfer Size
main.js                       | main          | 250.00 kB |              65.00 kB
polyfills.js                  | polyfills     | 180.00 kB |              57.00 kB
styles.css                    | styles        |  15.00 kB |               4.00 kB
runtime.js                    | runtime       |   2.00 kB |               1.00 kB



Lazy Chunk Files              | Names         |  Raw Size | Estimated Transfer Size
automobile-module.js          | automobile    |  80.00 kB |              20.00 kB
home-module.js                | home          |   5.00 kB |               2.00 kB
popout-module.js              | popout        |  10.00 kB |               3.00 kB



Build at: 2026-02-09T17:00:00.000Z - Hash: abc123def456 - Time: 15000ms
```

Key points:

- No errors or warnings
- Lazy chunks are generated for feature modules
- Bundle sizes are reasonable

Step 907.3: Integration Test Checklist

Run through this comprehensive test checklist:

Navigation Tests

Test	Expected Result	Pass?
Open `/`	Redirects to `/home`	[ ]
Click "Automobiles"	Navigates to `/automobiles`	[ ]
Click "Advanced Search"	Navigates to `/automobiles/discover`	[ ]
Click "Home" in header	Returns to `/home`	[ ]
Enter invalid URL `/xyz`	Redirects to `/home`	[ ]
Back button	Previous page loads	[ ]
Forward button	Next page loads	[ ]

URL State Tests

Test	Expected Result	Pass?
Apply filter on Discover	URL updates with query param	[ ]
Refresh page with filter	Filter persists	[ ]
Copy URL to new tab	Same state loads	[ ]
Clear all filters	URL resets to clean state	[ ]
Apply multiple filters	All params in URL	[ ]

Data Loading Tests

Test	Expected Result	Pass?
Load Discover page	Results count appears	[ ]
Apply manufacturer filter	Filtered results load	[ ]
Change pagination	Page param updates, new data loads	[ ]
Sort table column	Sort param updates, data re-orders	[ ]
Loading indicator	Shows during API calls	[ ]

Chart Interaction Tests

Test	Expected Result	Pass?
Charts display data	Bars/segments visible	[ ]
Hover on chart element	Tooltip appears	[ ]
Click chart bar (filter mode)	Filter applied, data updates	[ ]
Click chart bar (highlight mode)	Highlight added, visual change	[ ]
Multiple chart sources	All charts update with filters	[ ]

Pop-out Tests

Test	Expected Result	Pass?
Click pop-out button	New window opens	[ ]
Pop-out shows data	State synced from main	[ ]
Change filter in main	Pop-out updates	[ ]
Close pop-out	Panel reappears in main	[ ]
Pop-out header hidden	No navigation in pop-out	[ ]

Query Control Tests

Test	Expected Result	Pass?
Dropdown filter works	Options load, selection applies	[ ]
Text search works	Search executes on enter	[ ]
Clear individual filter	Only that filter removed	[ ]
Clear all filters	All filters removed	[ ]
Filter chips display	Active filters shown	[ ]

Error Handling Tests

Test	Expected Result	Pass?
API error occurs	Error notification shown	[ ]
Network timeout	Graceful error message	[ ]
Invalid filter value	Validation feedback	[ ]
404 route	Redirects to home	[ ]

Step 907.4: Performance Verification

Check application performance:

```
# Development server
$ ng serve


In another terminal, run Lighthouse or similar tool


Or manually check DevTools Performance tab
```

Performance targets:

- First Contentful Paint: < 1.5s
- Time to Interactive: < 3s
- Largest Contentful Paint: < 2.5s
- No layout shifts during load

Step 907.5: Code Quality Check

Run linting and formatting:

```
$ ng lint
```

Expected: No errors. (Warnings acceptable for rough draft.)

Step 907.6: Final Review of Key Patterns

Let's summarize the key architectural patterns used throughout the application:

URL-First State Management

```
URL (source of truth)

↓ UrlStateService (URL → Params) ↓ ResourceManagementService (Params → Filters → API)
↓ Components (via async pipe)
```

Every piece of state is derived from the URL. This enables:

- Deep linking
- Browser history support
- State sharing via URL
- Debuggability

Dependency Injection Hierarchy

```
AppModule (global providers)

↓ Feature Module (lazy-loaded) ↓ Component (component-level providers) ↓ Template (via
async pipe)
```

Services are provided at the appropriate level:

- Singletons at root
- Per-page instances at component level

Component Composition

```
Feature Component (orchestrator)
├── Framework Component (reusable) │   └── Domain Config (configuration) └── Framework Component (reusable) └── Domain Config (configuration)
```

Framework components are reusable shells. Domain configuration provides the specifics.

Verification

Final Build and Serve

```
$ ng build --configuration production
$ npx http-server dist/vvroom -p 8080
```

Open `http://localhost:8080` and verify:

- Application loads without errors
- All pages are accessible
- Data loads correctly
- Pop-outs work
- URL state persists

Commit the Final State

```
$ git add .
$ git commit -m "Section 907: Complete vvroom application" $ git tag phase-9-complete
```

What You've Built

Application Statistics

Metric	Value
TypeScript Files	~70
Lines of Code	~11,000
Components	~15
Services	~12
Interfaces	~10
Feature Modules	3

Capabilities

- **Data Exploration:** Filter, sort, and paginate through automobile data
- **Visualization:** Charts for manufacturers, years, body classes
- **Multi-Monitor:** Pop-out any panel to a separate window
- **URL Persistence:** Every view has a shareable URL
- **Error Handling:** Graceful error messages and recovery
- **Performance:** Lazy loading, OnPush change detection

Technologies Used

Technology	Purpose
Angular 13	Application framework
TypeScript	Type-safe JavaScript
RxJS	Reactive state management
Angular Router	Navigation
HttpClient	API communication
BroadcastChannel	Pop-out communication

Key Takeaways

- **Architecture matters** — The URL-First pattern made state management simple and predictable
- **Generics enable reuse** — Framework components work with any domain configuration
- **Separation of concerns** — Framework, domain, and feature code have clear boundaries
- **TypeScript is your friend** — Type safety caught errors before runtime
- **RxJS is powerful** — Reactive streams simplified complex async operations

---

Acceptance Criteria

- [ ] Production build completes without errors
- [ ] All navigation paths work correctly
- [ ] URL state persistence works (refresh preserves state)
- [ ] Data loads and displays correctly
- [ ] Charts are interactive
- [ ] Pop-out windows sync with main window
- [ ] Error handling shows appropriate messages
- [ ] Performance is acceptable (no jank, reasonable load times)
- [ ] Code passes linting

Where to Go from Here

You've built a complete application, but there's always more to learn:

Immediate Extensions

- Add more filter types (date ranges, multi-select)
- Implement data export (CSV, Excel)
- Add user preferences persistence
- Create custom chart types

Advanced Topics

- Unit testing with Jest
- E2E testing with Playwright
- State management with NgRx
- Server-side rendering with Angular Universal
- Progressive Web App (PWA) features

Architecture Evolution

- Micro-frontends
- Module federation
- GraphQL integration
- Real-time updates with WebSockets

The Companion Book Opportunity

Because you followed strict naming conventions and separated framework from domain code, expanding vvroom to support additional domains is straightforward. A companion book could cover:

- "Adding Agriculture to Vvroom" (just domain configuration)
- "Adding Chemistry to Vvroom" (just domain configuration)

The framework code you wrote is truly reusable.

Final Words

Congratulations. You've done something remarkable.

You didn't just copy code from a tutorial. You didn't rely on magic you don't understand. You built a production-quality Angular application from the ground up, and you know why every line is there.

This knowledge will serve you well. Whether you're building your own applications, contributing to enterprise projects, or teaching others, you now have a deep understanding of Angular architecture that most developers never achieve.

The URL-First pattern, the generic component architecture, the service layer design — these patterns will apply to projects far beyond vvroom.

Thank you for taking this journey. Now go build something amazing.

The Complete Journey

```
Phase 0: API Contract          ──→  "The API defines everything"

| Phase 1: Foundation           ──→  "Routes are the skeleton" | Interlude A:
Generics          ──→  "Type safety without duplication" | Phase 2: Framework
Models      ──→  "Interfaces are documentation" | Interlude B: RxJS             ──→
"Observables model change" | Phase 3: Framework Services    ──→  "The URL is the
source of truth" | Phase 4: Domain Models         ──→  "Models shaped by the API" |
Phase 5: Domain Adapters       ──→  "Adapters isolate change" | Phase 6: Domain
Configs        ──→  "Configuration is code" | Phase 7: Chart Sources         ──→
"Transform data for visualization" | Phase 8: Framework Components  ──→  "Generic +
specific = reusable" | Phase 9: Feature Components    ──→  "I understand every line"
```

Welcome to the club of Angular architects.

951: RxJS Operator Reference

Status: Reference Document **Type:** Appendix

Learning Objectives

After reading this reference, you will:

- Have a quick reference for all RxJS operators used in vvroom
- Understand when to use each operator
- Be able to look up operator behavior without leaving the book

---

Overview

This appendix provides a quick reference for every RxJS operator used in the vvroom application. Operators are grouped by category for easy lookup.

Creation Operators

`of()`

Creates an Observable that emits the provided values synchronously.

```
import { of } from 'rxjs';

of('a', 'b', 'c').subscribe(x => console.log(x)); // Output: 'a', 'b', 'c' (then
completes)
```

Used in vvroom: Error handling fallbacks, mock data

`Subject`

A Subject is both an Observable and an Observer. You can push values into it and subscribe to it.

```
import { Subject } from 'rxjs';

const subject = new Subject<string>();



subject.subscribe(x => console.log(x)); subject.next('hello'); // Output: 'hello'
```

Used in vvroom: `destroy$` for cleanup, event buses

`BehaviorSubject`

A Subject that requires an initial value and emits the current value to new subscribers.

```
import { BehaviorSubject } from 'rxjs';

const subject = new BehaviorSubject<number>(0);



subject.subscribe(x => console.log('A:', x)); // Output: 'A: 0'
subject.next(1);                                  // Output: 'A: 1' subject.subscribe(x
=> console.log('B:', x)); // Output: 'B: 1'
```

Used in vvroom: `stateSubject` in ResourceManagementService, `paramsSubject` in UrlStateService

---

`ReplaySubject`

A Subject that replays a specified number of previous values to new subscribers.

```
import { ReplaySubject } from 'rxjs';

const subject = new ReplaySubject<number>(2); // Buffer last 2 values


subject.next(1); subject.next(2); subject.next(3);


subject.subscribe(x => console.log(x)); // Output: 2, 3
```

Used in vvroom: PopOutContextService for message replay

`timer()`

Creates an Observable that emits after a delay.

```
import { timer } from 'rxjs';

timer(1000).subscribe(() => console.log('1 second passed'));
```

Used in vvroom: Request debouncing

`throwError()`

Creates an Observable that immediately emits an error.

```
import { throwError } from 'rxjs';

throwError(() => new Error('Something went wrong')) .subscribe({ error: err =>
console.error(err.message) });
```

Used in vvroom: Error propagation in services

Transformation Operators

`map()`

Transforms each emitted value using a projection function.

```
import { of } from 'rxjs';

import { map } from 'rxjs/operators';



of(1, 2, 3).pipe( map(x => x * 10) ).subscribe(x => console.log(x)); // Output: 10,
20, 30
```

Used in vvroom: Transforming API responses, extracting state properties

---

`switchMap()`

Maps to an inner Observable and cancels the previous inner Observable on each emission.

```
import { of, interval } from 'rxjs';

import { switchMap, take } from 'rxjs/operators';



of('A', 'B').pipe( switchMap(letter => interval(100).pipe( take(3), map(i => letter +
i) )) ).subscribe(x => console.log(x)); // Only 'B' values emit (A is cancelled) //
Output: B0, B1, B2
```

Used in vvroom: API calls that cancel previous requests when filters change

When to use: When only the latest request matters (search, autocomplete)

`mergeMap()`

Maps to an inner Observable and runs all inner Observables concurrently.

```
import { of } from 'rxjs';
import { mergeMap, delay } from 'rxjs/operators';

of(1, 2, 3).pipe( mergeMap(x => of(x).pipe(delay(100))) ).subscribe(x =>
console.log(x)); // Output: 1, 2, 3 (all run in parallel)
```

When to use: When all requests should complete (batch operations)

`concatMap()`

Maps to an inner Observable and waits for each to complete before starting the next.

```
import { of } from 'rxjs';
import { concatMap, delay } from 'rxjs/operators';

of(1, 2, 3).pipe( concatMap(x => of(x).pipe(delay(100))) ).subscribe(x =>
console.log(x)); // Output: 1 (wait) 2 (wait) 3 (sequential)
```

When to use: When order matters and each request must complete first

---

Filtering Operators

`filter()`

Emits only values that pass the predicate function.

```
import { of } from 'rxjs';

import { filter } from 'rxjs/operators';



of(1, 2, 3, 4, 5).pipe( filter(x => x % 2 === 0) ).subscribe(x => console.log(x)); //
Output: 2, 4
```

Used in vvroom: Filtering router events to NavigationEnd

`distinctUntilChanged()`

Emits only when the current value differs from the previous.

```
import { of } from 'rxjs';

import { distinctUntilChanged } from 'rxjs/operators';



of(1, 1, 2, 2, 3, 1).pipe( distinctUntilChanged() ).subscribe(x => console.log(x)); //
Output: 1, 2, 3, 1
```

With comparator:

```
of({ id: 1 }, { id: 1 }, { id: 2 }).pipe(

distinctUntilChanged((a, b) => a.id === b.id) ).subscribe(x => console.log(x)); //
Output: { id: 1 }, { id: 2 }
```

Used in vvroom: Preventing duplicate API calls when state hasn't changed

---

`take()`

Emits only the first N values then completes.

```
import { interval } from 'rxjs';
import { take } from 'rxjs/operators';

interval(100).pipe( take(3) ).subscribe(x => console.log(x)); // Output: 0, 1, 2 (then
completes)
```

Used in vvroom: Taking single values from streams

`takeUntil()`

Emits values until a notifier Observable emits.

```
import { interval, Subject } from 'rxjs';
import { takeUntil } from 'rxjs/operators';

const stop$ = new Subject<void>();

interval(100).pipe( takeUntil(stop$) ).subscribe(x => console.log(x));

setTimeout(() => stop$.next(), 350); // Output: 0, 1, 2 (then stops)
```

Used in vvroom: Component cleanup pattern with `destroy$`

`debounceTime()`

Emits a value only after a specified time has passed without new emissions.

```
import { Subject } from 'rxjs';
import { debounceTime } from 'rxjs/operators';

const search$ = new Subject<string>();

search$.pipe( debounceTime(300) ).subscribe(x => console.log('Search:', x));

search$.next('h'); search$.next('he'); search$.next('hel'); search$.next('help'); //
Output: 'Search: help' (after 300ms pause)
```

Used in vvroom: Search input debouncing

Error Handling Operators

`catchError()`

Catches errors on the Observable and returns a new Observable or throws.

```
import { of, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';

throwError(() => new Error('Oops!')).pipe( catchError(err =>
{ console.error(err.message); return of('fallback value'); }) ).subscribe(x =>
console.log(x)); // Output: 'Oops!' (error), 'fallback value'
```

Used in vvroom: API error handling, graceful degradation

`retry()`

Retries the source Observable a specified number of times on error.

```
import { interval, throwError } from 'rxjs';
import { mergeMap, retry } from 'rxjs/operators';

let attempts = 0;

interval(100).pipe( mergeMap(() => { attempts++; if (attempts < 3) { return
throwError(() => new Error('Retry')); } return of('success'); }),
retry(2) ).subscribe(x => console.log(x));
```

Used in vvroom: HTTP request retry in interceptor

Utility Operators

`tap()`

Performs side effects without modifying the stream.

```
import { of } from 'rxjs';
import { tap, map } from 'rxjs/operators';

of(1, 2, 3).pipe( tap(x => console.log('Before:', x)), map(x => x * 10), tap(x =>
console.log('After:', x)) ).subscribe(); // Output: Before: 1, After: 10, Before: 2,
After: 20, ...
```

Used in vvroom: Logging, debugging, caching side effects

`finalize()`

Performs an action when the Observable completes or errors.

```
import { of, throwError } from 'rxjs';
import { finalize } from 'rxjs/operators';

of(1, 2, 3).pipe( finalize(() => console.log('Done!')) ).subscribe(x =>
console.log(x)); // Output: 1, 2, 3, 'Done!'
```

Used in vvroom: Resetting loading state after API calls

`delay()`

Delays emission by a specified time.

```
import { of } from 'rxjs';
import { delay } from 'rxjs/operators';

of('delayed').pipe( delay(1000) ).subscribe(x => console.log(x)); // Output:
'delayed' (after 1 second)
```

Used in vvroom: Testing, simulated network latency

`timeout()`

Errors if no value is emitted within a specified time.

```
import { of, NEVER } from 'rxjs';
import { timeout, catchError } from 'rxjs/operators';

NEVER.pipe( timeout(1000), catchError(err => of('Timeout!')) ).subscribe(x =>
console.log(x)); // Output: 'Timeout!' (after 1 second)
```

Used in vvroom: API request timeouts

Combination Operators

`combineLatest()`

Combines multiple Observables and emits when any source emits (after all have emitted at least once).

```
import { combineLatest, of, interval } from 'rxjs';
import { take, map } from 'rxjs/operators';

combineLatest([ of('A'), interval(100).pipe(take(3)) ]).subscribe(([letter, num]) =>
console.log(letter, num)); // Output: A 0, A 1, A 2
```

Used in vvroom: Combining filters from multiple sources

`merge()`

Merges multiple Observables into a single stream.

```
import { merge, interval } from 'rxjs';
import { take, map } from 'rxjs/operators';

merge( interval(100).pipe(take(2), map(x => 'A' + x)), interval(150).pipe(take(2),
map(x => 'B' + x)) ).subscribe(x => console.log(x)); // Output: A0, B0, A1, B1
(interleaved by timing)
```

Used in vvroom: Merging event streams

`forkJoin()`

Waits for all Observables to complete, then emits an array of the last values.

```
import { forkJoin, of } from 'rxjs';
import { delay } from 'rxjs/operators';

forkJoin([ of('A').pipe(delay(100)), of('B').pipe(delay(200)),
of('C').pipe(delay(300)) ]).subscribe(x => console.log(x)); // Output: ['A', 'B', 'C']
(after 300ms)
```

Used in vvroom: Parallel API requests that must all complete

Quick Reference Table

Operator	Category	Purpose	Cancels Previous?
`of`	Creation	Emit static values	N/A
`Subject`	Creation	Observable + Observer	N/A
`BehaviorSubject`	Creation	Subject with initial value	N/A
`map`	Transform	Transform values	No
`switchMap`	Transform	Map + cancel previous	Yes
`mergeMap`	Transform	Map + run concurrent	No
`concatMap`	Transform	Map + run sequential	No
`filter`	Filter	Emit matching values	No
`distinctUntilChanged`	Filter	Skip duplicates	No
`take`	Filter	Emit first N	No
`takeUntil`	Filter	Emit until notifier	No
`debounceTime`	Filter	Delay until pause	No
`catchError`	Error	Handle errors	No
`retry`	Error	Retry on error	No
`tap`	Utility	Side effects	No
`finalize`	Utility	Cleanup on complete	No
`combineLatest`	Combine	Combine latest values	No
`merge`	Combine	Merge streams	No
`forkJoin`	Combine	Wait for all	No

Key Takeaways

- **Use `switchMap` for search/filter operations** — Cancels outdated requests

- **Use `BehaviorSubject` for state** — Always has a current value
- **Use `takeUntil(destroy$)` for cleanup** — Prevents memory leaks
- **Use `distinctUntilChanged` to prevent duplicate work** — Especially with API calls
- **Use `catchError` to handle errors gracefully** — Don't let streams die unexpectedly

---

Further Reading

- RxJS Official Documentation
- Learn RxJS
- RxJS Marbles — Interactive marble diagrams

952: TypeScript Generics Reference

Status: Reference Document **Type:** Appendix

Learning Objectives

After reading this reference, you will:

- Have a quick reference for generic patterns used in vvroom
- Understand advanced generic techniques beyond the primer
- Be able to look up type utilities without leaving the book

Overview

This appendix extends the TypeScript Generics Primer (document 150) with advanced patterns and a complete reference of generic types used in the vvroom application.

Quick Reference: Generic Syntax

Declaration Syntax

Syntax	Meaning
`<T>`	Single type parameter
`<T, U>`	Multiple type parameters
`<T extends Base>`	Constrained type parameter
`<T = Default>`	Type parameter with default
`<T extends Base = Default>`	Constrained with default

Usage Syntax

Syntax	Meaning
`Array<string>`	Concrete instantiation
`Partial<User>`	Utility type application
`keyof T`	Get keys of T as union
`T[K]`	Indexed access type
`T extends U ? X : Y`	Conditional type

Vvroom Generic Patterns

The Domain Config Pattern

The central generic pattern in vvroom:

```
interface DomainConfig<TFilters, TData, TStatistics> {

domainKey: string; displayName: string; urlMapper: IFilterUrlMapper<TFilters>;
apiAdapter: IApiAdapter<TFilters, TData, TStatistics>; tableConfig:
TableConfig<TData>; chartDataSources?: Record<string, ChartDataSource<TStatistics>>; }
```

Type flow:

```
DomainConfig<AutomobileFilters, VehicleResult, VehicleStatistics>

|               |               |               | |               |
|               └─► chartDataSources value types |               |
└─► apiAdapter, tableConfig |               └─► apiAdapter, urlMapper └─► The whole
thing
```

The Adapter Pattern

Adapters connect framework code to domain-specific implementations:

```
interface IApiAdapter<TFilters, TData, TStatistics> {

fetchData(filters: TFilters): Observable<ApiResponse<TData>>; fetchStatistics(filters:
TFilters): Observable<TStatistics>; }



// Concrete implementation class AutomobileApiAdapter implements IApiAdapter<
AutomobileFilters, VehicleResult, VehicleStatistics > { fetchData(filters:
AutomobileFilters): Observable<ApiResponse<VehicleResult>> { // Implementation } }
```

The Service Generic Pattern

Services that work with any domain use type parameters:

```
class ResourceManagementService<TFilters, TData, TStatistics> {

private readonly state$ = new BehaviorSubject<ResourceState<TFilters, TData,
TStatistics>>( initialState );



get data$(): Observable<TData[] | undefined> { return this.state$.pipe(map(s =>
s.data)); } }
```

Built-in Utility Types

Partial<T>

Makes all properties optional:

```
interface User {

id: number; name: string; email: string; }



// All properties optional type PartialUser = Partial<User>; // { id?: number; name?:
string; email?: string; }



// Use case: Update operations function updateUser(id: number, updates: Partial<User>)
{ // Only changed fields are required }



updateUser(1, { name: 'New Name' }); // OK - only name provided
```

Used in vvroom: Filter updates, partial state changes

Required<T>

Makes all properties required (opposite of Partial):

```
interface Config {

debug?: boolean; timeout?: number; }



type RequiredConfig = Required<Config>; // { debug: boolean; timeout: number; }
```

Readonly<T>

Makes all properties readonly:

```
interface State {
count: number; items: string[]; }



type ReadonlyState = Readonly<State>; // { readonly count: number; readonly items:
string[]; }



const state: ReadonlyState = { count: 0, items: [] }; state.count = 1; // Error:
Cannot assign to 'count'
```

Used in vvroom: Immutable state patterns

Pick<T, K>

Creates a type with only the specified properties:

```
interface Vehicle {
vin: string; make: string; model: string; year: number; color: string; }



type VehicleIdentity = Pick<Vehicle, 'vin' | 'make' | 'model'>; // { vin: string;
make: string; model: string; }
```

Used in vvroom: Extracting subsets for display or API calls

Omit<T, K>

Creates a type without the specified properties:

```
interface Vehicle {

vin: string; make: string; model: string; year: number; internalId: string; // Don't
send to API }



type VehicleForApi = Omit<Vehicle, 'internalId'>; // { vin: string; make: string;
model: string; year: number; }
```

Record<K, V>

Creates an object type with keys of type K and values of type V:

```
type ChartSourceMap = Record<string, ChartDataSource>;

// { [key: string]: ChartDataSource }



const sources: ChartSourceMap = { 'manufacturer': new ManufacturerChartSource(),
'year': new YearChartSource() };
```

Used in vvroom: `chartDataSources`, URL parameters

Extract<T, U>

Extracts types from T that are assignable to U:

```
type EventTypes = 'click' | 'focus' | 'blur' | 'scroll' | 'resize';

type MouseEvents = Extract<EventTypes, 'click' | 'scroll'>; // 'click' | 'scroll'
```

Exclude<T, U>

Excludes types from T that are assignable to U:

```
type EventTypes = 'click' | 'focus' | 'blur' | 'scroll' | 'resize';

type KeyboardEvents = Exclude<EventTypes, 'click' | 'scroll' | 'resize'>; // 'focus' | 'blur'
```

NonNullable<T>

Removes null and undefined from T:

```
type MaybeString = string | null | undefined;

type DefinitelyString = NonNullable<MaybeString>; // string
```

ReturnType<T>

Gets the return type of a function type:

```
function createFilters(): AutomobileFilters {

return new AutomobileFilters(); }



type Filters = ReturnType<typeof createFilters>; // AutomobileFilters
```

Parameters<T>

Gets the parameter types of a function as a tuple:

```
function fetchData(filters: AutomobileFilters, page: number): Promise<VehicleResult[]>
{
// ... }


type FetchParams = Parameters<typeof fetchData>; // [AutomobileFilters, number]
```

Advanced Patterns

Mapped Types

Transform properties of a type:

```
// Make all properties nullable
type Nullable<T> = { [P in keyof T]: T[P] | null; };


interface User { name: string; age: number; }


type NullableUser = Nullable<User>; // { name: string | null; age: number | null; }
```

Conditional Types

Types that depend on conditions:

```
type IsArray<T> = T extends any[] ? true : false;

type A = IsArray<string[]>;  // true type B = IsArray<string>;    // false
```

Practical example:

```
// Unwrap array types, leave others alone
type Unwrap<T> = T extends (infer U)[] ? U : T;


type A = Unwrap<string[]>;  // string type B = Unwrap<number>;    // number
```

Template Literal Types

Create types from string patterns:

```
type FilterKey = 'manufacturer' | 'year' | 'model';
type HighlightKey = h_${FilterKey}; // 'h_manufacturer' | 'h_year' | 'h_model'


// Used in URL state management function isHighlightParam(key: string): key is
HighlightKey { return key.startsWith('h_'); }
```

Type Guards with Generics

User-Defined Type Guards

Narrow types safely:

```
interface ApiError {

code: number; message: string; }



interface ApiSuccess<T> { data: T; }



type ApiResponse<T> = ApiError | ApiSuccess<T>;



function isSuccess<T>(response: ApiResponse<T>): response is ApiSuccess<T> { return
'data' in response; }



// Usage function handleResponse<T>(response: ApiResponse<T>): T | null { if
(isSuccess(response)) { return response.data;  // TypeScript knows this is
ApiSuccess<T> } console.error(response.message);  // TypeScript knows this is ApiError
return null; }
```

Generic Constraints in Vvroom

Constraining to Interfaces

```
interface HasId {

id: string; }



function findById<T extends HasId>(items: T[], id: string): T | undefined { return
items.find(item => item.id === id); }



// Works with any type that has an 'id' property const vehicle = findById(vehicles,
'VIN123'); const user = findById(users, 'USER456');
```

Constraining to Keys

```
function getProperty<T, K extends keyof T>(obj: T, key: K): T[K] {

return obj[key]; }



const vehicle = { make: 'Toyota', year: 2022 }; const make = getProperty(vehicle,
'make');  // string const year = getProperty(vehicle, 'year');  // number const bad =
getProperty(vehicle, 'color');  // Error: 'color' not in keyof vehicle
```

Observable Generic Patterns

RxJS uses generics extensively:

```
// Observable of specific type

statistics$: Observable<VehicleStatistics>;



// Subject with type parameter private readonly state$ = new
BehaviorSubject<ResourceState<TFilters, TData>>(initial);



// Operators preserve/transform types this.state$.pipe( map(state =>
state.data),      // Observable<TData[] | undefined> filter((data): data is TData[] =>
data !== undefined),  // Observable<TData[]> map(data => data.length)       //
Observable<number> );
```

Common Mistakes

Mistake 1: Missing Type Arguments

```
// Wrong - Map needs type arguments
const cache = new Map();  // Map<any, any>


// Right const cache = new Map<string, VehicleResult>();
```

Mistake 2: Overly Narrow Constraints

```
// Too restrictive - only works with AutomobileFilters
function process<T extends AutomobileFilters>(filters: T) { }


// Better - works with any filter type function process<T>(filters: T) { }


// Or if you need specific properties function process<T extends { page?: number }
>(filters: T) { }
```

Mistake 3: Ignoring Inference

```
// Unnecessary - TypeScript infers T
const result = identity<string>('hello');


// Let TypeScript infer const result = identity('hello');  // T is inferred as string
```

Mistake 4: Using `any` Instead of Generics

```
// Bad - loses type safety
function firstElement(arr: any[]): any { return arr[0]; }


// Good - preserves type function firstElement<T>(arr: T[]): T | undefined { return
arr[0]; }
```

Quick Reference Table: Vvroom Generic Types

Type	Purpose	Type Parameters
`DomainConfig<F, D, S>`	Domain configuration	Filters, Data, Statistics
`IApiAdapter<F, D, S>`	API adapter interface	Filters, Data, Statistics
`IFilterUrlMapper<F>`	URL mapping interface	Filters
`TableConfig<D>`	Table configuration	Data row type
`ChartDataSource<S>`	Chart data transformation	Statistics type
`ResourceState<F, D, S>`	Service state	Filters, Data, Statistics
`ApiResponse<D>`	API response wrapper	Data type
`Observable<T>`	RxJS observable	Emitted value type
`BehaviorSubject<T>`	RxJS subject with initial value	Value type

Key Takeaways

- **Generics flow through the architecture** — From DomainConfig through services to components
- **Utility types reduce boilerplate** — Use Partial, Pick, Omit instead of redefining
- **Constraints ensure type safety** — Use `extends` to require specific properties

- **Let TypeScript infer when possible** — Don't over-annotate

---

Further Reading

- TypeScript Handbook: Generics
- TypeScript Handbook: Utility Types
- TypeScript Handbook: Conditional Types

953: Debugging Guide

Status: Reference Document **Type:** Appendix

Learning Objectives

After reading this guide, you will:

- Know how to diagnose common Angular and RxJS issues
- Understand how to use browser DevTools effectively
- Be able to debug URL-First state management problems

Overview

This appendix provides debugging strategies for the vvroom application. It covers Angular-specific debugging, RxJS observable debugging, and URL-First architecture troubleshooting.

Browser DevTools Essentials

Opening DevTools

Browser	Shortcut
Chrome/Edge	`F12` or `Ctrl+Shift+I` (Windows) / `Cmd+Option+I` (Mac)
Firefox	`F12` or `Ctrl+Shift+I` (Windows) / `Cmd+Option+I` (Mac)
Safari	`Cmd+Option+I` (enable in Preferences → Advanced first)

Essential Panels

Panel	Use For
Console	Error messages, `console.log` output
Network	API requests, response data, timing
Elements	DOM inspection, CSS debugging
Sources	Breakpoints, step debugging
Application	localStorage, sessionStorage, cookies

Console Debugging

Strategic Console Logging

```
// Bad - vague
console.log(filters);


// Good - contextual console.log('[UrlStateService] Parsed filters:', filters);


// Better - with timestamp for async flows console.log( [${Date.now()}] [ResourceService] Fetching data for: , filters);
```

Console Methods

Method	Use Case
`console.log()`	General output
`console.error()`	Errors (red in console)
`console.warn()`	Warnings (yellow in console)
`console.table()`	Display arrays/objects as table
`console.group()` / `console.groupEnd()`	Group related logs
`console.time()` / `console.timeEnd()`	Measure execution time

Debugging Objects

```
// See full object structure
console.dir(complexObject);



// Table view for arrays console.table(vehicles);



// Group related logs console.group('Filter Processing'); console.log('Input:',
rawParams); console.log('Parsed:', parsedFilters); console.log('Validated:',
validatedFilters); console.groupEnd();
```

Angular-Specific Debugging

Angular DevTools Extension

Install the Angular DevTools browser extension for:

- Component tree inspection
- Change detection profiling
- Dependency injection debugging

Common Angular Errors

"Can't bind to 'X' since it isn't a known property"

Cause: Component/directive not imported in the module.

Solution:

```
// Check the module imports

@NgModule({ imports: [ CommonModule,      // For ngIf,
ngFor FormsModule,       // For [(ngModel)] RouterModule,      // For routerLink
SomeComponent      // For standalone components ] })
```

"No provider for X"

Cause: Service not provided in module or component.

Solution:

```
// Option 1: providedIn root (preferred)

@Injectable({ providedIn: 'root' }) export class MyService { }



// Option 2: Provide in module @NgModule({ providers: [MyService] })



// Option 3: Provide in component @Component({ providers: [MyService] })
```

"ExpressionChangedAfterItHasBeenCheckedError"

Cause: Value changed during change detection cycle.

Solution:

```
// Wrap in setTimeout or use ChangeDetectorRef

constructor(private cdr: ChangeDetectorRef) {}



ngAfterViewInit(): void { // Defer the change setTimeout(() => { this.value =
newValue; this.cdr.markForCheck(); }, 0); }
```

Template Binding Errors

Debug with JSON pipe:

```
<!-- See what the value actually is -->

<pre>{{ someValue | json }}</pre>



<!-- Check if observable has emitted --> <pre>{{ observable$ | async | json }}</pre>
```

RxJS Debugging

The tap() Operator

Insert `tap()` to inspect stream values without modifying them:

```
this.urlState.params$.pipe(

tap(params => console.log('[1] Raw params:', params)), map(params =>
this.parseFilters(params)), tap(filters => console.log('[2] Parsed filters:',
filters)), switchMap(filters => this.apiService.fetch(filters)), tap(response =>
console.log('[3] API response:', response)) ).subscribe();
```

Debugging Subscription Issues

"Observable not emitting"

Check these in order:

- **Is it subscribed?**

```
// Cold observables don't run without subscription
observable$.subscribe();  // or use | async in template
```

- **Has the source emitted?**

```
source$.pipe(
tap(v => console.log('Source emitted:', v)) ).subscribe();
```

- **Is it filtered out?**

```
source$.pipe(
tap(v => console.log('Before filter:', v)), filter(v => v.isValid), tap(v =>
console.log('After filter:', v)) ).subscribe();
```

- **Is switchMap cancelling it?**

```
// switchMap cancels previous inner observables
// Use mergeMap if you need all to complete
```

"Observable emitting too many times"

```
// Add distinctUntilChanged to prevent duplicates

source$.pipe( tap(v => console.log('Before distinct:', v)), distinctUntilChanged(),
tap(v => console.log('After distinct:', v)) ).subscribe();
```

Memory Leaks

Check for missing unsubscribe:

```
// Bad - leaks on component destroy

ngOnInit() { this.service.data$.subscribe(data => this.data = data); }



// Good - uses takeUntil pattern private destroy$ = new Subject<void>();



ngOnInit() { this.service.data$.pipe( takeUntil(this.destroy$) ).subscribe(data =>
this.data = data); }



ngOnDestroy() { this.destroy$.next(); this.destroy$.complete(); }
```

URL-First Debugging

Verifying URL State

- **Check the URL directly:**
- Look at the browser address bar
- Compare expected vs actual query parameters
- **Log URL changes:**

```
// In UrlStateService or component

this.router.events.pipe( filter(event => event instanceof NavigationEnd), tap(event =>
console.log('[Router] Navigation:', event.url)) ).subscribe();
```

- **Log parsed parameters:**

```
this.route.queryParams.pipe(

tap(params => console.log('[Route] Query params:', params)) ).subscribe();
```

Common URL-First Issues

Filters not updating URL

Check the URL mapper:

```
console.log('[UrlMapper] toParams input:', filters);

const params = this.urlMapper.toParams(filters); console.log('[UrlMapper] toParams
output:', params);
```

URL not updating view

Check the filter parsing:

```
console.log('[UrlMapper] fromParams input:', params);

const filters = this.urlMapper.fromParams(params); console.log('[UrlMapper] fromParams
output:', filters);
```

Circular updates (infinite loop)

Add distinctUntilChanged with deep comparison:

```
this.route.queryParams.pipe(

distinctUntilChanged((a, b) => JSON.stringify(a) === JSON.stringify(b)), tap(params =>
console.log('Params changed:', params)) ).subscribe();
```

Network Debugging

Inspecting API Calls

In DevTools Network panel:

- Filter by "XHR" or "Fetch" to see only API calls
- Click a request to see:
- **Headers**: Request/response headers
- **Payload**: Request body (POST/PUT)
- **Preview**: Response data (formatted)
- **Response**: Raw response
- **Timing**: How long each phase took

Common API Issues

CORS Errors

```
Access to XMLHttpRequest at 'http://api.example.com' from origin 'http://localhost:
4200'

has been blocked by CORS policy
```

Solution: API server must include CORS headers, or use Angular proxy:

```
// proxy.conf.json

{ "/api": { "target": "http://api.example.com", "secure": false, "changeOrigin": true } }
```

401 Unauthorized

Check:

- Is the auth token being sent?
- Is the token expired?
- Is the token in the correct header format?

```
// Log request headers in interceptor

intercept(req: HttpRequest<any>, next: HttpHandler) { console.log('[HTTP] Request headers:', req.headers.keys()); console.log('[HTTP] Auth header:', req.headers.get('Authorization')); return next.handle(req); }
```

404 Not Found

Check:

- Is the URL correct? (log it)
- Is the API endpoint deployed?
- Are path parameters correct?

Plotly.js Chart Debugging

Chart Not Rendering

- **Check container dimensions:**

```
const container = this.chartContainer.nativeElement;

console.log('Container size:', container.clientWidth, 'x', container.clientHeight); //
Must be > 0
```

- **Check data format:**

```
console.log('Traces:', JSON.stringify(chartData.traces, null, 2));

console.log('Layout:', JSON.stringify(chartData.layout, null, 2));
```

- **Check for Plotly errors:**

```
Plotly.newPlot(element, traces, layout)

.then(() => console.log('Plot created successfully')) .catch(err =>
console.error('Plotly error:', err));
```

Chart Not Updating

Ensure you're calling react, not newPlot:

```
// newPlot creates a new chart (slower)

// react updates existing chart (faster) if (this.plotlyElement)
{ Plotly.react(this.plotlyElement, traces, layout); } else { Plotly.newPlot(element,
traces, layout); }
```

Pop-out Window Debugging

Messages Not Received

- **Check message posting:**

```
// In parent window

console.log('[Parent] Sending message:', message); popoutWindow.postMessage(message,
'*');
```

- **Check message receiving:**

```
// In pop-out window

window.addEventListener('message', event => { console.log('[Popout] Received
message:', event.data); console.log('[Popout] Origin:', event.origin); });
```

- **Check window reference:**

```
console.log('[Parent] Popout window:', popoutWindow);

console.log('[Parent] Window closed?:', popoutWindow.closed);
```

Pop-out Styling Issues

The pop-out window may not load all styles:

- Check that global styles are linked in pop-out index.html
- Verify component styles are loaded (not lazy-loaded)
- Use browser DevTools in the pop-out window itself

Performance Debugging

Change Detection Issues

```
// Log change detection runs

constructor(private zone: NgZone) { zone.onStable.subscribe(() =>
{ console.log('[Zone] Change detection complete'); }); }
```

Identifying Slow Operations

```
console.time('expensiveOperation');

// ... operation console.timeEnd('expensiveOperation'); // Output: expensiveOperation:
234.56ms
```

Memory Leaks

- Open DevTools → Memory panel
- Take heap snapshot before and after suspected leak
- Compare snapshots to find retained objects

Debugging Checklist

When something doesn't work:

- **Check the console** for errors
- **Check the network** for failed requests
- **Add console.log** at key points
- **Verify data** with `| json` pipe
- **Check subscriptions** are active
- **Verify imports** in modules
- **Check the URL** for state issues
- **Use breakpoints** for complex logic

Key Takeaways

- **Console.log strategically** — Add context, use groups, remove when done
- **Use tap() for RxJS** — Inspect streams without modifying them
- **Check the Network panel** — API issues often hide there
- **Angular DevTools helps** — Especially for component tree and DI issues
- **URL-First means URL-First** — When in doubt, check the URL

Further Reading

- Angular Debugging Guide
- Chrome DevTools Documentation
- RxJS Debugging

954: Glossary

Status: Reference Document **Type:** Appendix

Overview

This glossary defines terms used throughout the vvroom book. Terms are organized alphabetically within categories.

Angular Terms

Change Detection

The process by which Angular checks component properties for changes and updates the DOM accordingly. Can be triggered manually via `ChangeDetectorRef.markForCheck()` or `detectChanges()`.

Component

A building block of Angular applications consisting of a TypeScript class, HTML template, and optional CSS styles. Decorated with `@Component()`.

Decorator

A TypeScript feature that adds metadata to classes, properties, or methods. Angular uses decorators like `@Component()`, `@Injectable()`, `@Input()`, and `@Output()`.

Dependency Injection (DI)

A design pattern where a class receives its dependencies from external sources rather than creating them. Angular's DI system provides services to components.

Directive

A class that modifies DOM elements or component behavior. Structural directives (`*ngIf`, `*ngFor`) change DOM structure; attribute directives modify appearance or behavior.

Guard

A service that controls route access. Can prevent navigation (`CanActivate`), leaving (`CanDeactivate`), or loading (`CanLoad`).

Interceptor

A service that intercepts HTTP requests and responses. Used for adding headers, handling errors, or transforming data globally.

Lifecycle Hook

Methods that Angular calls at specific points in a component's lifecycle: `ngOnInit`, `ngOnChanges`, `ngAfterViewInit`, `ngOnDestroy`, etc.

Module (NgModule)

A container for related components, directives, pipes, and services. Decorated with `@NgModule()`. Angular 13 uses NgModules (not standalone components).

Observable

An RxJS type representing a stream of values over time. Components subscribe to observables to receive data. See also: RxJS Terms.

Pipe

A function that transforms data in templates. Built-in examples: `| json`, `| async`, `| date`. Custom pipes transform domain-specific data.

Route

A URL path mapped to a component. Defined in routing modules with path, component, and optional guards.

Service

A class that provides functionality across components. Typically decorated with `@Injectable()` and provided at root level.

Template

The HTML portion of a component that defines its view. Can include Angular syntax like interpolation (`{{ }}`), property binding (`[prop]`), and event binding (`(event)`).

RxJS Terms

BehaviorSubject

A Subject that requires an initial value and emits the current value to new subscribers. Used for state that always has a value.

combineLatest

An operator that combines the latest values from multiple observables. Emits whenever any source emits (after all have emitted at least once).

distinctUntilChanged

An operator that only emits when the current value differs from the previous value. Prevents duplicate emissions.

filter

An operator that emits only values that pass a predicate function.

map

An operator that transforms each emitted value using a projection function.

Observable

A lazy push collection that can emit zero or more values over time. Must be subscribed to for values to flow.

Operator

A function that transforms an observable stream. Applied via the `pipe()` method. Examples: `map`, `filter`, `switchMap`.

pipe

A method that chains operators together. `source$.pipe(op1, op2, op3)` applies operators in sequence.

ReplaySubject

A Subject that replays a specified number of previous values to new subscribers. Useful for late subscribers.

shareReplay

An operator that multicasts an observable and replays the last N values to new subscribers. Prevents duplicate API calls.

Subject

An observable that is also an observer. Can push values with `next()` and be subscribed to.

subscribe

The method that activates an observable and receives its values. Returns a Subscription that should be cleaned up.

Subscription

An object representing the execution of an observable. Call `unsubscribe()` to stop receiving values and prevent memory leaks.

switchMap

An operator that maps to an inner observable and cancels the previous inner observable on each new emission. Ideal for search/filter operations.

takeUntil

An operator that emits values until a notifier observable emits. Used for component cleanup with a `destroy$` subject.

tap

An operator that performs side effects (like logging) without modifying the stream.

URL-First Architecture Terms

Adapter

A class that converts between framework types and domain-specific types. Examples: `AutomobileUrlMapper`, `AutomobileApiAdapter`.

API Adapter

An adapter that handles API communication for a specific domain. Implements `IApiAdapter<TFilters, TData, TStatistics>`.

Cache Key Builder

A class that generates unique cache keys from filter objects. Ensures API responses are cached correctly.

Chart Data Source

A class that transforms domain statistics into chart-ready data (Plotly traces). Each chart type has its own data source.

Domain

A specific area of functionality in the application. In vvroom, "automobile" is the domain. Other books might add "agriculture" or "real estate".

Domain Config

The central configuration object for a domain. Contains adapters, UI configs, and type references. Type: `DomainConfig<TFilters, TData, TStatistics>`.

Domain Config Registry

A service that stores and retrieves domain configurations. Allows framework code to access the active domain's config.

Filter

A set of criteria that narrow search results. In vvroom: manufacturer, model, year, body class, etc.

Framework Code

Domain-agnostic code that works with any domain through interfaces. Located in `src/app/framework/`.

Highlight

A visual emphasis on data that matches specific criteria. Highlights don't filter data; they color-code matching items.

Pop-out Window

A separate browser window displaying part of the application. Communicates with the main window via `postMessage`.

Resource Management Service

The central service managing data fetching, caching, and state. Coordinates URL state with API calls.

Single Source of Truth

A principle where one authoritative location stores each piece of state. In URL-First, the URL is the single source of truth for filter state.

URL Mapper

An adapter that converts between filter objects and URL query parameters. Bidirectional: `toParams()` and `fromParams()`.

URL State Service

A service that manages reading from and writing to URL query parameters. Provides reactive streams of URL state.

TypeScript Terms

Concrete Type

A specific, known type like `string`, `number`, or `AutomobileFilters`. Contrast with type parameters.

Constraint

A limit on what types can be used as a type parameter. Syntax: `<T extends SomeInterface>`.

Generic

A type or function that works with multiple types through type parameters. Provides type safety without code duplication.

Interface

A TypeScript structure defining a contract for objects. Does not exist at runtime; only provides compile-time type checking.

keyof

A TypeScript operator that returns a union of an object's property names as string literal types.

Partial<T>

A utility type that makes all properties of T optional.

Pick<T, K>

A utility type that creates a new type with only the specified properties from T.

Record<K, V>

A utility type representing an object with keys of type K and values of type V.

Type Guard

A function that narrows a type within a conditional block. Uses `is` return type syntax.

Type Parameter

A placeholder type in a generic definition, filled in when the generic is used. Conventionally named `T`, `U`, or descriptively like `TFilters`.

Union Type

A type that can be one of several types. Syntax: `string | number | null`.

Utility Type

Built-in TypeScript types that transform other types. Examples: `Partial`, `Required`, `Pick`, `Omit`, `Record`.

UI Component Terms

Base Chart Component

A reusable Plotly.js wrapper that works with any `ChartDataSource`. Handles rendering, resizing, and click events.

Base Picker Component

A reusable multi-select dropdown that works with any `PickerConfig`. Handles option loading and selection.

CDK (Component Dev Kit)

Angular's set of behavior primitives for building UI components. Used for drag-drop functionality.

Dockview

A third-party library providing tabbed, resizable, dockable panel layouts. Framework-agnostic.

PrimeNG

The UI component library used in vvroom. Provides styled components like tables, buttons, dialogs, etc.

Query Panel

A component displaying filter inputs for user-adjustable search criteria.

Results Table

A component displaying paginated search results in a table format.

Statistics Panel

A component displaying multiple charts showing data distributions.

API Terms

Base URL

The root URL for all API endpoints. Configured in `environment.ts`.

Endpoint

A specific API URL path that accepts requests and returns responses. Example: `/api/specs/v1/search`.

Pagination

The practice of returning data in pages rather than all at once. API returns `page`, `size`, `totalItems`, `totalPages`.

Query Parameter

A key-value pair in a URL after the `?`. Example: `?manufacturer=Toyota&year=2022`.

Request

An HTTP call to an API endpoint. Includes method (GET, POST), headers, and optional body.

Response

Data returned by an API. Includes status code, headers, and body (usually JSON).

Development Terms

Barrel Export

An `index.ts` file that re-exports symbols from a directory, simplifying imports.

Build

The process of compiling TypeScript and bundling the application for deployment. Command: `ng build`.

Hot Module Replacement (HMR)

A development feature that updates modules without full page reload.

Lazy Loading

Loading modules on demand rather than at startup. Improves initial load time.

Linting

Static code analysis to catch errors and enforce style. Tool: ESLint.

Serve

Running the application in development mode with live reload. Command: `ng serve`.

Tree Shaking

Removing unused code during the build process. Reduces bundle size.

Key Acronyms

Acronym	Meaning
API	Application Programming Interface
CDK	Component Dev Kit
CLI	Command Line Interface
CORS	Cross-Origin Resource Sharing
CSS	Cascading Style Sheets
DI	Dependency Injection
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
RxJS	Reactive Extensions for JavaScript
SCSS	Sassy CSS (CSS preprocessor)
SPA	Single Page Application
URL	Uniform Resource Locator
VIN	Vehicle Identification Number

Further Reading

For deeper understanding of these terms, refer to:

- Angular Glossary
- TypeScript Handbook
- RxJS Documentation
- MDN Web Docs

Appendix A01: Styling and Branding

Overview

This appendix documents the styling system and branding configuration for the vvroom application. The visual design follows the dark theme pattern established in generic-prime while applying vvroom-specific branding.

Theme Configuration

PrimeNG Theme

The application uses PrimeNG's **lara-dark-blue** theme as the base, providing:

- Dark background colors (#2a2a2a, #3c3c3c)
- Blue accent colors (#64B5F6, #64c8ff)
- Consistent component styling

Global Styles (styles.scss)

```
/ PrimeNG Theme and Component Styles /

@import "primeng/resources/themes/lara-dark-blue/theme.css"; @import "primeng/
resources/primeng.min.css"; @import "primeicons/primeicons.css";



/ KaTeX CSS for LaTeX rendering / @import "katex/dist/katex.min.css";



/ Global Styles /


  • {


  box-sizing: border-box;

}



body { margin: 0; font-family: var(--font-family); background-color: #3c3c3c; color:
#ffffff; }
```

Color Palette

Purpose	Color	Usage
Background (dark)	`#2a2a2a`	Header, panels
Background (main)	`#3c3c3c`	Page background
Text primary	`#ffffff`	Main text
Text secondary	`#888888`	Version labels, hints
Accent	`#64B5F6`	Interactive elements
Accent hover	`#64c8ff`	Hover states
Border	`#444444`	Dividers, separators

App Component Styling

Header Structure

```
.app-header {

background-color: #2a2a2a; padding: 0.5rem 1rem; display: flex; justify-content: flex-start; align-items: center; gap: 2rem; border-bottom: 1px solid #444; height: 60px; }
```

Navigation Links

```
.home-link,

.discover-link { color: #ffffff; text-decoration: none; font-size: 1.2rem; font-weight: bold; white-space: nowrap; cursor: pointer;



&:hover { color: #64c8ff; } }
```

Branding Configuration

Domain Label

The primary branding appears in the **domain configuration** (`domainLabel` property):

File: `src/app/domain-config/automobile/automobile.domain-config.ts`

```
return {

domainName: 'automobile', domainLabel: 'Vvroom Discovery',  // Branding displayed in
discover header apiBaseUrl: apiBaseUrl, // ... };
```

Application Title

File: `src/app/app.component.ts`

```
@Component({ ... })

export class AppComponent { title = 'vvroom'; // ... }
```

HTML Title

File: `src/index.html`

```
<title>Vvroom - Automobile Discovery</title>
```

Pop-Out Window Styling

Pop-out windows require special handling to hide scrollbars:

```
body.popout-body {
overflow: hidden; height: 100vh; width: 100vw; }


html.popout-html { overflow: hidden; height: 100vh; width: 100vw;


.app-content { overflow: hidden !important; height: 100vh !important; } }
```

Component Style Files

Each component has its own SCSS file for component-specific styles:

Component	Style File
App Shell	`app.component.scss`
Discover	`features/discover/discover.component.scss`
Home	`features/home/home.component.scss`
Query Control	`framework/components/query-control/query-control.component.scss`
Query Panel	`framework/components/query-panel/query-panel.component.scss`
Base Chart	`framework/components/base-chart/base-chart.component.scss`
Base Picker	`framework/components/base-picker/base-picker.component.scss`
Results Table	`framework/components/results-table/results-table.component.scss`
Statistics Panel	`framework/components/statistics-panel-2/statistics-panel-2.component.scss`

Implementation Checklist

- [ ] Import PrimeNG lara-dark-blue theme in styles.scss
- [ ] Set body background to #3c3c3c

- [ ] Configure header with #2a2a2a background
- [ ] Update domainLabel to "Vvroom Discovery"
- [ ] Set app component title to "vvroom"
- [ ] Update index.html title tag
- [ ] Apply hover colors (#64c8ff) to interactive elements

Reference

- **PrimeNG Themes**: https://primeng.org/theming
- **Source Reference**: generic-prime/frontend/src/styles.scss
- **Color System**: VS Code Dark Theme inspired

Appendix A02: URL-First Testing Rubric

Overview

This rubric provides a systematic method for verifying URL-First State Management compliance across popped-in and popped-out controls. The URL is the single source of truth; all state changes must flow through URL updates.

Test Categories

Category 1: Main Window (Popped-In) Control Changes

Test that changes in the main window controls are reflected in:

- The browser URL parameters
- All other controls in the main window
- Any open pop-out windows

Test ID	Test Description	Expected Behavior	Pass/ Fail
M1.1	Change a query control filter (e.g., manufacturer dropdown)	URL updates with filter param; results table updates; statistics update	
M1.2	Change a highlight filter (e.g., year range)	URL updates with `h_` prefixed param; highlighted rows change; pop-outs receive update	
M1.3	Change pagination (page number)	URL updates with `page` param; table shows correct page	
M1.4	Change page size	URL updates with `size` param; table row count matches	
M1.5	Change sort column	URL updates with `sort` param; table re-sorts	
M1.6	Change sort direction	URL updates with `sortDirection` param; table order reverses	
M1.7	Clear all filters	URL params removed; controls reset to defaults; full dataset shown	
M1.8	Apply multiple filters simultaneously	All filter params appear in URL; results reflect intersection	

Category 2: Pop-Out Window Control Changes

Test that changes in pop-out windows are communicated to the main window and reflected appropriately.

Test ID	Test Description	Expected Behavior	Pass/ Fail
P2.1	Change highlight filter in pop-out	Main window URL updates with `h_` param; main window highlights update	
P2.2	Pop-out sends filter change message	BroadcastChannel message received by main window	
P2.3	Pop-out does NOT update its own URL	Pop-out URL remains static (initial state only)	
P2.4	Pop-out does NOT make its own API calls	Network tab shows no API requests from pop-out after initial load	
P2.5	Pop-out receives state via BroadcastChannel	`syncStateFromExternal()` called; no API fetch triggered	
P2.6	Multiple pop-outs stay synchronized	Change in one pop-out reflects in main window and all other pop-outs	

Category 3: URL Paste Tests (Without Highlight Filters)

Test that pasting a URL with standard filters correctly restores application state.

Test ID	Test Description	Expected Behavior	Pass/ Fail
U3.1	Paste URL with single filter param	Filter control shows correct value; results match filter	
U3.2	Paste URL with multiple filter params	All filter controls populated; results show intersection	
U3.3	Paste URL with pagination params	Correct page displayed; pagination control shows correct page	
U3.4	Paste URL with sort params	Table sorted correctly; sort indicators match URL	
U3.5	Paste URL with all param types combined	All controls reflect URL state; results correct	
U3.6	Paste URL with invalid filter value	Graceful handling; invalid param ignored or defaulted	
U3.7	Share URL to another browser/ session	New session shows identical state to original	

Category 4: URL Paste Tests (With Highlight Filters)

Test that pasting a URL with highlight filters (`h_` prefix) correctly applies highlighting.

Test ID	Test Description	Expected Behavior	Pass/ Fail
H4.1	Paste URL with `h_yearMin` param	Year highlight filter populated; matching rows highlighted	
H4.2	Paste URL with `h_manufacturer` param	Manufacturer highlight populated; matching rows highlighted	
H4.3	Paste URL with multiple highlight params	All highlight filters populated; rows matching ALL highlighted	
H4.4	Paste URL mixing query and highlight params	Query filters filter data; highlight filters highlight within results	
H4.5	Paste URL with highlight param into pop-out	Pop-out shows initial highlights; syncs with main window	
H4.6	Clear highlight via URL (remove `h_` param)	Highlights removed; highlight controls cleared	

Category 5: Pop-Out Window Presentation

Test that pop-out windows display correctly without main window chrome.

Test ID	Test Description	Expected Behavior	Pass/ Fail
W5.1	Pop-out hides site banner/header	No navigation header visible in pop-out	
W5.2	Pop-out shows query control panel	Filter controls visible and functional	
W5.3	Pop-out URL contains `popout=true` param	URL includes pop-out indicator	
W5.4	Pop-out title reflects content	Window title indicates popped-out component	
W5.5	Pop-out respects `autoFetch = false`	No initial API call; waits for main window data	

Category 6: Cross-Window Synchronization

Test bidirectional communication between main window and pop-outs.

Test ID	Test Description	Expected Behavior	Pass/ Fail
S6.1	Main window filter change updates all pop-outs	All pop-outs receive BroadcastChannel message and update	
S6.2	Pop-out filter change updates main window URL	Main window URL reflects pop-out's requested change	
S6.3	Main window data refresh updates pop-outs	New API data propagated to all pop-outs	
S6.4	Close pop-out does not affect main window state	Main window continues functioning normally	
S6.5	Open multiple pop-outs of same type	Each pop-out shows consistent state	
S6.6	Open pop-outs of different types	Each receives relevant state updates	

Category 7: Router Navigate Encapsulation

Verify that `router.navigate()` is only called from `UrlStateService`.

Test ID	Test Description	Expected Behavior	Pass/ Fail
R7.1	Grep codebase for `router.navigate`	Only appears in `url-state.service.ts`	
R7.2	Components call `updateFilters()` method	Components never call `router.navigate()` directly	
R7.3	Pop-out components call parent messaging	No `router.navigate()` in pop-out components	

Anti-Pattern Checklist

These patterns indicate URL-First violations and should fail testing:

Anti-Pattern	How to Detect	Severity
Direct state mutation bypassing URL	State changes without URL param update	Critical
`router.navigate()` in components	Grep for `router.navigate` outside UrlStateService	Critical
Pop-out making API calls	Network tab shows fetch from pop-out window	Critical
Pop-out updating its own URL	Pop-out URL changes after initial load	Critical
State not in URL that should be shareable	Filter applied but not in URL; refresh loses state	High
Highlight state without `h_` prefix	Highlight params using wrong naming convention	Medium

---

Test Execution Checklist

Before running tests:

- [ ] Development server running on port 4207
- [ ] Browser DevTools Network tab open (to verify API calls)
- [ ] Browser DevTools Console open (to catch errors)
- [ ] At least one pop-out window open for cross-window tests

After each test:

- [ ] Verify URL params match expected state
- [ ] Verify all controls reflect URL state
- [ ] Verify pop-out windows synchronized (if applicable)
- [ ] Check console for errors

Known Issues (Observed)

The following issues were observed during initial inspection and should be addressed:

- **Pop-out URL incorrect** - Pop-out window URL does not reflect expected state
- **Pop-out shows site banner** - Header/navigation visible in pop-out (should be hidden)
- **Query control not visible in pop-out** - Filter panel missing from pop-out view

---

References

- instructions.md - URL-First Compliance Checklist
- A01-styling-and-branding.md - Theme configuration
- `~/projects/vroom/docs/STATE-MANAGEMENT-SPECIFICATION.md` - Complete specification
- `~/projects/vroom/docs/POPOUT-ARCHITECTURE.md` - Cross-window communication