**Architecture and Design**

**Introduction**

The architecture of the ticketing system consists of multiple working components. The main components are a Web Portal, a hosting web server, a Web API consisting of controllers and business processing logic, Data Transport Objects for holding and transferring plain data through the wires, Data Access objects for communicating with database providing Object relational mapping (ORM), and a Database management system for managing the data. However, the most important aspect of the architecture will the MVC pattern used for separation of concerns. If we notice the components mentioned above, it somewhat hints that the pattern being used fits the Model View Controller pattern.

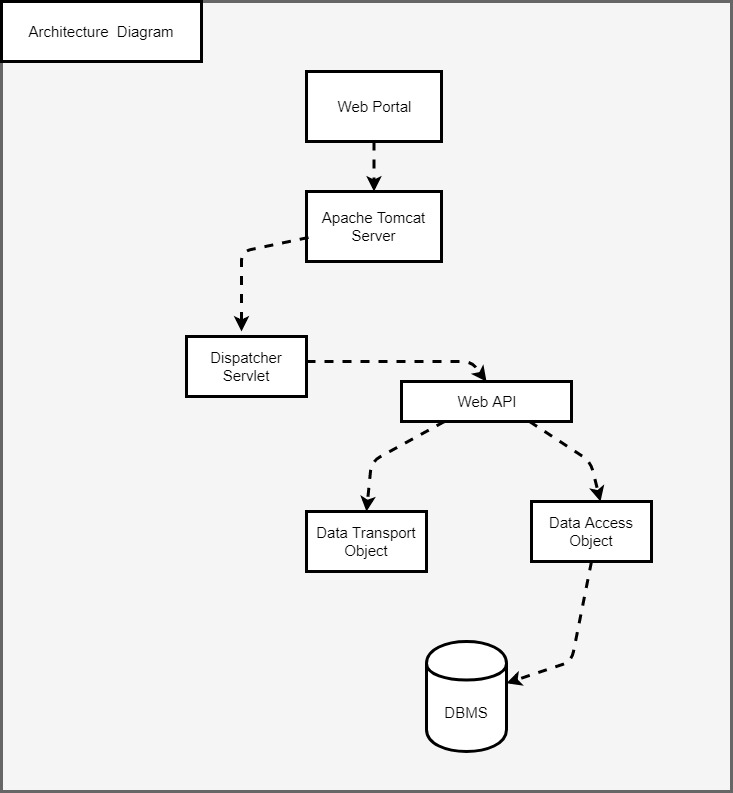


Figure 1: System Architecture Diagram

**Architecture and Design Philosophy**

The system is supposed to serve a main simple purpose of creating and managing tickets. For that matter, the system should be able to provide a Web view for the users to interact. It should also be able to receive requests from the client and process it and perform logic on the incoming data on a backend server. Finally, it must be able to somehow persist the data. Keeping the said overall purpose of the system in mind, the MVC pattern sounds like the most reasonable approach for the development of this system. The Web Portal signifies the View component, the Web API constitutes the controller component while the DTO/DAO provides the Model component of the MVC pattern. The DTO/DAO use the factory pattern which facilitates the instantiation of the objects and also eliminates code redundancy. The DAO factory class would hold accessors for all the DAO objects which will be annotated with special spring annotation called @Autowire. This would allow spring to locate the DAO objects just by calling the accessors through the DAO factory without explicitly instantiating. This tremendously supports the maintainability of the code and the system overall conforming to the non-functional requirements of the system.

Among some of the initial consideration of the architure pattern was the client-server pattern. However, the choice was discarded considering the client-server pattern is more suitable for system which provides multiple services while our system is solely focused on one particular service. Moreover, the MVC pattern on the other hand not only provides separation of concerns but it will also facilitate further extension of the system to incorporate it into a bigger platform. Since, the components are separated from each other, it would be easier to test and debug it while merging it into a bigger system as well as for when it is independent.

**Architecture Views:**

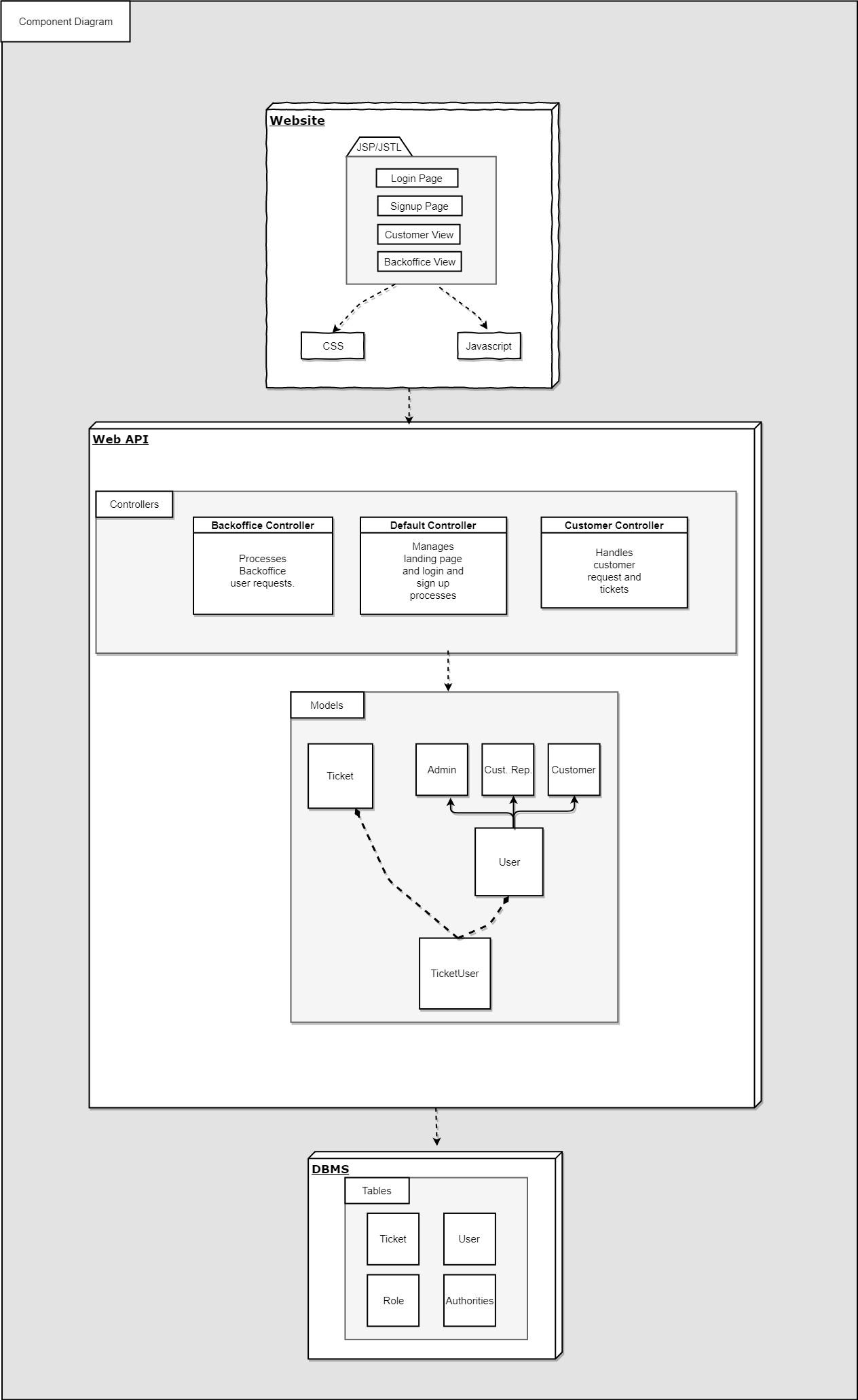


Figure 2: Component Diagram

**Design Models:**

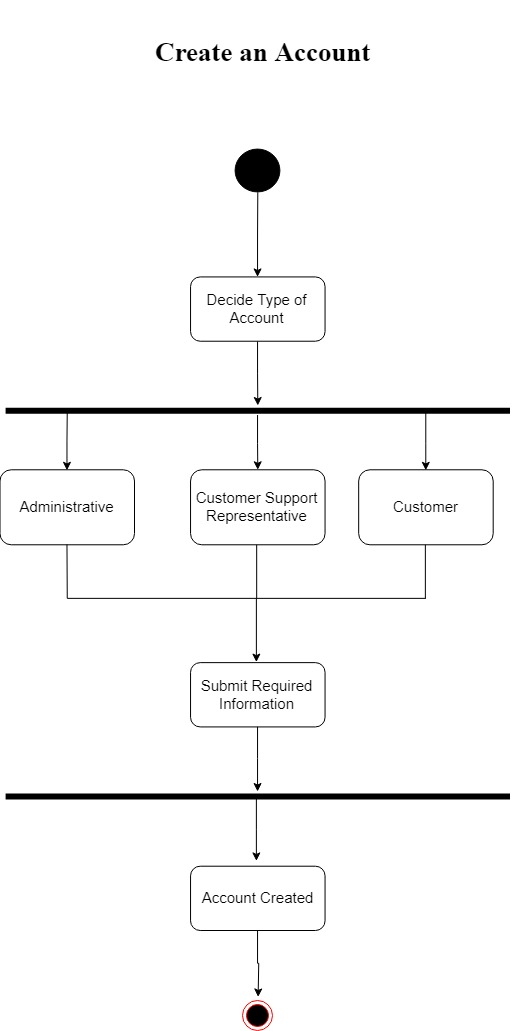
****

Figure 3: Create Account Activity Diagram

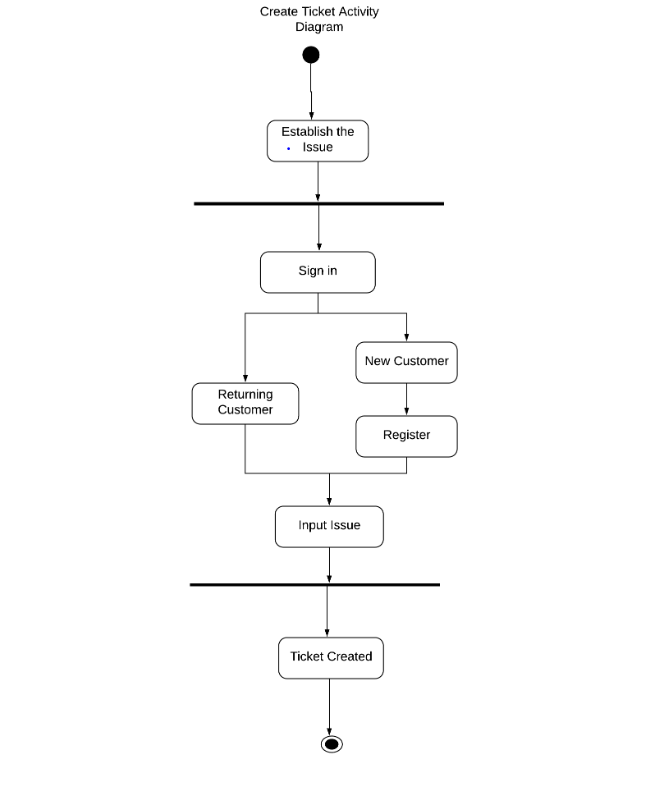


Figure 4: Create Ticket Activity Diagram

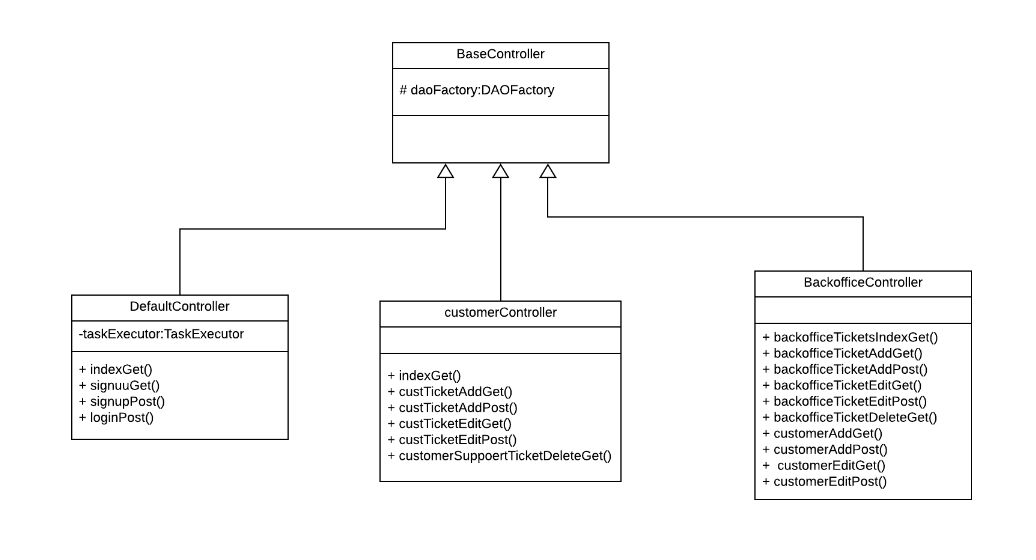


Figure 5: Controllers Class Diagram (Generalization)

Controller classes are the classes that actually dictate the behaviour of the system. These classes contains all kinds of methods to facilitate the dynamic behaviour of the system and accomplish all the functional requirements. Each method in the individual controller classes are mapped to the request url provided to the user on the client side. The most frequently used data structures within our controller classes are ArrayLists and HashMaps. There is no explicit algorithm implemented however, in hashmap the hashing algorithm is implemented implicitly by the library

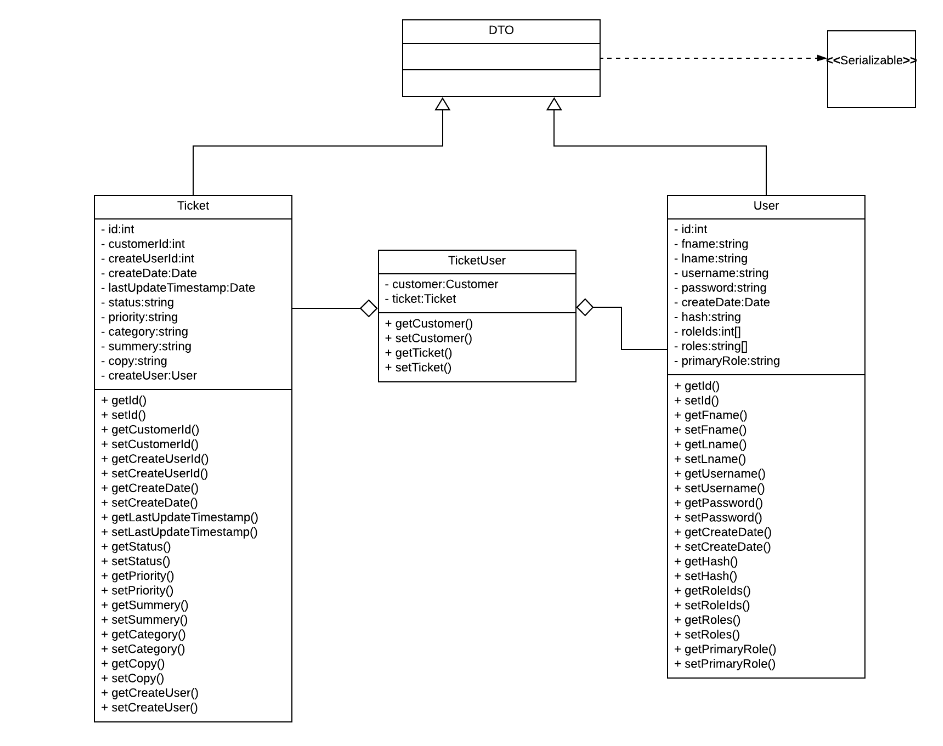


Figure 6: DTO Class Diagram

DTO classes are used for holding the data of the program’s current state such as form data from the user input and also data from the database to be rendered on the html pages. Each business entity has a DTO class which contains its relevant property fields and their accessor methods.

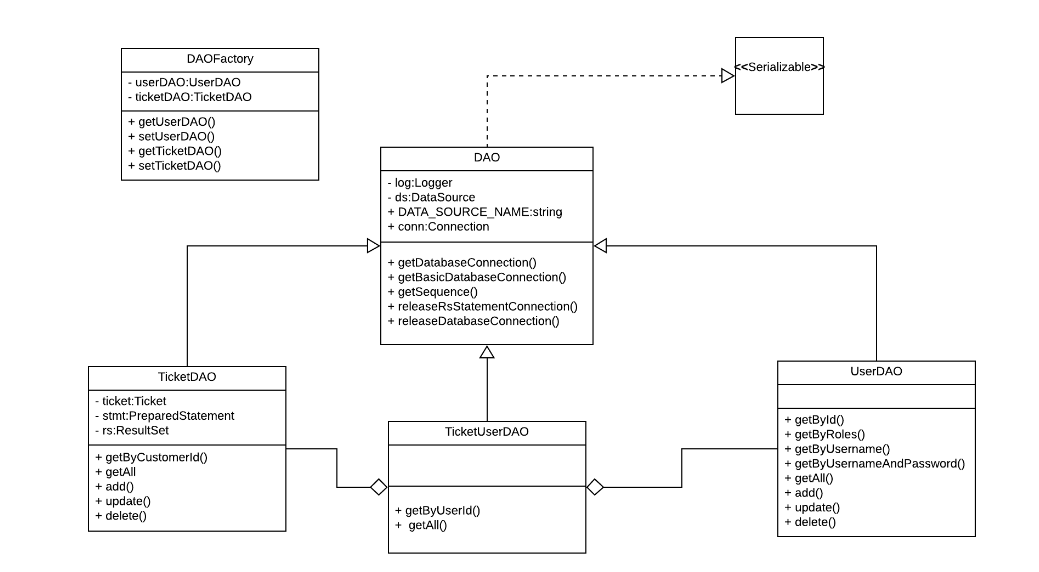


Figure 7: DAO Class Diagram

Each DTO class for any entity has its corresponding Data Access Object (DAO) class. The DAO classes are the use to establish connection to the database and to map the object classes to their respective tables in the Database. The parent DAO class is the one used for establishing the connection. While its subclasses which refer to the business entities actually perform the queries and transactions.

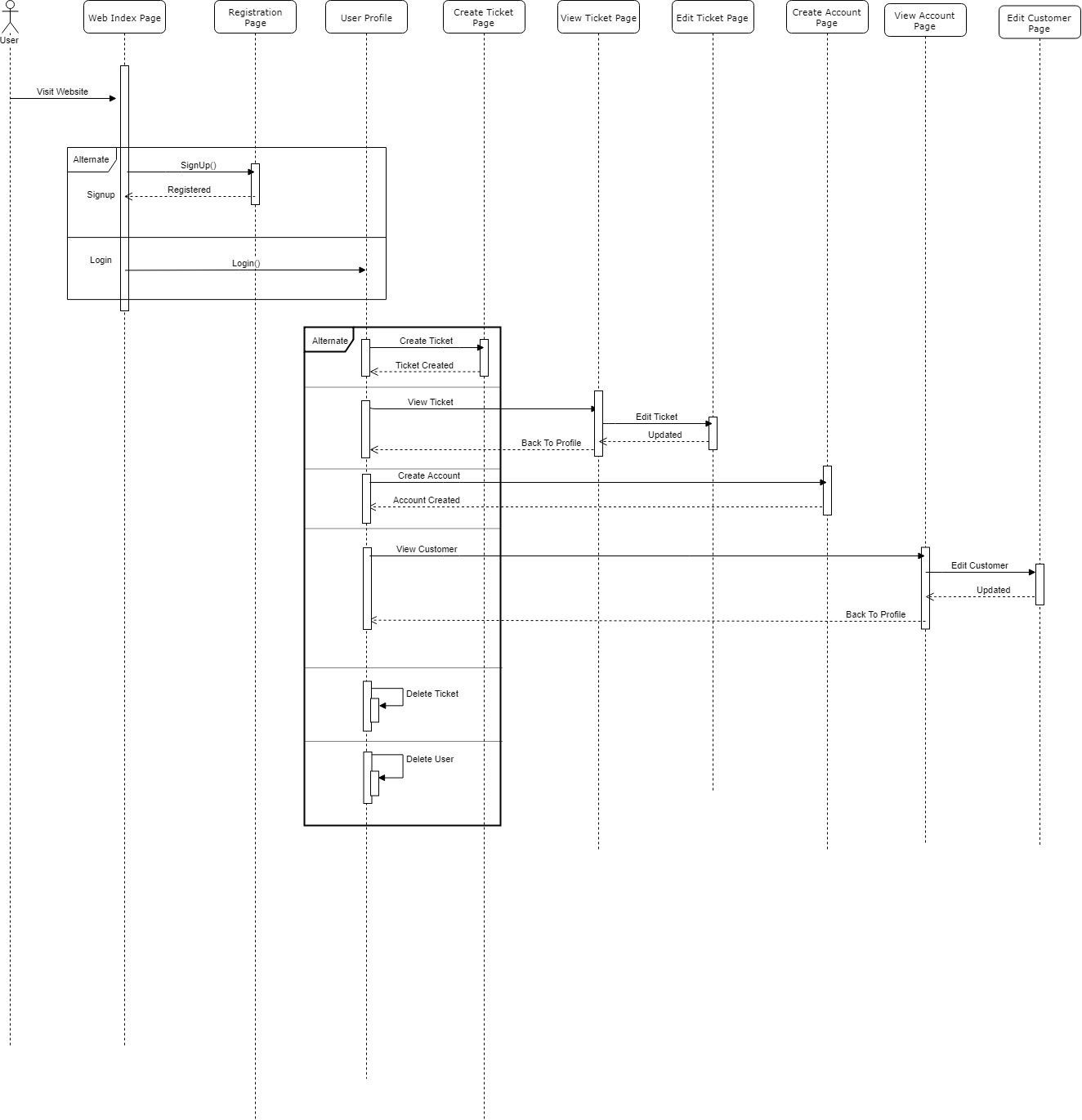


Figure 8: Overall System Sequence Diagram

The above sequence diagram depicts the overall behaviour of the system.

**Initialization files:**

* Pom.xml
* Web.xml

**Configuration Files:**

* Mvc-conf.xml
* applicationContext.xml
* Web-beans-conf.xml
* Security-conf.xml

For proper operation of the system, following tools must be installed:

* Apache Tomcat 8
* Maven