



High-Level Design

Date Submitted: 10/6/2021

Team Hobby:

Colin Creasman

Daniel Bribiesca

Team Lead: Jacob Delgado

Long Nguyen

Rifat Hasan

Document History

Date	Version	Changes
9/29/2021	0.1	Initial Architecture Draft
10/1/2021	0.2	Restructure Backend; Control Flow of Initial HTML
10/2/2021	0.3	Incorporate Security Protocols to Control Flow
10/3/2021	0.4	Remove Technology Implicit Components
10/4/2021	0.5	Add Additional Error Handling in Backend
10/4/2021	0.6	Fix Control Flow for Token Authentication
10/6/2021	0.7	Restructured Architecture for MVVM Frontend and Microservice Backend

Contents

1 - Design Choices

- 1.1 - Requirements Overview
- 1.2 - Frontend Architecture
- 1.3 - Backend Architecture
- 1.4 - Advantages
- 1.5 - Disadvantages

2 - Control Flow - Initial Request

- 2.1 - Request Processing
- 2.2 - Response Processing

3 - Control Flow - Asynchronous Requests

- 3.1 - Request Processing
- 3.2 - Updating Data
- 3.3 - Response Processing

4 - HLD Diagram

1 - Design Choices

1.1 - Requirements Overview //TODO

As outlined in the BRD, the general functional requirements of the application from a system's high level view of the system can be condensed as follows:

- Project links can be collected from external sources
- An unregistered user can create an account with a custom profile
- An unregistered user can search for projects
- A registered user can be recommended projects and tools based off their profile
- A registered user can post a project to the site
- A registered user can remove a project from the site
- A registered user can comment on projects
- A registered user can receive various project notifications

1.2 - Frontend Architecture

In order to optimize the SPA functionality of our application, the frontend utilizes a partial-MVC architecture with client-side controllers and views living in the browser corresponding controllers in the server-side. The client-side controller is the entry point of the application and is specific for the browser requests that routes to it. It also acts as the middle man between the client and the server. This is done in order to preserve and complement the performance benefits we get from an SPA from a lack of browser refreshes. By using server-side controllers on the frontend MVC architecture to the backend, we are ensuring that all requests for a model can be filled asynchronously within the browser - without needing to be re-routed to the appropriate model for every request. This effectively minimizes server communication for asynchronous requests from the browser because frontend controllers only need to retrieve necessary model data from the backend once, then all the requests routed that model can be filled wholly within the browser. This eliminates the need for server communication from the frontend as long as additional requests utilize the same model; thus additional routing is only required for requests for different models.

1.3 - Backend Architecture //TODO

1.4 - Advantages

- Backend is reusable and completely separated from frontend development. Good for MVC architecture.
- Increased load speed and responsiveness because all content of the single page is loaded automatically.

- Linear experience that is more straightforward
- Enhanced user experience, particularly for mobile

1.4.1 - Requirement Fulfillment // TODO - explain why the chosen design/architectures satisfies the non-functional requirements from BRD

1.5 - Disadvantages

- Lack of multiple unique links
 - Makes search engine optimization difficult
 - Hard to index different parts of the site
- Navigation
 - Users cannot go back and forth between page “states” as they could with pages in MPA

1.5.1 - Requirement Sacrifices // TODO - show where this design falls short of filling non-functional requirements of BRD

2 - Control Flow: Initial Request

2.1 - Request Processing

When the browser initiates the first page load, the initial HTTP request is sent to the client-side controller. Since this is only a request to load the home page of the application, no user data is sent along with it. This means that the controller doesn't require any validation nor authentication for the request; it simply processes the URL mapping and sends it straight to the backend.

The server-side controller then receives the request; and, based off the URL indicative of the initial page load, knows that it doesn't require any input validation/authentication on the backend either. The request is simply translated and sent through business logic to the persistence layer to get the updated data for the response.

2.2 - Response Processing

In the persistence layer, the initial request queries all the data from the database that is needed for the initial page to be loaded and sends it to the server-side controller. Even though the initial page that is eventually returned to the browser is just a partial rendering of the DOM, all of the available script data for the entire DOM is sent back from the persistence layer - but only the raw relational data for the initial page is sent back with it. This is so that all partial DOM data for every possible view is sent back with the initial response where they are downloaded and stored in the view component - that

6. way all additional views after the initial request can be rendered on the client's side.

Before returning to the frontend, however, the server data has to be translated from its raw relational form into script data (e.g. JavaScript) and abstract object data (e.g. JSON). A unique header and footer for the initial page are added so that the client-side controller can accurately map the object data to the right view template. Everything gets sent back to the frontend as a single payload that the client-side controller receives.

The client-side controller then takes all of the DOM script data from the payload to simultaneously translate and separate it into all the unique view templates for the entire application. All those empty view templates get sent to the view component where they will continue to live for the remainder of the active session. The remaining portion of the payload, however, (the object data for the initial page) gets sent straight to the view component. Once there, it can be bound to the initial page's view template. This renders the view for the initial page that is shown in the response to the browser.

3 - Control Flow: Asynchronous Requests

3.1 - Request Processing

3.1.1 - Input Validation

Just like the initial request, asynchronous requests go first to the client-side controller which processes them before going to the server. Any input with simple logic constraints is checked here within the client-side controller so it can be validated asynchronously without requiring server communication.

3.1.2 - Login

When a user first logs in during a session, their credentials are sent to the persistence layer where they will be used to verify the security of additional requests

3.1.3 Requests After Login

For data security, every request after logging in must be authenticated before the server can send back a response. This is done by verifying that the credentials match. If authentication fails, however, the error is caught and logged to a server file for login errors

3.1.4 - Business Logic

After validating input, the request is translated and any necessary business logic is performed inside the server-side controller. Any request errors - a request mapping to a URL that does not exist, for example - are caught and logged to a server file for request errors. The request data is then sent to the persistence layer .

3.2 - Updating Data

The incoming request data is first validated within the persistence layer so that verification requiring complex logic can be validated using server data queried from the database. Any necessary data logic is also performed in this component, as well as database queries for all types of requests. The layer also listens for any data errors that might occur during queries. After being queried, the requested data is sent back to the client-side controller in its pure relational form (e.g. XML). Finally, any data errors that occur during queries are caught and logged to a server file for data errors.

3.3 - Response Processing

3.3.1 - Data Packaging

The raw relational data from the persistence layer must first be translated into agnostic object data (e.g. JSON). Furthermore, a unique header and footer are attached to it so that the client-side controller can accurately map it once sent back to the frontend. Everything gets packaged into a data transfer object (DTO) which allows the response to move between frontend and backend components agnostic to any one component's technology. Finally, the DTO response can be sent back to the client side controller on the frontend.

3.3.2 - Receiving the Response

The response is sent to the client-side controller which maps it to the appropriate view using its unique header and footer data. Inside the view component, the object data is bound to its template using the corresponding partial HTML from the DOM. This renders the view that is displayed in the response to the browser.

4 - HLD Diagram



