Software Design Document

1. Introduction

1.1 Purpose

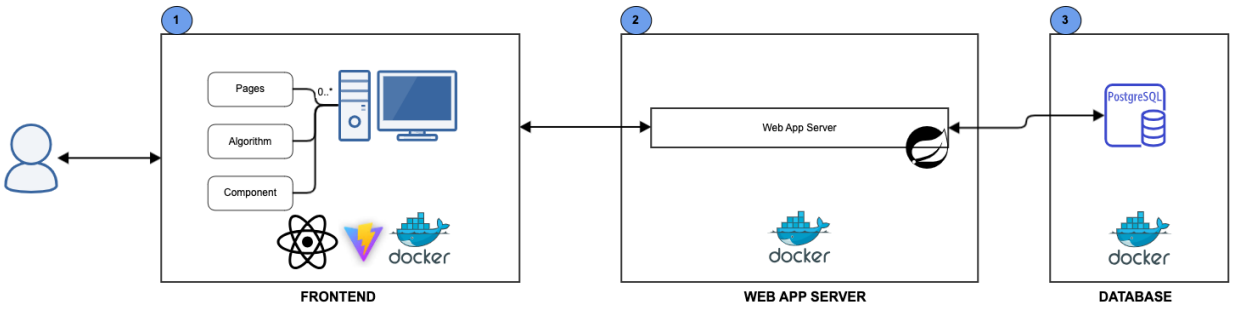
This document describes the design of Algo-Shadow web application.

1.2 Scope

This design document is for version V1 of Algo-Shadow, which includes high level architecture, modules, and UML diagrams. Since this application uses existing frameworks and libraries (SpringBoot, Vite, and Reacts), this design document excludes framework classes.

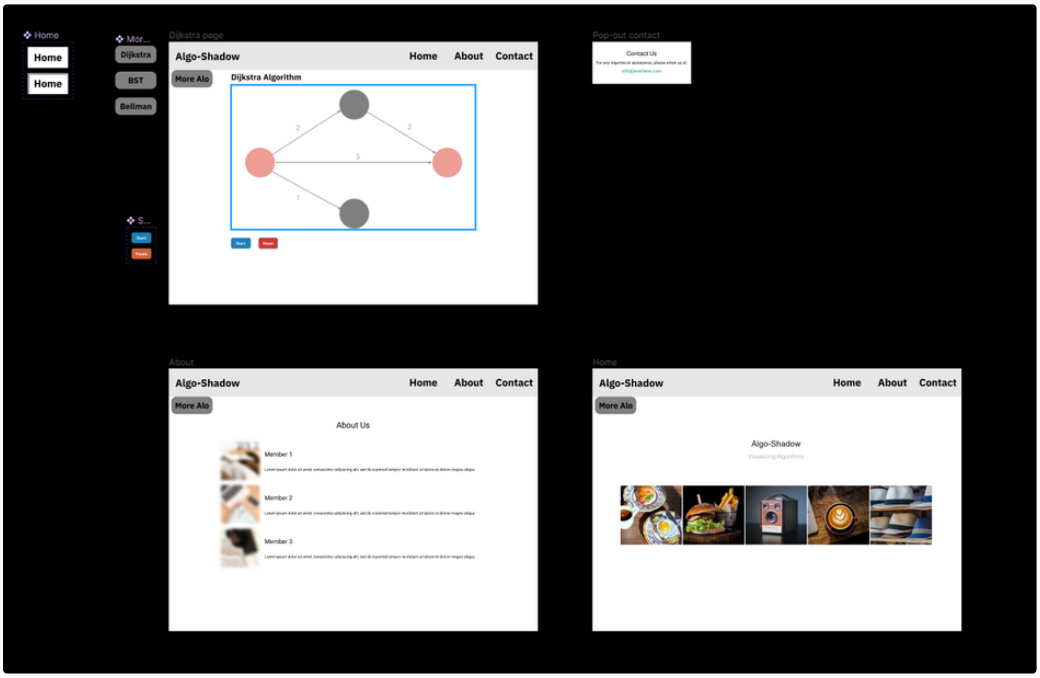
2. High Level Architecture

This application resembles a Client-Server-Database architecture, which comprises of 3 major components, including a web application, a web server, and a database. The client communicates with the server via REST APIs to then perform standard database functions. In addition, we choose a thick-client application that carries out algorithmic computations and animations to reduce load to our server and achieve a faster performance.



3. Front-end Application design

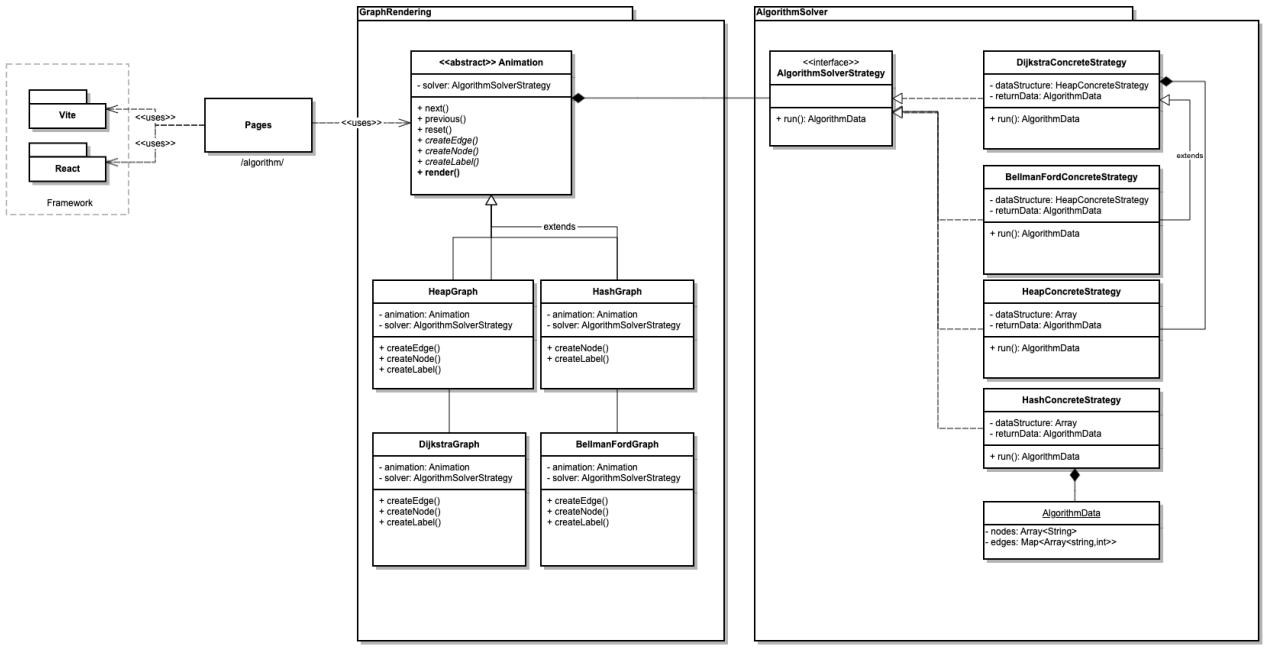
3.1. UI design



3.2. AlgorithmSolver and GraphRendering modules

The AlgorithmSolver module consists of different algorithmic logic implementations, such as Dijkstra, Heap, etc. This module is responsible for running the algorithm, and returning the result for rendering.

The GraphRendering module consists of an AlgorithmSolverStrategy from AlgorithmSolver module, which is instantiated when the user visits the url endpoint associated with each type of algorithm, for example `/algorithm/heap`. This module is responsible for web rendering and animation using `d3` and `svg`.



4. REST API design

Our web application communicates with the server via REST APIs, below is the list of APIs we are implementing

---

Experiment lookup by User Id

Returns the last 30 experiments by the user

```
GET v1/experiments/:userId?limit=30
```

Parameters:

Name	Type	Description
userId (Required)	String	Unique identifier of the authenticated User to request data back

Response

```
{
  "data": {
    "id": "12345",
    "experiment_id": "12345",
    "experiment_type": "heap",
    "experiment_status": "success",
    "actual_runtime_ms": "900"
  }
}
```

Post an experiment by User Id

Save current completed experiment to Database on behalf of the authenticated user

```
POST v1/experiments/
```

Response

```
{
  "data": {
    "experiment_id": "12345",
    "experiment_type": "heap",
    "experiment_status": "success",
    "actual_runtime_ms": "900"
    "insert_ts": "2020-01-01 12:12:12.0000"
  }
}
```

5. Back-end Server application design

5.1. AuthenticationService module

- We will integrate personal authentication project API to do the authentication and authorization for this application.

API Name	Description	Parameters
POST v1/auth/signup	able to sign up	`email`: string `password`: string `lastName`: string `firstName`: string `buID`: string `role`: number
POST /auth/login	able to sign in	`email`: string `password`: string

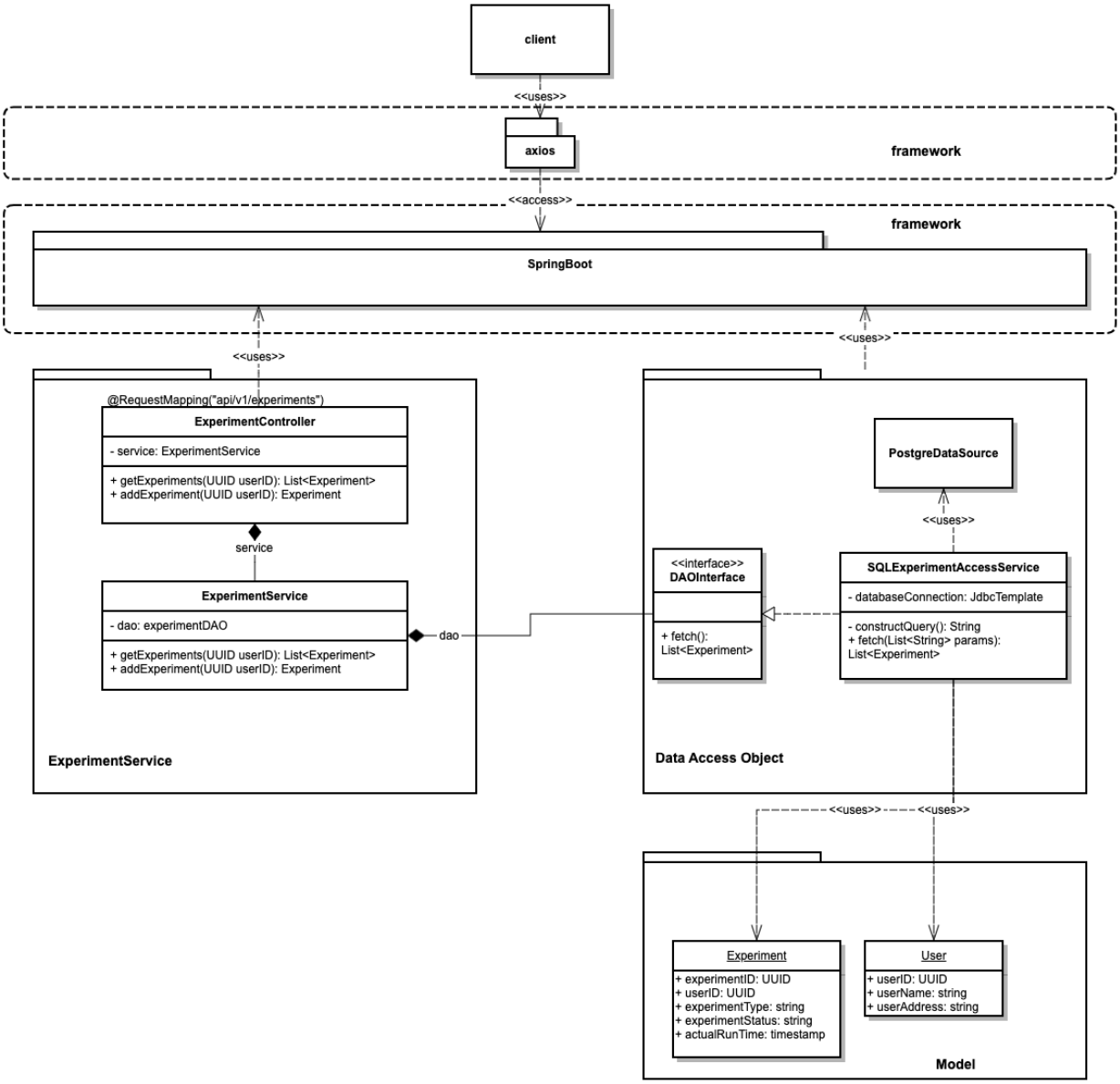
5.2. ExperimentService, DAO and Model module

The `ExperimentService` module has a `Controller` to map routes to underlying application logic that is encapsulated in `ExperimentService` class. This module is responsible for accepting requests from a client, retrieving data from `Datasource`, then applying logics to retrieved data before returning to the client.

The `DataAccessObject` module is a layer between the server's application logic and the actual Database server. It is responsible for handling actual extraction query, connection, and communication with the actual Database server.

The `Model` module consists of all object models, such as `User`, `Experiment` , etc.

Below is the UML Class data for Experiment Service module



6. Database

Database Engine: PostgreSQL

Host: (TBD)

Below is the current ERD