finiteloop

Movie Booking - Architecture

Build a scalable, robust, efficient app, Extensible and Configurable.

1. Introduction

A ticketing company is looking to build an web based application for customers to easily book movie tickets online. This document provides architecture perspective of the web based app.

1.1 Purpose and Scope

The purpose of this document is to describe the broad architectural, non-functional requirements and platform requirements, derive the architectural views and document the considered solutions and the recommendations made.

This document provides guidance in terms of the definition of layers, components, so that subsequent detailed design is aligned with the views set out here.

This document only focusses on frontend (User Interface) side.

This document does not contain any functional requirements or complete set of non-functional requirements, detailed design and or implementation guides

1.2 Background Context

Ticketing company would like to improve user experience, customer satisfaction by providing ease of lookup and book movie tickets online.

The web based application must be easily accessible mobile devices and tablets. It should also integrate with top social platforms w.r.to movies to get details around rating, reviews, popularity.

2. Architectural Tenets and Values

The architecture defined is guided by the below set of tenets and principles. Architectural and design decisions are based on these core principles.

Principle	Rationale	Impact
In the best Interest of the customers	The thought process is to deliver a solution that is optimal and maximizes benefit to the customer	• Build a system which is pleasurable to use, accessible anytime and anywhere • System must be monitorable
Uses Appropriate Technology	Makes the right and necessary technical choices based on the business needs. Does not overengineer or under-engineer. Business drives the change not IT	• Right Sizing but elastic to scale • Serviceable and fulfils needs
Is Design Led	Keeps the experience, design and assimilation to the user simple and predictable	• Understandable flows, screens • Provisioning for the mishap • Ability to retract or undo
Is Responsive and Resilient and Pluggable	Flexible to adopt to the changing business and IT landscapes Modular bouquet of services working together cohesively, adopts standards and is integrable	• Instant Access Anywhere, Anytime, Any Device • Customized, Configurable • Redundant and Recoverable • Standard and Extensible Processes and Data Flows
Is Component Based?	Emphasis on modeling, reuse, Components that are shared across project boundaries.	• Component Libraries for Reuse • Standard Practices for development

3. Primary Architectural Drivers

The factors that guide and drive the architecture from a business and technical perspective are recorded below.

3.1 Business Drivers

- Quickly and easily access required details

- Book movies faster with seamless experience
- Faster and predictable search tailored to user preferences
- Provide personalized experience through user's preference and search

3.2 Technical Drivers

- Application is responsive and accessible on mobile, tablets
- Application is accessible in the event of network unavailability / offline (limited functionality)
- Application is accessible to all set of users following W3C AA standards
- Application performs in optimized way despite variable network bandwidth / speed`

4. Architectural Constraints

- Payment will be static and does not integrate with actual payment methods
- Integration to social medial platforms will be restricted to availability of APIs and other limitations

5. Non-Functional Requirements

This section describes some the major non-functional requirements that influence the architecture and its decisions.

5.1 Security

- All users must be authenticated to access user specific tasks
- Use both Google Auth and Custom JWT based authentication (choice to user)

5.2 Performance

Following performance standard must be met

- Web performance
    - First Contentful Paint - <= 1500ms
    - First Meaningful Paint – <= 2000ms
    - Time to interactive – <=3000ms

5.3 Accessibility

Must adhere 100% to W3C AA standards for web accessibility

5.4 Extensibility

Application must be easily extensible to add new features, new components, adding additional integration systems with minimal impact

5.5 Monitorability

Monitoring of the application must be in place to monitor the health of services, access to logs, performance, throughput of I/O operations

5.6 Testability

Application must be unit testable through automated scripts and achieve at least 80% code coverage

6. Architectural Considerations

This section describes what has been taken into consideration before deciding on the architecture. The architecture is influenced by the above functional and non-functional requirements and these considerations.

6.1 Principles

The defined architecture is influenced and governed by the following principles,

- **Layering:** Frontend components should be multi layered ensuring separation of concerns. The enables easy maintainability and testability. Components should be split into container and presentational components.
- **State Management:** A global state and multiple local component state should be used for managing data to be shown to the user
- **Optimal Performance:** State should be persisted in local storage and service workers (web workers) should be used to cache assets, static data and update on master data with latest values
- **Cloud Enabled and Portable**: The chosen architecture is expected to be cloud based where appropriate with the ability to be portable across any cloud platform provider with little or no impact to the solution.
- **Security:** Users are authenticated before they can do authorized transactions. Environmental variable must be used to store secured data like keys, tokens…

7. Architectural Representation

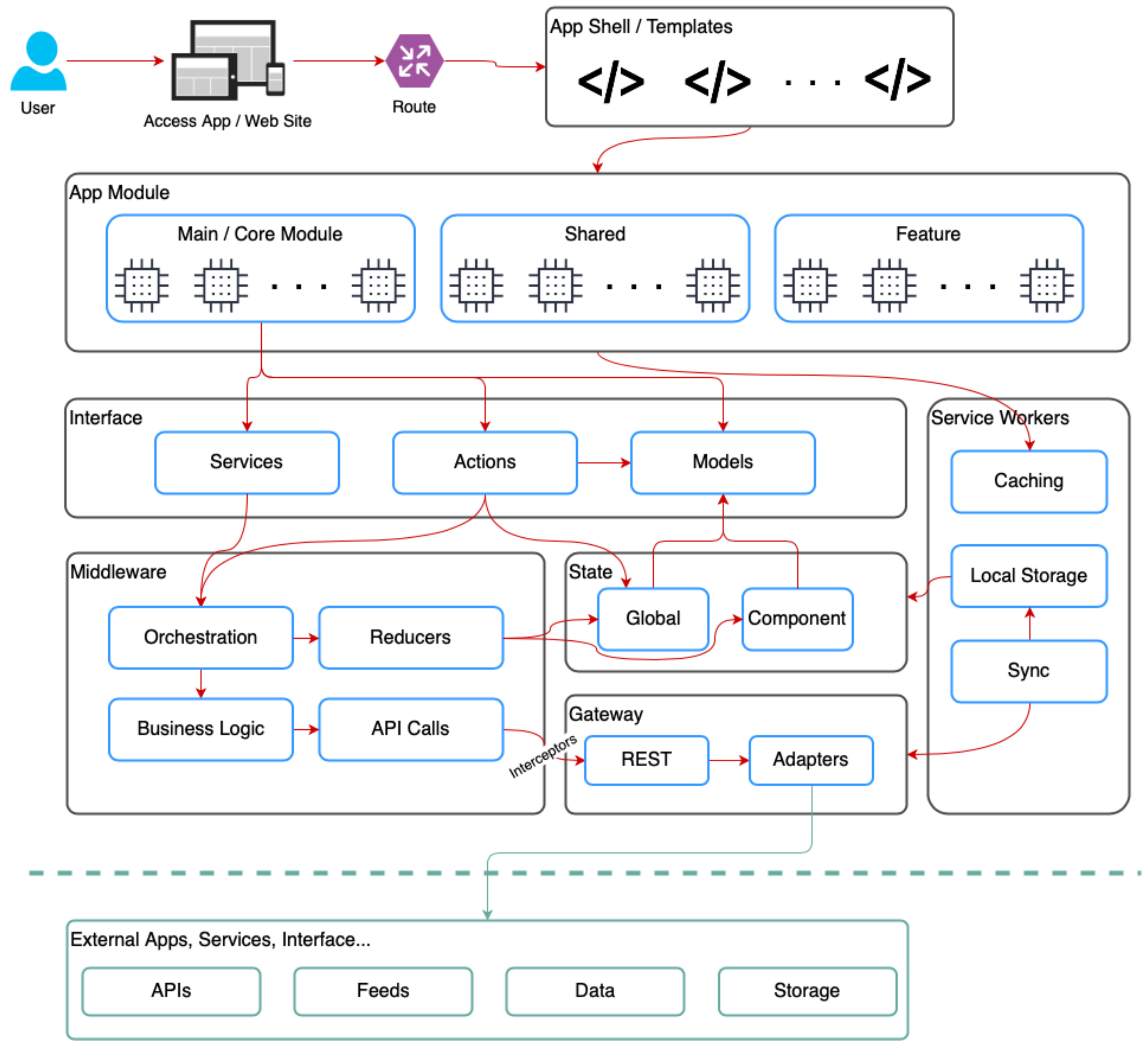
This section describes the architecture of Booking App through various views. Each view highlights unique aspects of the proposed solution

The conceptual view represents the usage scenarios in the system and the key user groups and sub systems involved.

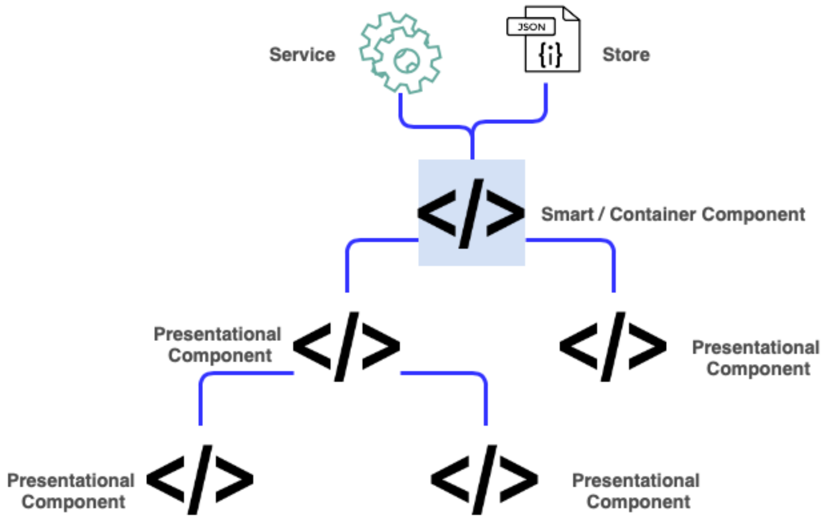
The logical view describes the logical partitioning of the application based on the functionality. This view provides the insight at the functional groups and can be cross checked to ensure all functional features are covered as part of the solution.

Application or Software view describes the various layers, components along with the indication of the software stack that would be utilized. This view helps in realization of the application architecture.

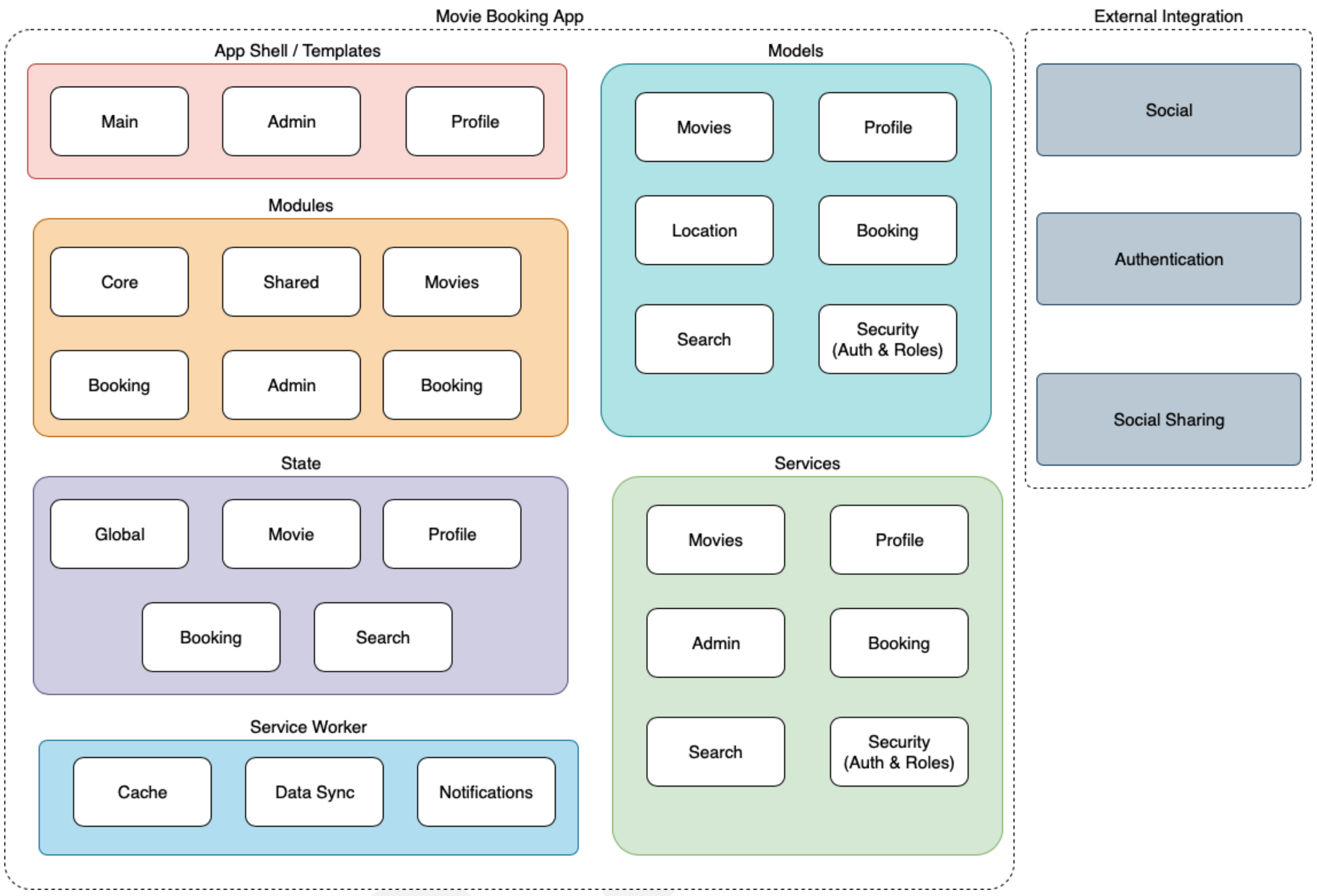
7.1 Conceptual View



- **App Shell:** This acts as template and used by all pages. More app shells are created for different layouts and used in appropriate pages.
- **Modules:** Modules should be split as follows,
  - **Core Module:** It should hold App Shell components and all other components used across all pages
  - **Shared Module:** All reusable components, common methods should reside here including notifications, pipes, configs, constants
  - **Feature Modules:** Following modules will hold respective components
    - Movies Module
    - Booking & Payment Module
    - User Profile Module
    - Admin Module
- **State:** State should be used for managing data shown to the user,
  - **Global:** A global state should be maintained which holds data used across the application like User Profile, Theme, authentication & authorization details etc.,
  - **Local / Component:** A local state will hold the data required for the particular component / page. This should include any master data, APIs data shown in the component
- **Components:** Components will be split into,
  - **Container Components:** These component access services, state / store to get data and passes them to child components to render them
  - **Presentational Components:** These component takes inputs from parent components for data and renders the views



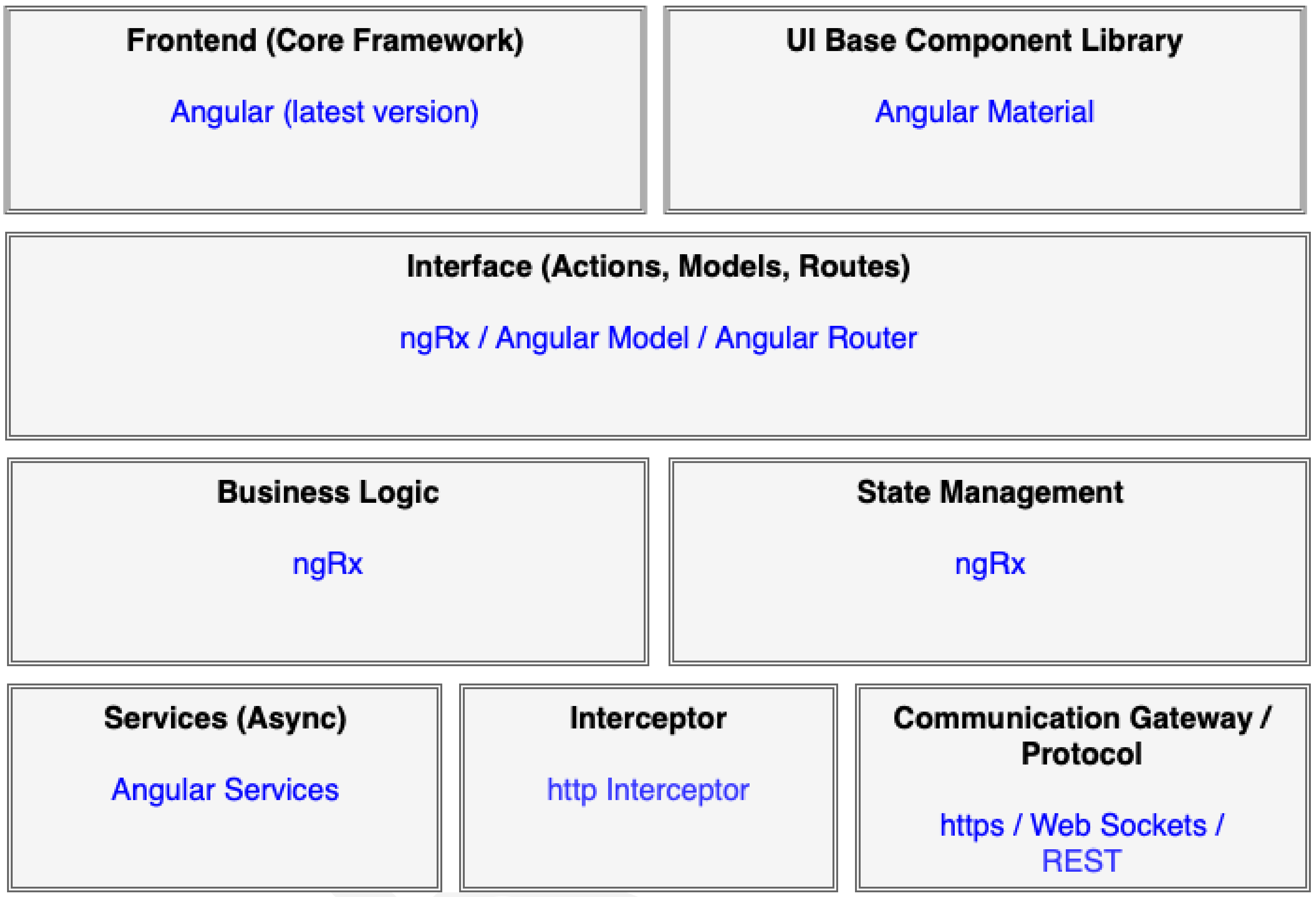
7.2 Logical View



8. Architectural Decisions

8.1 Technology Stack

8.1.1 Frontend


Frontend (Core Framework)
Angular (latest version)
UI Base Component Library
Angular Material
Interface (Actions, Models, Routes)
ngRx / Angular Model / Angular Router
Business Logic
ngRx
State Management
ngRx
Services (Async)
Angular Services
Interceptor
http Interceptor
Communication Gateway / Protocol
https / Web Sockets / REST

9. Appendix

9.1 Guidelines

- Core layout components which forms a shell includes header, footer, navigation (top, side). These components should be loaded by all child components / pages and hence to be part of Core and loaded once.
- Load the App Shell components through Route so that pages requiring different layout can use respective app shell.
- Make sure the 'Core Module' is loaded only once and imported only in 'App Module'.
- Use 'Interceptors' for adding header prefixes, caching requests, error handling.
- Use 'ngRx/effects' to handle errors and logging user actions.
- Keep the imports in 'App Module' to the minimum restricting to 'Core Module', 'Material Modules', 'Providers', 'Auth Modules'.
- Use 'Shared Module' for any reusable components, directive, pipes, services, UI interfaces like loaders, notifications, animations, date pickers…

9.2 Reference Projects

- https://github.com/DanWahlin/Angular-JumpStart
- https://github.com/ngx-rocket/starter-kit
- https://github.com/partha360/ng-ngrx-forms-demo

Contact

Partha – partha@finiteloop.io