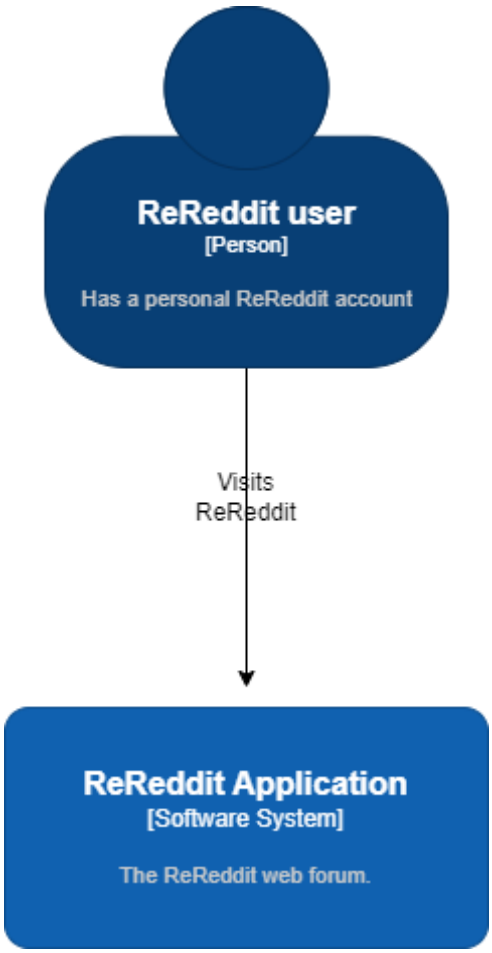
Design Document ITS

ReReddit

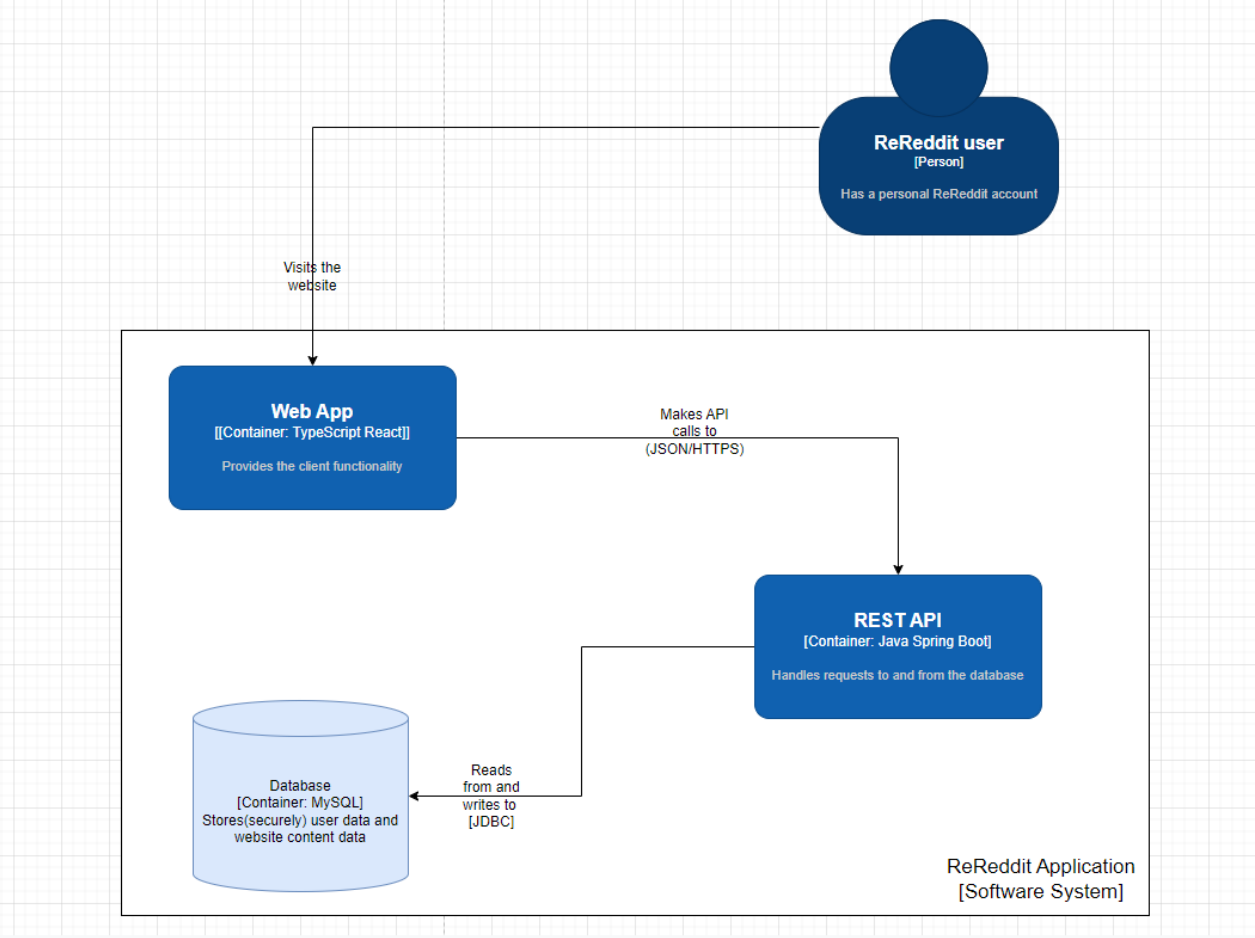
Architecture

C1



The application is designed so that no external dependencies are consumed and the whole of the implementation consists of the ReReddit Application itself, which is composed of multiple software components under the hood. In the above diagram(C1), the interaction with the user is described as a visit to the URL, after which the normal flow of interaction follows.

C2

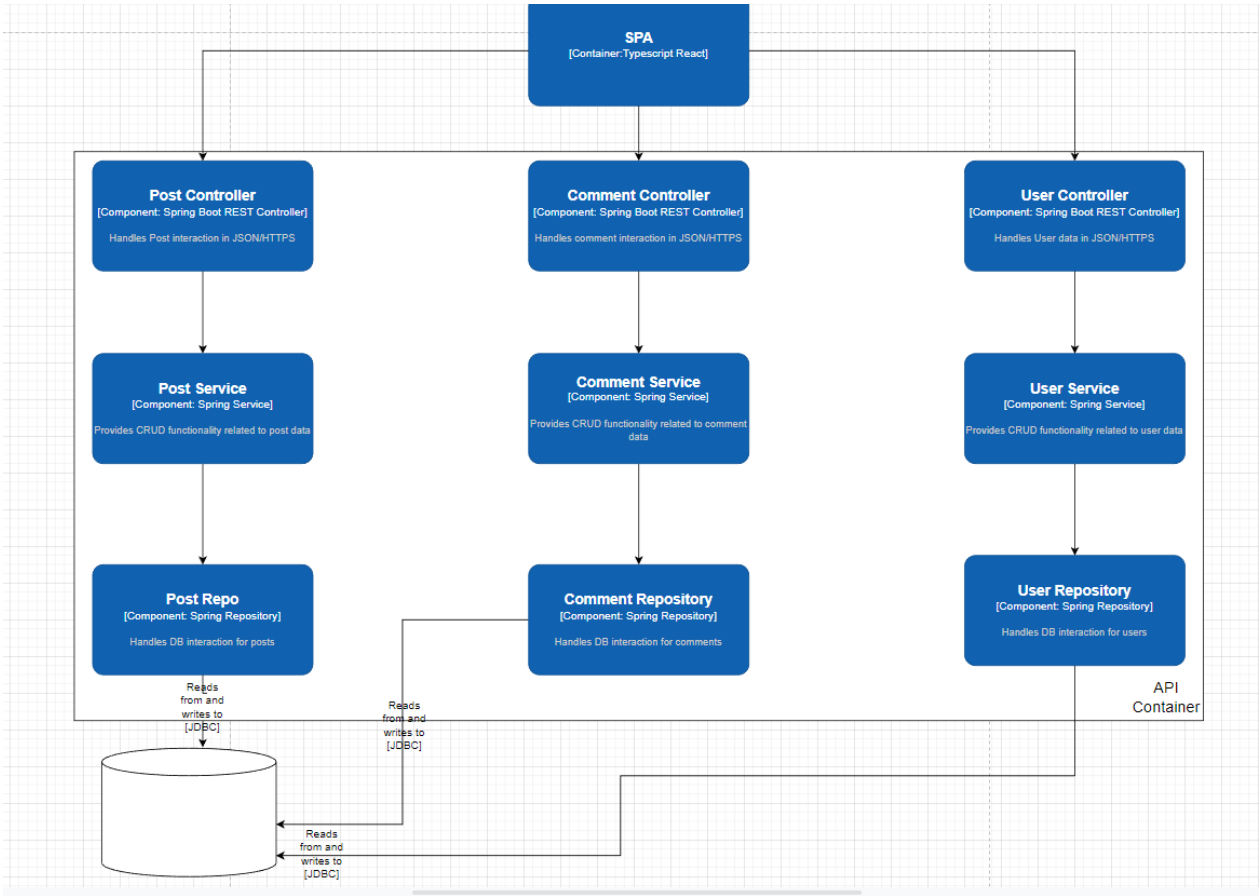


Taking a closer look at the application container, we can see that it is made up of 3 components, namely: The Single-Page Application(Web App) serves as the presentation layer for the entire system. It's made with TypeScript React and uses the Tailwind framework for styling. The back-end REST API, made with Java Spring Boot and lastly the MySQL database.

Upon the client wanting to send data via a form or to view a particular object, a request is sent to the back-end API, which processes said request and returns a response containing the relevant data that the user requested.

To ensure data persistence, the API also uses the JDBC interface to provide a connection to the MySQL database, which is used to write or read data to the schema. Something that is important to be mentioned here is that the flow of data in the diagram is shown to go only towards the database, meaning top-down. In actuality, the flow goes both ways. It is not depicted in the diagram because the arrows overlap and the look is not clean.

C3



Finally, as we inspect the API container, we can see that the layering structure is very standardized, according to common practices. It is done using Java 18 with the Spring Boot framework. At the top of the container and outside of the hierarchy is the Single-Page Application(SPA).

As a request is sent to the API, it gets processed inside the service layer. The layer's structure is made up of interfaces which represent the certain use cases used in the SPA, (e.g registering a user). Said interfaces define the architecture that is to be used and are therefore implemented by implementation classes, that contain the actual business logic. During runtime, Spring looks for implementations of these interfaces and injects them via a constructor injection. This particular approach to laying out the business layer is especially useful for abstraction, since the implementation can be changed without necessarily tinkering with the pre-defined architecture. As a result, maintainability and scalability are significantly increased and the code is just cleaner overall.

Next up, the persistence layer takes care of all database interactions and handles data persistence. This is done using JPA(Java Persistence API), with Hibernate as the Object Relational Mapping tool of choice.

To finish up, the result all of the data processing by the business and persistence layers is sent to the controller layer, where the response to the client is sent from. These controllers return response entities of a specified type, which are sent back to the SPA in JSON(JavaScript Object Notation) format and resolved to the proper TypeScript type.

To adhere to API design standards, in the event of an error occurring at runtime, custom exceptions and a custom REST exception handler is implemented in the business layer to handle these exceptions and return proper error messages. This helps consumers of the API (if any) to identify if they should correct their usage, or if the server itself encountered an internal error.

Design Decisions

As planning was initially done for the project, a lot of important and lasting design decisions had to be made. This is an unordered list of some of the most important ones, with explanations given as to why they're used instead of any other option.

- Front End JavaScript Framework --- React
  - It was either React or Angular for this particular choice. Both have significant benefits and drawbacks to this particular use case. Here is a list of each of them:
    - React:
    - Pros:
      - Has included examples and assignments on Canvas
      - Most popular choice for front-end development at the moment
      - A lot of resources online
      - Functional design and hooks
    - Cons:
      - Slower than angular in some cases(JSFrameworkBenchmark, 2020)
      - Steeper learning curve(functional programming)
    - Angular
    - Pros:
      - More experience before semester 3 with it
      - Imperative programming

- HTML is separate from TypeScript logic

- Cons
  - No material on Canvas
  - Less frequently used in the industry
  - Outdated material online (most is on angular 12, latest is 14)

- TypeScript

  Pros:
  - TypeScript is just a convenience used to substitute regular JavaScript as a programming language. It also provides a significant advantage in debugging, code readability and maintainability. Knowing what a function returns is also really convenient.
  - TypeScript code is compiled to JavaScript at runtime without affecting performance.
  - API responses can be easily mapped to custom types or interfaces.
  - Provides type support for JSX elements.
  - Compatible with React and most libraries on NPM that provide type packages along with the implementation.

  Con(s):
  - Build time is slightly slower than regular JavaScript.

Back End Framework – Spring Boot (Java)

Due to not being familiar with APIs, Spring Boot is the safest option, since a lot of canvas material covers how to use it properly. Additionally, it's a really widespread framework and with Java being one of the most popular programming languages, it is definitely worth to learn.

Hashing Algorithm – Argon2 (C)

All explanations and answers to questions can be found in the research document on https://git.fhict.nl/I478554/sem3-its/-/tree/documentation

References

Stefan Krause, JavaScript Framework Benchmark, (2020), GitHub Repository,

https://github.com/krausest/js-framework-benchmark