MSK.NET #10

Web Development with ASP.NET CORE and React.JS

Web Evolution

Yesterday

Environment

- Predicted homogenous consumer environment
- Restricted types of clients (browser, desktop)

As a result this led to:

- Thick server and thin client
- Server is responsible not only for business logic, but for content rendering as well
- Client is responsible for content presentation

Today

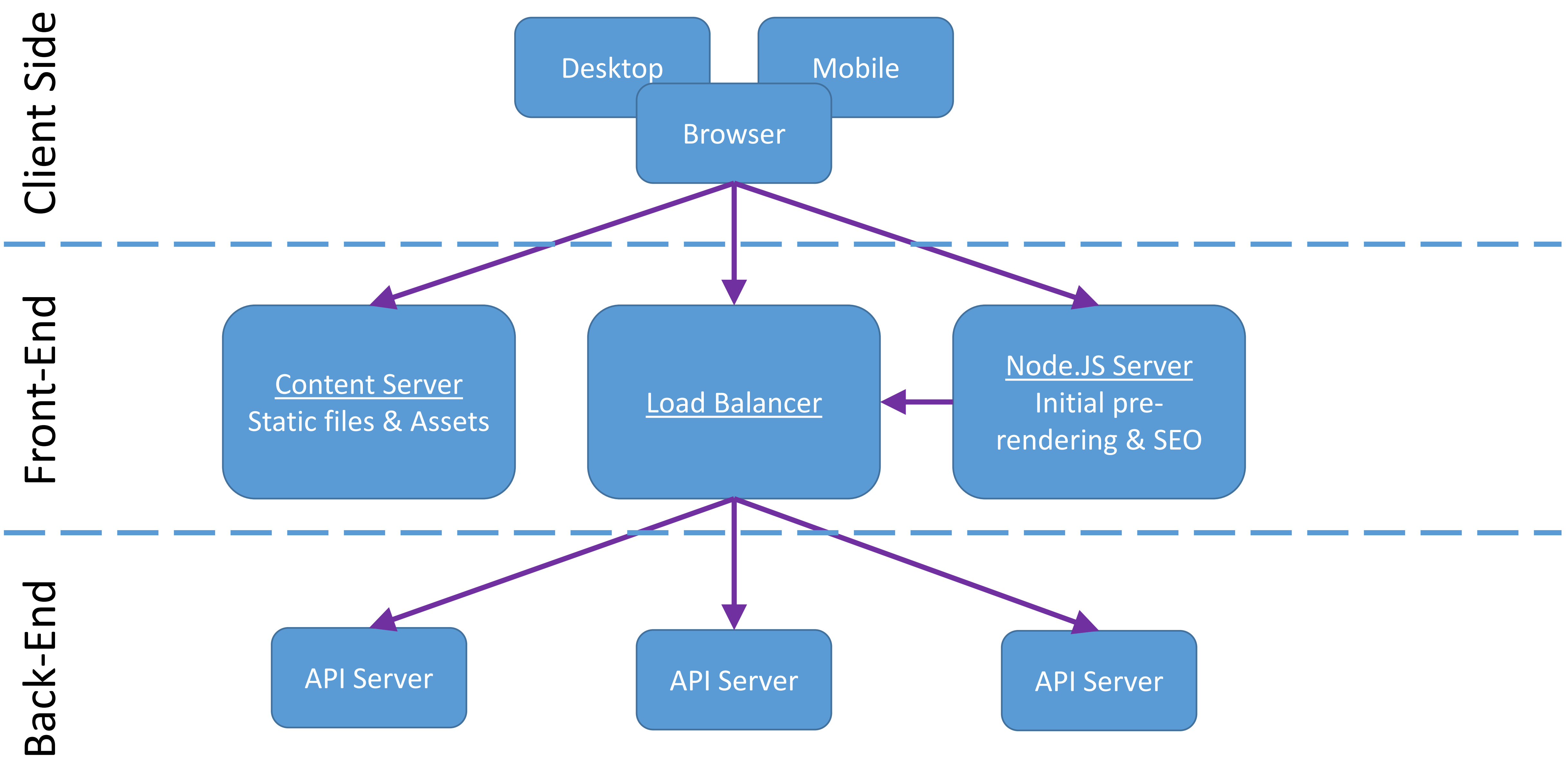
Environment

- Unpredicted heterogeneous consumer environment
- Unrestricted types of clients
- Different browsers, different devices with different form factors
- Not only desktops, but mobiles as well
- Not only people, but applications and machines

As a result this lead to:

- Segregation of Duties between server and clients
- Standards in connectivity protocols and data formats
- Server is responsible only for business logic
- Server is exposed outside only data and services via API
- Clients are responsible for content rendering and presentation

High-Level Architecture



Technology Stack

Client

- BROWSER: Chrome
- PRESENTATION & LAYOUT: Bootstrap
- TOOLS: React & Redux DevTools for Chrome (debug, monitoring)

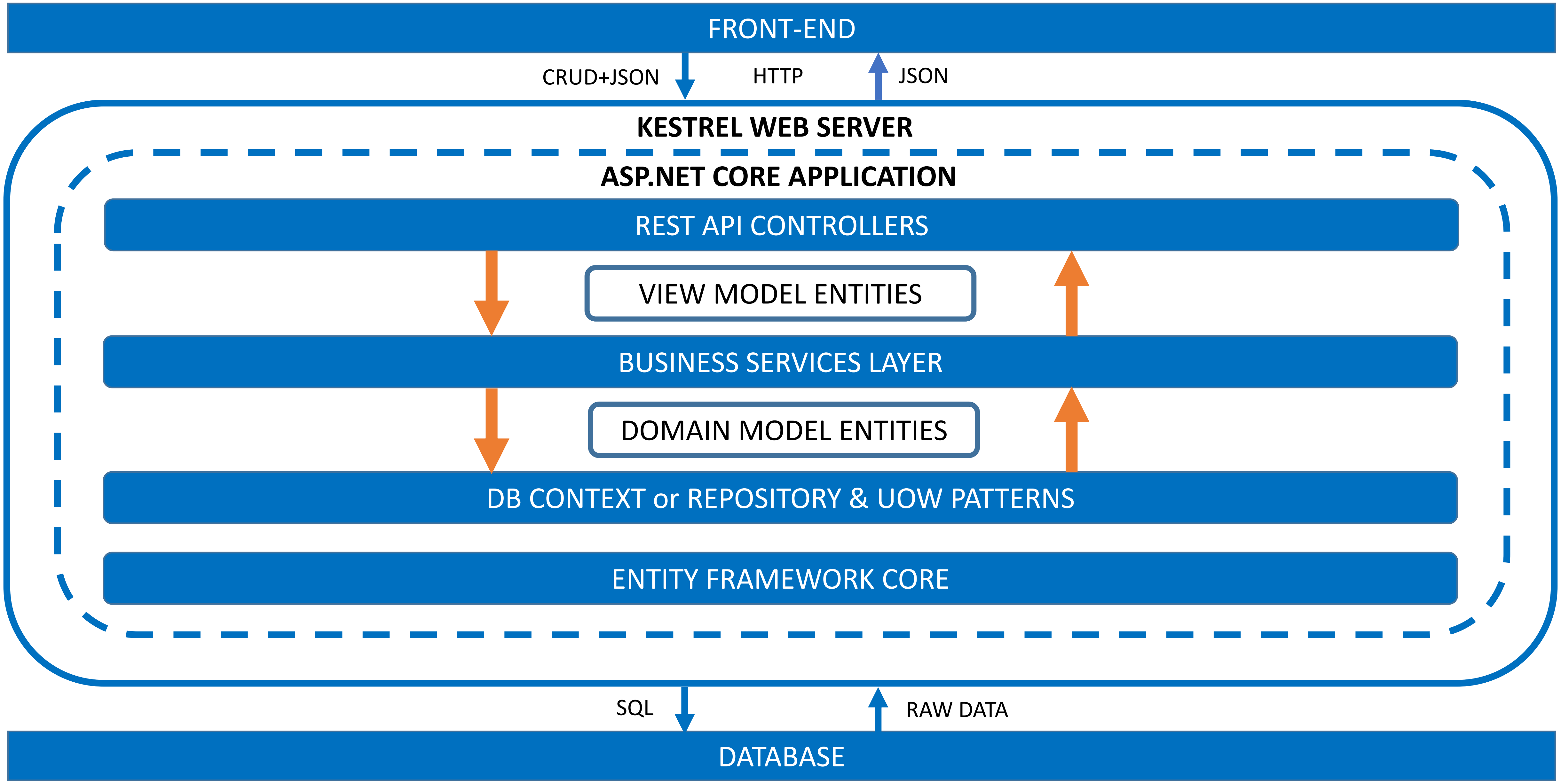
Front-End

- IDE: Visual Studio Code, IntelliJ IDEA 2017
- VIEW: React.JS
- MODEL: Redux
- TOOLS: WebPack (bundling, minification, cross-compilation with babel)
- HOSTING: Apache, Express, Firebase, Node.JS

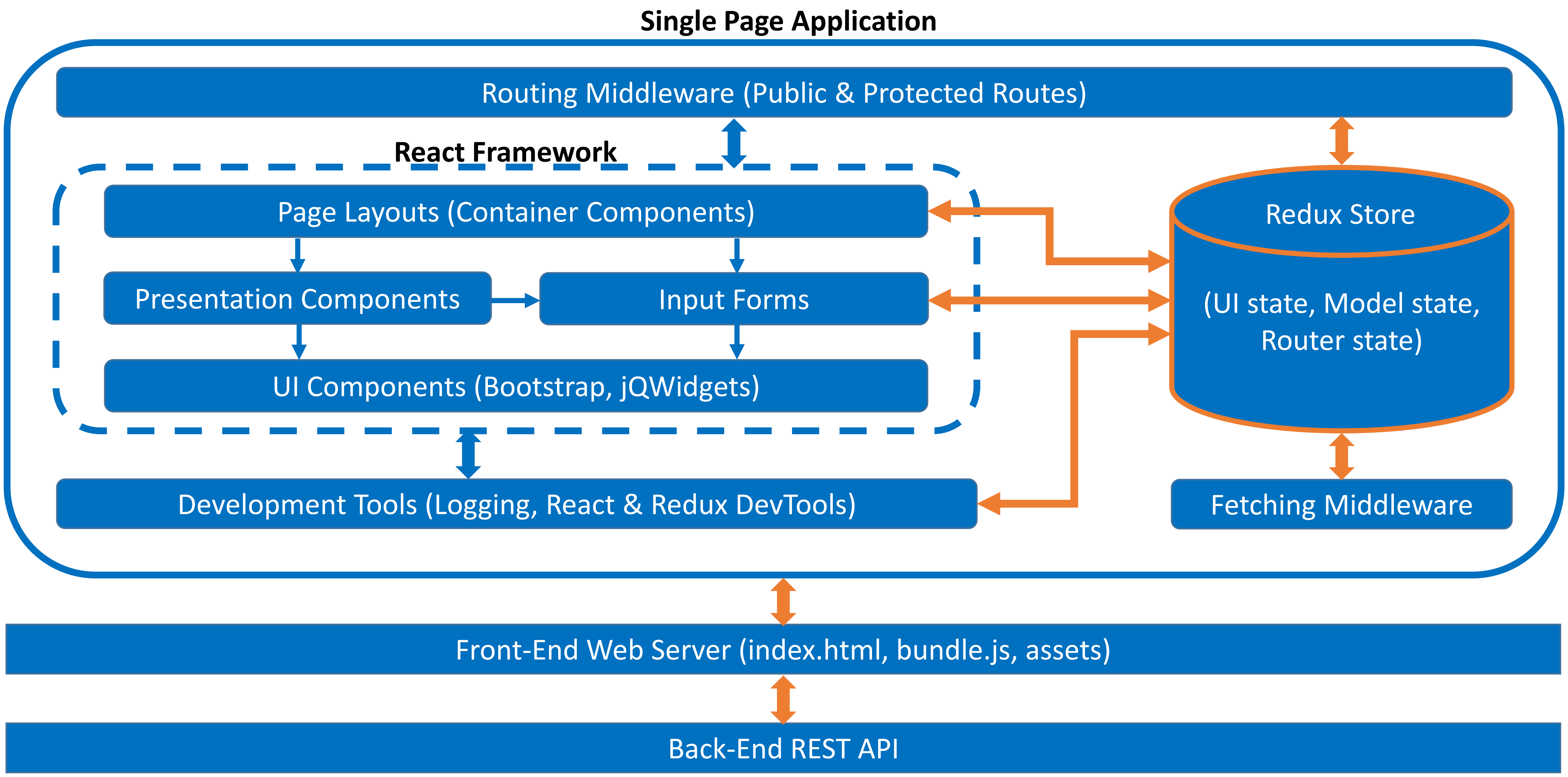
Back-End

- IDE: Visual Studio 2015 with Update 3, Visual Studio 2017
- SOLUTION: REST API, ASP.NET CORE
- ORM: EF.CORE
- DATABASE: MS SQL SERVER
- HOSTING: KESTREL

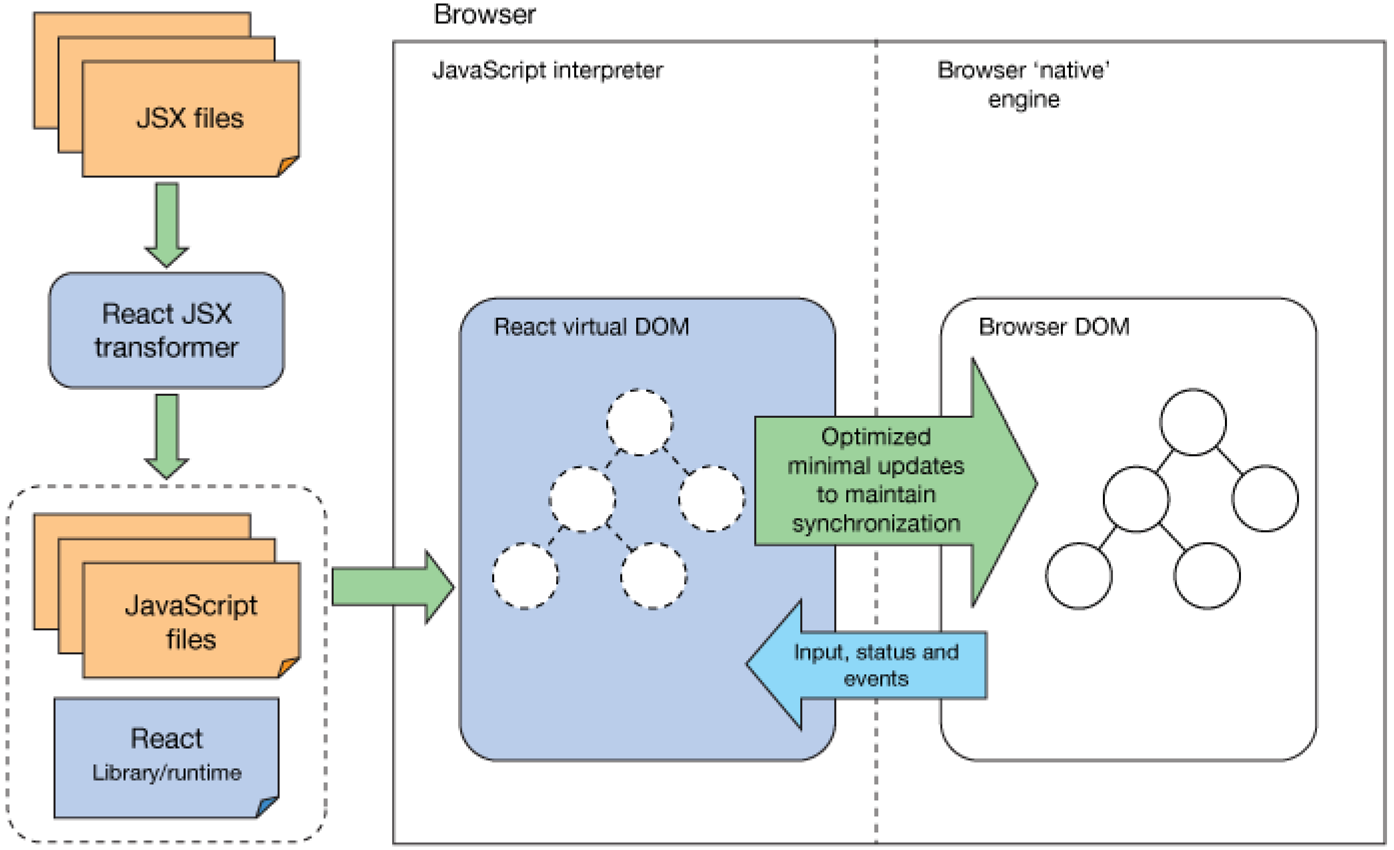
Back-End Architecture



Front-End Architecture

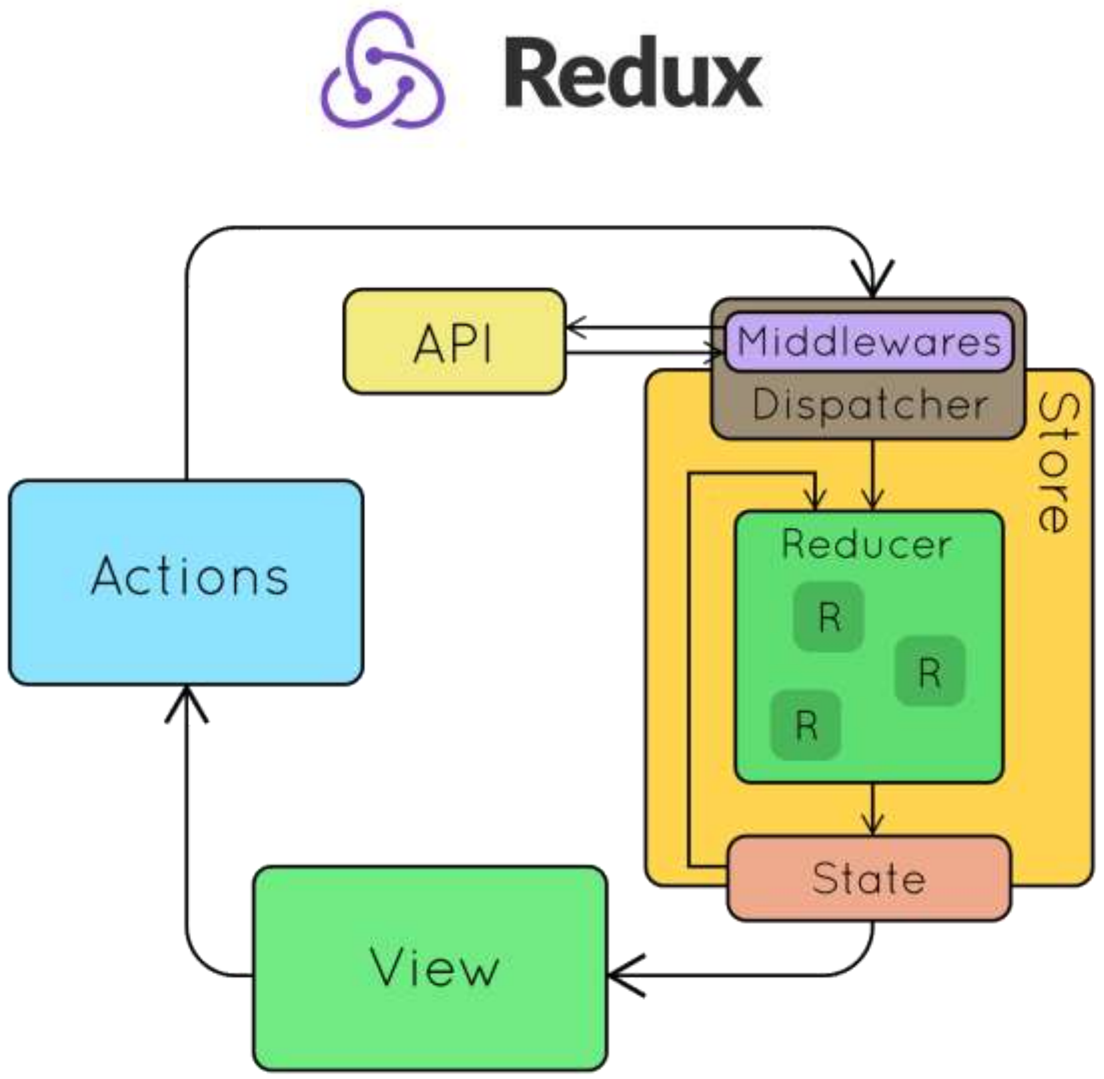

Single Page Application
Routing Middleware (Public & Protected Routes)
React Framework
Page Layouts (Container Components)
Presentation Components
Input Forms
UI Components (Bootstrap, jQWidgets)
Redux Store
(UI state, Model state, Router state)
Development Tools (Logging, React & Redux DevTools)
Fetching Middleware
Front-End Web Server (index.html, bundle.js, assets)
Back-End REST API

React Architecture



- React means what is "V" in MVC pattern.
- React is declarative and component-based. It mixes JavaScript expressions with standard HTML markup.
- React uses one-way data flow via "props" which are down streamed from root element to all children in component hierarchy. React components are pure functions with props as input and rendering as output. Props are immutable.
- React creates virtual DOM for components tree. Manipulations are done in virtual DOM, because interventions with objects in browser DOM are much more expensive then interventions with POJO objects in JavaScript runtime. On each rendering React reconciles changes between current UI state and next UI state and only applies necessary updates in browser DOM.
- React has a powerful composition model and encourages using composition instead of inheritance. It is possible via children prop and high-ordered components.

Redux Architecture



- Redux attempts to make state mutations more predictable. With some principles below.
- **Single source of truth**. The state of your application is stored within one single store.
- **State is read-only**. The only way to change the state is to make an action.
- **Changes are made with reducers**. Reducers are pure functions which take the current state, an action and return the next state. No side effects. No API calls. No mutations. Just a calculation.
- **Components are pure functions.** Your React components are subscribed to necessary slice of application state in Redux store which is mapped to components props. Once changes they are interested in are happened, components are automatically re-rendered with new props extracted from the new state.

Live Coding

- Let’s create a simple web application – classic TODO list.
- Develop back-end with Asp.Net Core.
- Develop front-end with Reacj.JS and Redux.

Web Development Tomorrow

- What else? React Native.
- What is next? React new generation – React Fiber.
- No cross-compilation. No interpretation. Web Assembly.
- No vendors. Try Open Source.

Contacts

- Speaker: Dmitry Tezhelnikov
- Email: dmitry.tezhelnikov@gmail.com
- Back-end demo source code: https://github.com/DmitryTezh/msk.net-backend-demo.git
- Front-end demo source code: https://github.com/DmitryTezh/msk.net-frontend-demo.git
- Video: https://www.youtube.com/watch?v=wg6QPyxDDho