YukiSwapi

Johnny Driesen

2025-08-03

YukiSwapi - Tech Assessment for Yuki

VUE.JS 3.4+ QUASAR 2.16+ TYPESCRIPT 5.5+ PINIA 3.0+

A modern, responsive web application for exploring the Star Wars universe

Features • Installation • Architecture • Development

Table of Contents

Overview

YukiSwapi is a comprehensive Star Wars universe explorer built with modern web technologies. It provides an intuitive interface to browse and discover information about characters, planets, films, species, vehicles, and starships from the Star Wars saga using the SWAPI (Star Wars API).

Key Highlights

- **Modern Architecture**: Built with Vue 3 Composition API and TypeScript
- **Responsive Design**: Optimized for desktop, tablet, and mobile devices
- **Multi-language Support**: Available in English, Dutch, and Spanish
- **Dark Mode**: Toggle between light and dark themes
- **Real-time Search**: Instant search across all resource types
- **Caching Strategy**: Intelligent caching for improved performance

- **Type Safety**: Full TypeScript implementation for reliability

Features

Interactive Dashboard

- **Category Cards**: Visual navigation to different resource types
- **Hover Effects**: Smooth animations and transitions
- **Responsive Grid**: Adapts to different screen sizes

Data Tables

- **Paginated Lists**: Efficient browsing of large datasets
- **Search Functionality**: Real-time filtering across all fields
- **Sortable Columns**: Click to sort by any column
- **Loading States**: Visual feedback during data fetching

Detail Pages

- **Comprehensive Information**: Complete resource details
- **Related Resources**: Links to connected entities
- **Formatted Data**: Human-readable formatting for all fields
- **Navigation**: Easy back navigation and breadcrumbs

Internationalization

- **Multi-language**: English (en-US), Dutch (nl-NL), Spanish (es-ES)
- **Dynamic Switching**: Change language without page reload
- **Localized Formatting**: Numbers, dates, and units in local format

User Experience

- **Dark/Light Mode**: System preference detection with manual toggle
- **Responsive Design**: Mobile-first approach with breakpoint optimization
- **Accessibility**: ARIA labels and keyboard navigation support
- **Error Handling**: Graceful error states with retry options

Tech Stack

Core Technologies

Technology	Version	Purpose
Vue.js	3.4+	Progressive JavaScript framework
Quasar Framework	2.16+	Vue.js based UI framework
TypeScript	5.5+	Type-safe JavaScript development
Pinia	3.0+	State management for Vue
Vue Router	4.0+	Client-side routing
Vue i18n	11.0+	Internationalization
Axios	1.2+	HTTP client for API requests

Development Tools

Tool	Version	Purpose
Vite	Latest	Fast build tool and dev server
Vitest	3.2+	Unit testing framework
Cypress	14.5+	End-to-end testing
ESLint	9.14+	Code linting and quality
Prettier	3.3+	Code formatting
SCSS	Latest	CSS preprocessing

Additional Libraries

- **@quasar/extras**: Material Icons and Roboto font
- **@intlify/unplugin-vue-i18n**: i18n build optimization
- **MSW**: Mock Service Worker for testing
- **start-server-and-test**: E2E testing automation

Installation & Setup

Prerequisites

Ensure you have the following installed:

- **Node.js**: Version 18, 20, 22, 24, 26, or 28
- **npm**: Version 6.13.4 or higher
- **yarn**: Version 1.21.1 or higher (optional)

Quick Start

1. **Clone the repository**

   ```
   git clone <repository-url>
   cd yukiswapi
   ```

2. **Install dependencies**

   ```
   npm install
   # or
   yarn install
   ```

3. **Start development server**

   ```
   npm run dev
   # or
   quasar dev
   ```

4. **Open your browser** Navigate to `http://localhost:9000`

Environment Configuration

Create a `.env` file in the root directory:

```
# API Configuration
VITE_API_BASE_URL=https://swapi.dev/api

# Application Configuration
VITE_APP_VERSION=0.0.1
VITE_APP_NAME=YukiSwapi

# Development Configuration
VITE_DEV_MODE=true
```

Build for Production

```
# Build the application
npm run build
# or
quasar build

# Preview the build
npm run preview
```

Project Architecture

Directory Structure

```
yukiswapi/
  public/                  # Static assets
     icons/              # App icons and favicons
     favicon.ico
  src/
     assets/             # Static assets (images, fonts)
     boot/               # Quasar boot files
        axios.ts        # HTTP client configuration
        components.ts   # Global component registration
        dark-mode.ts    # Dark mode initialization
        i18n.ts         # Internationalization setup
     components/         # Reusable Vue components
        App/            # Application-level components
        Cards/          # Resource card components
        DetailPage/     # Detail page components
        Tables/         # Data table components
        TabPanels/      # Tab panel components
        Tabs/           # Tab navigation components
     composables/        # Vue composables
        useAppInfo.ts   # Application information
        useDetail.ts    # Detail page logic
        useI18n.ts      # Internationalization
        use*Table.ts    # Table-specific composables
     config/             # Configuration files
        resource-icons.ts  # Resource icons and colors
        */columns.ts   # Table column definitions
    css/               # Global styles
        app.scss       # Main application styles
        quasar.variables.scss  # Quasar variables
    i18n/              # Internationalization
        en-US/         # English translations
        nl-NL/         # Dutch translations
        es-ES/         # Spanish translations
    layouts/           # Layout components
        MainLayout.vue # Main application layout
    pages/             # Route-based page components
        IndexPage.vue  # Home page
        *Page.vue      # Resource list pages
        *DetailPage.vue # Resource detail pages
    router/            # Vue Router configuration
        index.ts       # Router setup
        routes.ts      # Route definitions
    services/          # API service layer
        base-swapi.service.ts  # Base service class
        *.service.ts   # Resource-specific services
    stores/            # Pinia state management
        base.store.ts  # Generic store factory
        *.store.ts     # Resource-specific stores
    types/             # TypeScript type definitions
        base.type.ts   # Base types
        *.type.ts      # Resource-specific types
        index.ts       # Type exports
    App.vue            # Root component
test/                  # Test files
    fixtures/          # Test data fixtures
    helpers/           # Test helper utilities
    unit/              # Unit tests
    vitest/            # Vitest configuration
cypress/               # E2E tests
    e2e/               # Test specifications
    fixtures/          # Test fixtures
    support/           # Support files
docs/                  # Documentation
```

Design Patterns

1. Generic Store Factory Pattern The application uses a generic store factory to eliminate code duplication across different resource types:

```
// Base store factory
export function createSwapiStore<T, R>(config: StoreConfig<T, R>) {
  return defineStore(config.storeName, {
    state: () => ({
      items: [],
      currentItem: null,
      isLoading: false,
      // ... other state
    }),
    actions: {
      async fetchItems(page: number, search?: string) {
        // Generic implementation
      },
    },
  })
}

// Usage in specific stores
export const usePlanetsStore = createSwapiStore({
  storeName: 'planets',
  resourceName: 'planets',
  fetchMany: PlanetsService.getAll,
  fetchOne: PlanetsService.getById,
  errorMessages: {
    /* ... */
  },
})
```

2. Service Layer Abstraction All API interactions go through a service layer with a base class:

```
export abstract class BaseSwapiService {
  protected static async fetchResources<T>(endpoint: string, page: number, search?: s
    // Generic API logic with error handling
  }
}

export class PlanetsService extends BaseSwapiService {
  static async getAll(page: number, search?: string) {
    return this.fetchResources<PlanetsResponse>('planets', page, search)
  }
}
```

3. Composable Pattern Vue composables encapsulate reusable logic:

```
export function useTable<T>(storeName: string) {
  const store = useStore(storeName)

  const onRequest = async (props: TableRequestProps) => {
    await store.onRequest(props)
  }

  return {
    tableData: computed(() => store.tableData),
    isLoading: computed(() => store.isLoading),
    onRequest,
  }
}
```

4. Configuration-Driven Development Resource configurations are centralized:

```
export const RESOURCE_CONFIGS = {
  planets: {
    icon: 'public',
    color: 'green',
    description: 'Explore worlds throughout the universe',
  },
  // ... other resources
}
```

API Integration

SWAPI (Star Wars API)

The application integrates with the Star Wars API (SWAPI) to fetch data about:

- **People**: Characters from the Star Wars universe
- **Planets**: Worlds and locations
- **Films**: Movies in the saga
- **Species**: Different life forms
- **Vehicles**: Ground and air transport
- **Starships**: Space vessels and cruisers

Base URL

```
https://swapi-api.hbtn.io/api/
```

Caching Strategy

The application implements intelligent caching:

- **TTL**: 10 minutes for all cached data
- **Storage**: In-memory Map-based caching
- **Invalidation**: Manual cache clearing available
- **Scope**: Both list and detail data cached separately

Error Handling

Comprehensive error handling covers:

- **Network Errors**: Connection issues
- **HTTP Errors**: 4xx and 5xx responses
- **Timeout Errors**: Request timeouts
- **Parsing Errors**: Invalid JSON responses

Rate Limiting

The application respects SWAPI rate limits:

- **Concurrent Requests**: Limited to prevent overwhelming the API
- **Retry Logic**: Exponential backoff for failed requests
- **User Feedback**: Loading states and error messages

Development Guide

Available Scripts

Script	Command	Description
Development	`npm run dev`	Start development server with hot reload
Build	`npm run build`	Build for production
Preview	`npm run preview`	Preview production build locally
Lint	`npm run lint`	Run ESLint on source files
Format	`npm run format`	Format code with Prettier
Type Check	`npm run type-check`	Run TypeScript compiler check

Testing Scripts

Script	Command	Description
Unit Tests	`npm run test`	Run unit tests in watch mode
Test Run	`npm run test:run`	Run unit tests once
Test UI	`npm run test:ui`	Open Vitest UI interface
Coverage	`npm run test:coverage`	Generate test coverage report
E2E Tests	`npm run e2e`	Run end-to-end tests
E2E Open	`npm run e2e:open`	Open Cypress test runner

Development Workflow

1. **Start Development Server**

   ```
   npm run dev
   ```

   - Hot module replacement enabled
   - TypeScript checking
   - ESLint integration
   - Automatic browser opening

2. **Code Quality**

   ```
   # Lint code
   npm run lint

   # Format code
   npm run format
   ```

```
# Type checking
npm run type-check
```

3. **Testing**

```
# Run unit tests
npm run test:watch

# Run E2E tests
npm run e2e:open
```

Code Style Guidelines

TypeScript

- Use strict TypeScript configuration
- Define interfaces for all data structures
- Prefer type inference where possible
- Use generic types for reusable components

Vue Components

- Use Composition API with `<script setup>`
- Define props with TypeScript interfaces
- Use composables for shared logic
- Follow single responsibility principle

Naming Conventions

- **Files**: kebab-case (e.g., `planet-card.vue`)
- **Components**: PascalCase (e.g., `PlanetCard`)
- **Variables**: camelCase (e.g., `planetData`)
- **Constants**: UPPER_SNAKE_CASE (e.g., `API_BASE_URL`)

Testing

Testing Strategy

The application uses a comprehensive testing approach:

Unit Testing (Vitest)

- **Framework**: Vitest with jsdom environment
- **Coverage**: 85% lines, 80% branches/functions
- **Focus**: Component logic, composables, services
- **Mocking**: MSW for API mocking

Example unit test:

```
describe('PlanetCard', () => {
  it('displays planet information correctly', () => {
    const wrapper = mountComponent(PlanetCard, {
      props: { planet: mockPlanet },
    })

    expect(wrapper.text()).toContain('Tatooine')
    expect(wrapper.text()).toContain('arid')
  })
})
```

E2E Testing (Cypress)

- **Framework**: Cypress with TypeScript
- **Coverage**: Critical user journeys
- **Environment**: Automated browser testing
- **CI/CD**: Integrated with build pipeline

Example E2E test:

```
describe('Navigation Flow', () => {
  it('should navigate through planets', () => {
    cy.visit('/')
    cy.get('[data-cy="planets-card"]').click()
    cy.url().should('include', '/planets')
    cy.get('[data-cy="planet-row"]').first().click()
    cy.url().should('match', /\/planets\/\d+/)
  })
})
```

Test Configuration

Vitest Configuration

```
export default defineConfig({
  test: {
    environment: 'jsdom',
    setupFiles: ['test/vitest/setup.ts'],
    coverage: {
      thresholds: {
        global: {
          branches: 80,
          functions: 80,
          lines: 85,
          statements: 85,
        },
      },
    },
  },
})
```

Cypress Configuration

```
export default defineConfig({
  e2e: {
    baseUrl: 'http://localhost:9000',
    viewportWidth: 1280,
    viewportHeight: 720,
    defaultCommandTimeout: 10000,
  },
})
```

Deployment

Build Process

1. **Production Build**

   ```
   npm run build
   ```

2. **Build Output**

   ```
   dist/
      index.html
      assets/
          *.js
          *.css
          *.woff2
      icons/
   ```

Deployment Options

Static Hosting (Recommended) Deploy to any static hosting service:

- **Netlify**: Drag and drop `dist` folder
- **Vercel**: Connect GitHub repository
- **GitHub Pages**: Use GitHub Actions
- **AWS S3**: Upload to S3 bucket with CloudFront

Docker Deployment

```
FROM nginx:alpine
COPY dist/ /usr/share/nginx/html/
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Environment Variables Set environment variables for different environments:

```
# Production
VITE_API_BASE_URL=https://swapi-api.hbtn.io/api
VITE_APP_ENV=production

# Staging
VITE_API_BASE_URL=https://swapi-api.hbtn.io/api/
VITE_APP_ENV=staging
```

Contributing

Development Setup

1. Fork the repository
2. Clone your fork
3. Install dependencies: `npm install`
4. Create a feature branch: `git checkout -b feature/amazing-feature`
5. Make your changes
6. Run tests: `npm run test`
7. Commit changes: `git commit -m 'Add amazing feature'`
8. Push to branch: `git push origin feature/amazing-feature`
9. Open a Pull Request

Code Review Process

1. All changes require pull request review
2. Automated tests must pass
3. Code coverage must meet thresholds
4. ESLint and Prettier checks must pass
5. TypeScript compilation must succeed

Commit Convention

Follow conventional commits:

```
feat: add new planet detail component
fix: resolve pagination issue in tables
docs: update API documentation
test: add unit tests for services
refactor: improve store factory pattern
```

Ideas for Possible Enhancements

Performance Optimizations

1. Virtual Scrolling

- **Problem**: Large datasets cause performance issues
- **Solution**: Implement virtual scrolling for tables
- **Impact**: Handle thousands of items smoothly
- **Implementation**: Use Quasar's QVirtualScroll component

2. Image Optimization

- **Problem**: No images currently displayed
- **Solution**: Add character/planet images with lazy loading
- **Impact**: Enhanced visual experience
- **Implementation**: WebP format with fallbacks

3. **Service Worker**
    - **Problem**: No offline functionality
    - **Solution**: Implement PWA with service worker
    - **Impact**: Offline browsing capability
    - **Implementation**: Workbox integration

Feature Enhancements

1. **Advanced Search**
    - **Current**: Basic text search
    - **Enhancement**: Filters, faceted search, advanced queries
    - **Features**:
        - Filter by multiple criteria
        - Date range filters
        - Numeric range filters
        - Saved search queries

2. **Favorites System**
    - **Feature**: Save favorite characters, planets, etc.
    - **Storage**: Local storage with sync option
    - **UI**: Heart icons, favorites page
    - **Export**: Share favorites via URL

3. **Comparison Tool**
    - **Feature**: Side-by-side comparison of resources
    - **UI**: Drag and drop interface
    - **Data**: Highlight differences
    - **Export**: Generate comparison reports

4. **Data Visualization**
    - **Charts**: Character heights, planet populations
    - **Graphs**: Relationship networks
    - **Maps**: Galaxy map with planet locations
    - **Timeline**: Film release timeline

Technical Improvements

1. **State Persistence**
    - **Problem**: State lost on page refresh
    - **Solution**: Persist store state to localStorage
    - **Benefits**: Better user experience
    - **Implementation**: Pinia persistence plugin

2. **Real-time Updates**
    - **Feature**: WebSocket connection for live data
    - **Use Case**: Collaborative features
    - **Technology**: Socket.io or native WebSockets
    - **Fallback**: Polling mechanism

3. **Micro-frontends**
    - **Architecture**: Split into smaller applications
    - **Benefits**: Independent deployment, team autonomy
    - **Technology**: Module federation
    - **Challenges**: Shared state management

4. **GraphQL Integration**
    - **Problem**: REST API limitations
    - **Solution**: GraphQL wrapper around SWAPI
    - **Benefits**: Flexible queries, reduced over-fetching
    - **Implementation**: Apollo Client integration

UI/UX Enhancements

1. **Advanced Theming**
    - **Current**: Light/dark mode
    - **Enhancement**: Multiple theme options
    - **Features**: Custom color schemes, font options
    - **Accessibility**: High contrast themes

2. **Mobile App**
    - **Technology**: Capacitor for native mobile apps
    - **Features**: Push notifications, offline sync
    - **Platforms**: iOS and Android
    - **Distribution**: App stores

3. **Accessibility Improvements**
    - **Screen Reader**: Enhanced ARIA labels
    - **Keyboard Navigation**: Full keyboard support
    - **Color Contrast**: WCAG AA compliance
    - **Focus Management**: Proper focus handling

4. **Animation System**
    - **Current**: Basic hover effects
    - **Enhancement**: Comprehensive animation library
    - **Features**: Page transitions, micro-interactions
    - **Performance**: GPU-accelerated animations

Infrastructure Enhancements

1. Monitoring & Analytics

- **Error Tracking**: Sentry integration
- **Performance**: Web Vitals monitoring
- **User Analytics**: Privacy-focused analytics
- **Alerts**: Real-time error notifications

2. CI/CD Pipeline

- **Current**: Basic build process
- **Enhancement**: Full CI/CD with multiple environments
- **Features**: Automated testing, deployment
- **Tools**: GitHub Actions, Docker

3. Content Delivery Network

- **Problem**: Single server dependency
- **Solution**: CDN for global distribution
- **Benefits**: Faster loading times worldwide
- **Implementation**: CloudFlare or AWS CloudFront

Built with by Johnny Driesen

May the Force be with you!