Bilkent CS319

Deliverable 4 : Solution Level Class Diagram

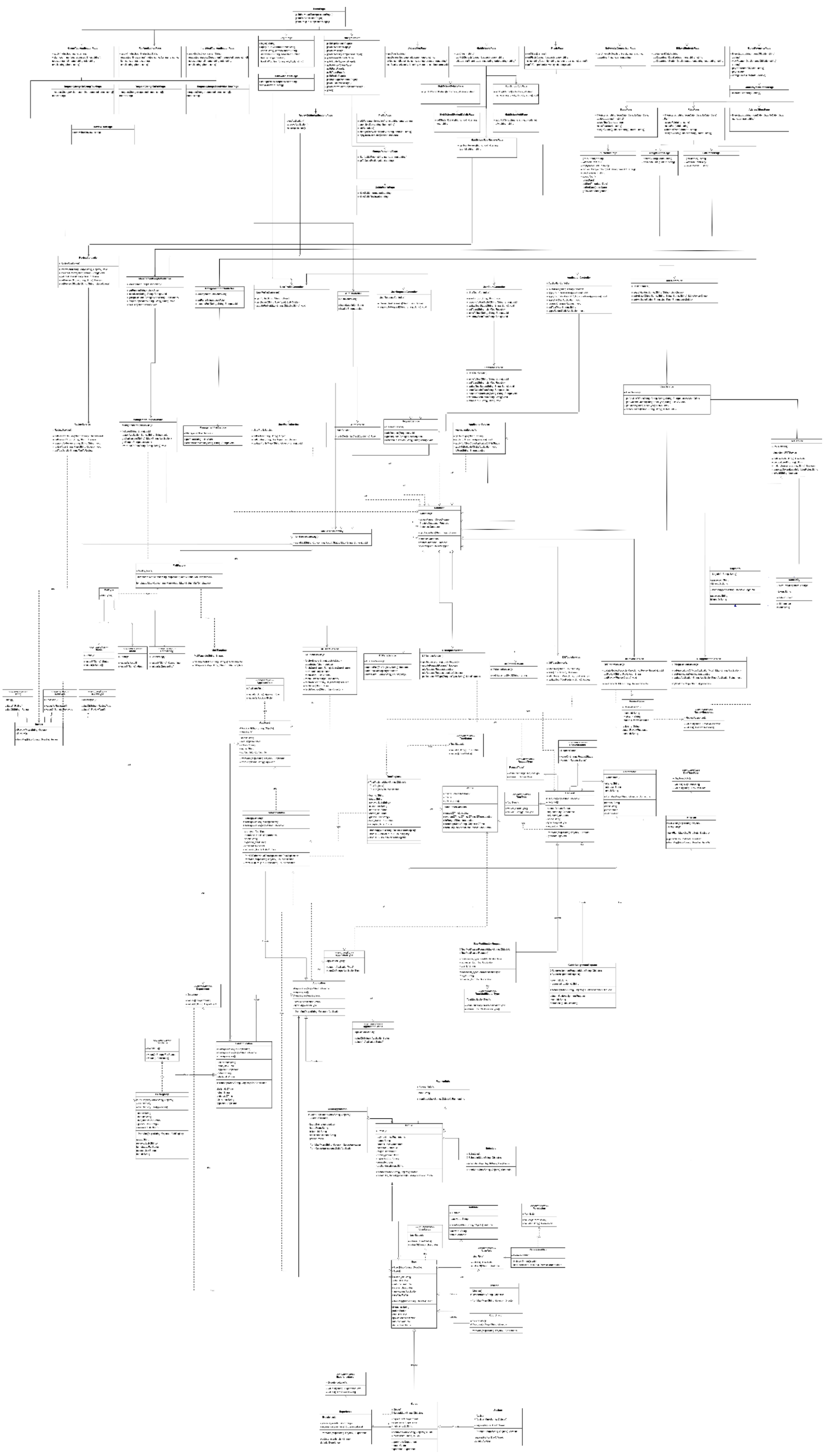
22202321 Orhun Ege Çelik

22203239 Can Tücer

22203470 Begüm Filiz Öz

22202913 Göktuğ Ozan Demirtaş

22202995 Mehmet Emin Avşar



The full view of the class diagram is given in the link below:
https://drive.google.com/file/d/18s_YZF1attnlbQh17C9H1k0y0SPx_3VB/view?usp=share_link

Facade Pattern

We delegate logic and responsibilities to the backend, instead of doing these in the frontend. This provides a level of security, so that no user has direct database access. This also allows for permission delegation, restricted and censored information access. We achieve this in the following way:

1. Spring Controllers handle network connections, and parameter parsing. They do not contain any business logic on high level response handling. The requests are forwarded to related services.
2. Spring Services handle the high level logic of the request. This includes authentication check, and other request specific logic. Non-special logic is delegated to other services. For authentication, the system uses Javascript Web Tokens or JWTs. There are medium length encoded strings with some information embedded in them. These are handled by JWTService.
3. For database interactions, we use specialized services. These are abstractions on top of the Firebase Firestore service.

This Facade Pattern applies to the entirety of the backend services. Endpoints are context-sensitively grouped into Spring Rest Controllers, which handle the network side of requests, logic is delegated to Spring Services, which can access other services. These abstractions allow for proper unit testing capability, along with easier debugging.

Singleton Pattern

Most pages in our website need access to the current user to display the relevant information. These can be buttons, text or any other elements. To facilitate this, we make use of React Contexts, which in our project is effectively a singleton. We have a context called UserContext in our code base that has the following properties:

1. The UserContext provider wraps every rendered component in a page.
2. At any point in time, there is one UserContext and its contents are the same across all components.
3. Any change in a UserContext is reflected across every use of it.

4. The contents of the current UserContext can be accessed from any component currently on the page, regardless of what level it is in without depending on what was passed from a parent component.

The UserContext holds the information of the currently logged in user, or empty values when a user is not logged in.