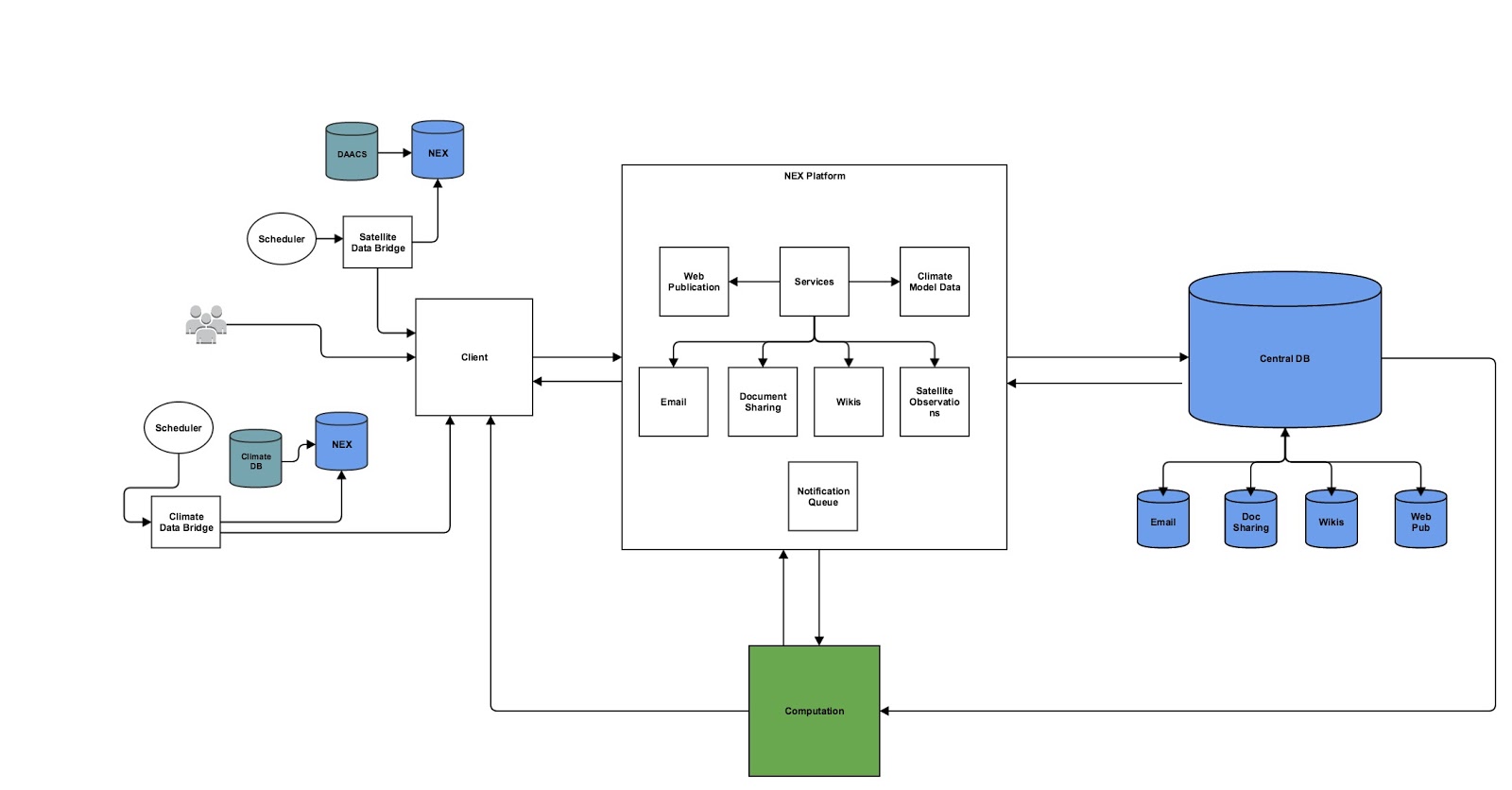
**System design**

****

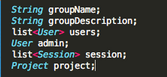
The initial design was to have services for each of the different components of Open NEX. There were data bridges that sent data from NEX to our databases through the client. All the services were on a platform, except for the computation service which interacted with the platform and was its own stand alone service. There were different databases for different purposes. Overtime we revised the granularity of various components and services until we reached a desired system architecture that supported Open NEX’s collaboration platform along with its various requirements and took advantage of the nature of service oriented architecture.

**Components**

Our system consisted of five essential components. These components included Users, Groups, Sessions, Projects and Tasks. Users are the users of the system. Groups are interest groups that are created by users so that a particular topic or service that is used commonly amongst the users can be discussed and worked on within the group. Sessions are created by users or groups of users when an urgent matter needs to be discussed and the state of that discussion needs to be stored for the future. Projects are created for project teams that have an administrator that assigns tasks to other users and this can also be used in a corporate setting. Tasks are operations that need to be taken care by a particular user. Groups can consist of many users so there is a ManyToMany relationship between these components. Groups can have many sessions, but the general idea was to have one group per session so that the session can be more focused. Groups can have many projects associated with it, but a project would mostly want to have one group associated with it. There can be many users in groups, sessions and projects. There can be many tasks associated with users, and multiple users can also be assigned to one task. This was done with the consideration that there could be people pair programming or working on the same task and in those situations, both the people would be assigned to one task.

For the sake of having a simple POC, we have simplified the relationships between the objects. The general object structure of the four components associated with a user are as follows:

Groups:



Sessions:

creen Shot 2015-05-08 at 10.17.59 PM.png

Projects:

creen Shot 2015-05-08 at 10.18.29 PM.png

Tasks:

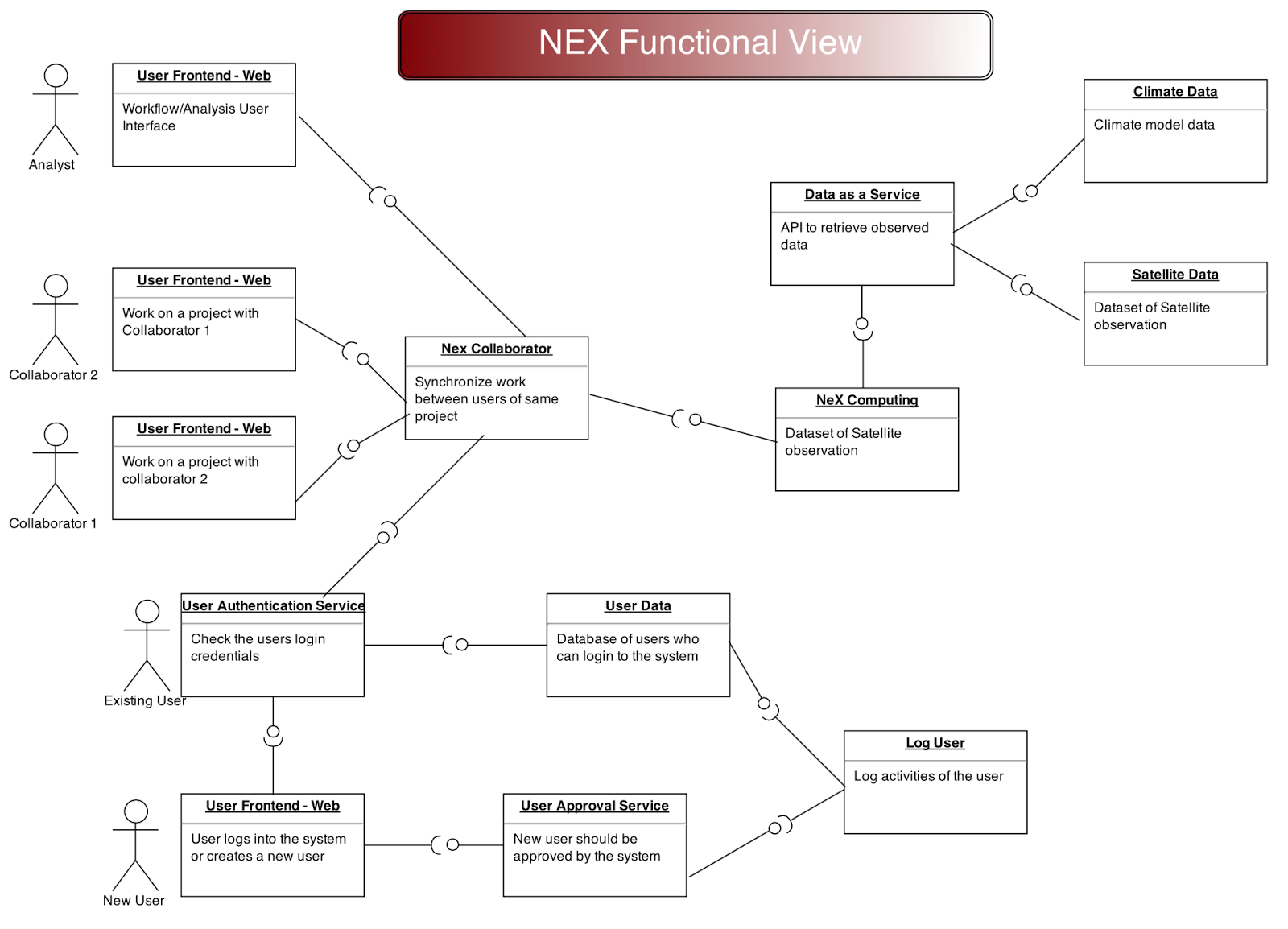
creen Shot 2015-05-08 at 10.18.20 PM.png

**View Points**

To visualize the system design, we created different views of the system from various view points, namely the functional view, the information view, and the deployment view.

**The Functional View**

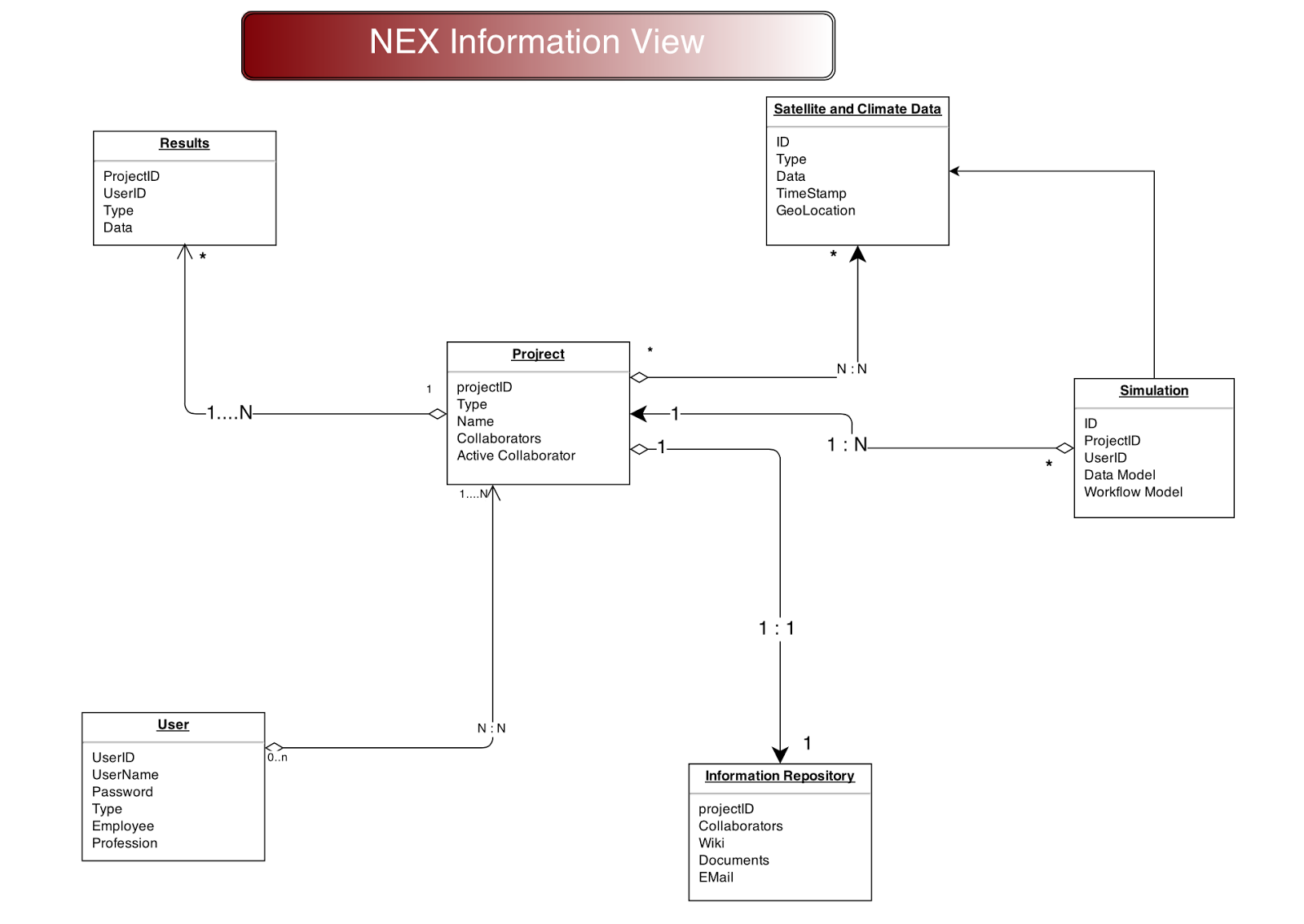
In the functional view, the NEX collaborator is the central piece of the architecture which interacts with all the users of the system and fosters social networking. It is also in turn responsible for authentication of users and the approval workflow. Project, group, sessions are all tightly coupled together and are also handled by the NEX collaborator as shown in Figure 1.1.

****

**Figure 1.1**

**The Information View**

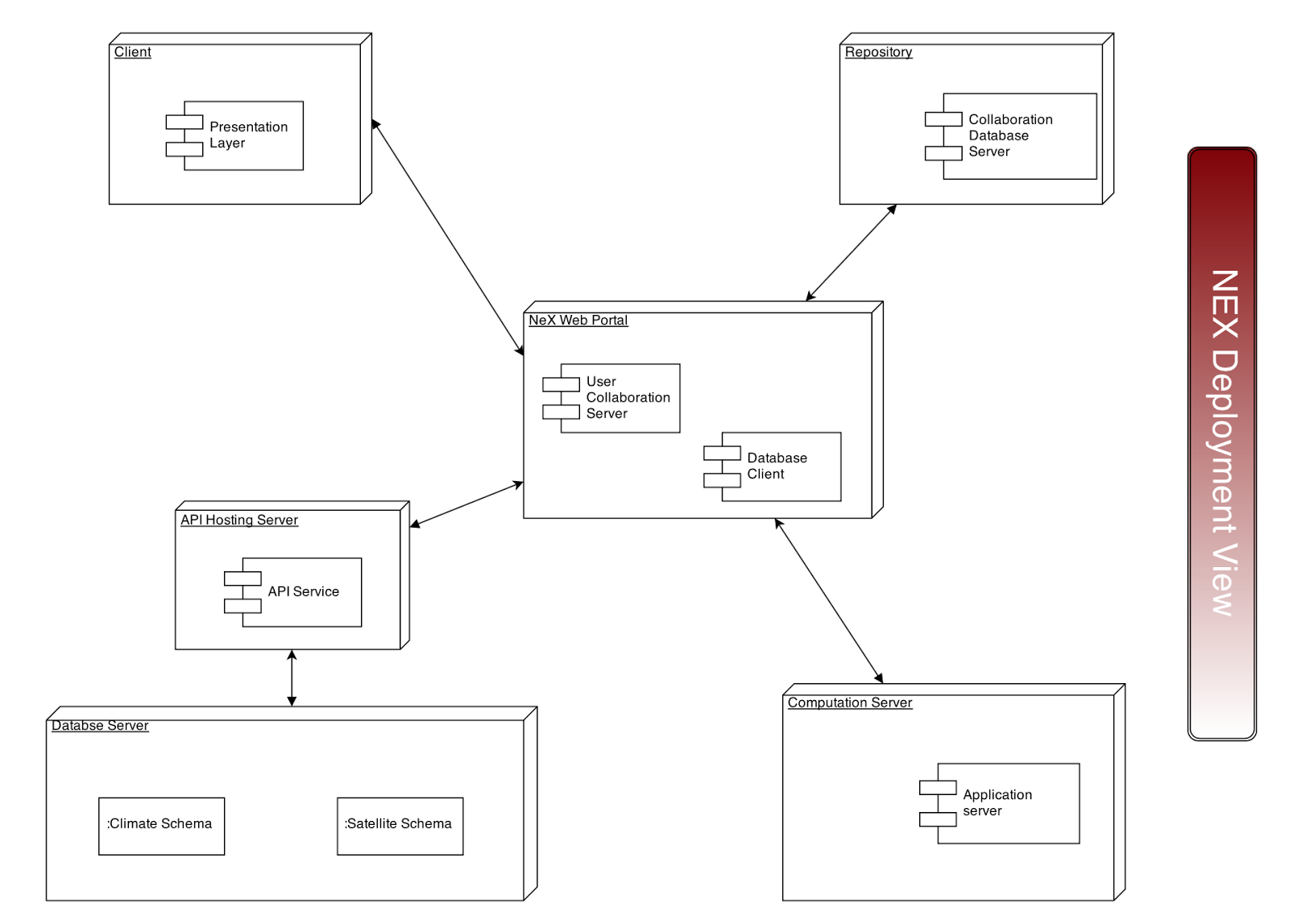
The information view below in figure 1.2 represents the flow of data inside the system and the relationships among the different objects. The relationships between the objects are many to many. The below representation was refined continuously throughout the project and although the basic structure and relationships remained the same, the attributes of the states were modified as necessary.



**Figure 1.2**

**The Deployment View**

After the above views were designed, we analyzed them to come up with a deployment view as shown below in figure 1.3. The constraints of the deployment were based off of a web service enabled architecture (Service Oriented Architecture). All the interactions with the backend servers are done via API calls through various services.



**Figure 1.3**

**System implementation**

Once all the views and high level implementation was done by the team, the next step was to implement the system and get a working prototype up and running. The constraint put on all the teams was the use of play framework which is an [open source](http://en.wikipedia.org/wiki/Open_source) web application framework, written in Scala and Java, which follows the model–view–controller (MVC) architectural pattern**.** Our system components such as Project and Session have models associated with each of them which is the object representation of the actual component. The controllers contain the business logic, and the views display the information by deserializing the json and presenting the information in a easy to view way. Play is heavily inspired by Ruby on Rails and Django and is similar to this family of frameworks. Play uses Java to build web applications in an environment that may be less Java Enterprise Edition-centric. Play uses no Java EE constraints. This can make Play simpler to develop compared to other Java-centric platforms.

To enable interaction between the views and controllers, we used restful APIs. This is in conjunction with the SOA architecture. At the back end, we make use of an h2 in-memory database which is only available when the application is up and running. Although there are more configurations available with the play framework, for simplicity and prototype, we chose to use an in-memory database. Changing the database configuration is very easy with the framework as we just have to change a few lines in the configuration of the framework.

The user-interaction views are HTML pages populated using the scala templates. Scala works by passing data as objects from the JAVA application to the scala templates. All the object methods are available in the templates and can be accessed using simple getter methods. As is common today we used *git* for version control.

The Play framework makes use of JAVA annotations to create database models, define unique keys for database tables, and define relationships between database objects. The database CRUD operations are completely automated by the framework and are refreshed upon any changes to models. Moreover, searching the data in tables by unique keys are defined by a few simple lines of code which are standard for any application developed on the play framework.

The real challenge was similar to learning anything new. It took the team time to ramp-up and get the first user story up and running in system. After that it was relatively smooth. The main bottlenecks were troubleshooting the framework configuration errors. In retrospective, we would have prefered to start developing the application or even a small prototype on play a little earlier during the development cycle.

For source code please refer to the [Building & Running the Project](https://docs.google.com/document/d/1W5XdikNhv__3rP6JsvN5xw1MWPTrgKN6UIUruh1pIX0/edit#heading=h.154r4rfe4jff) section of the report.