UNIVERSITÉ Concordia UNIVERSITY

SOEN 6441

Advanced Programming Practices

Build 2

Architectural Design Document

Team 20

Konstantin Chemodanov…. 40049285
Farzaneh Jamshidi…..…...…40075234
Saman Safaei………………. 40119399
Ali Molla Mohammadi …….40042597
Negar Ashouri………………40071530

March 2019

Department of Computer Science and Software Engineering

Introduction:

The purpose of this build is to add "Observer Pattern" design pattern to the single-page web application of Online Risk game implementing an MVC architectural design model. We designed the project as web client-server application, where server running backend is implemented using Java language and client interface is using Angular. In order to implement the observer pattern design we used "webSocket" in Springboot and Angular.

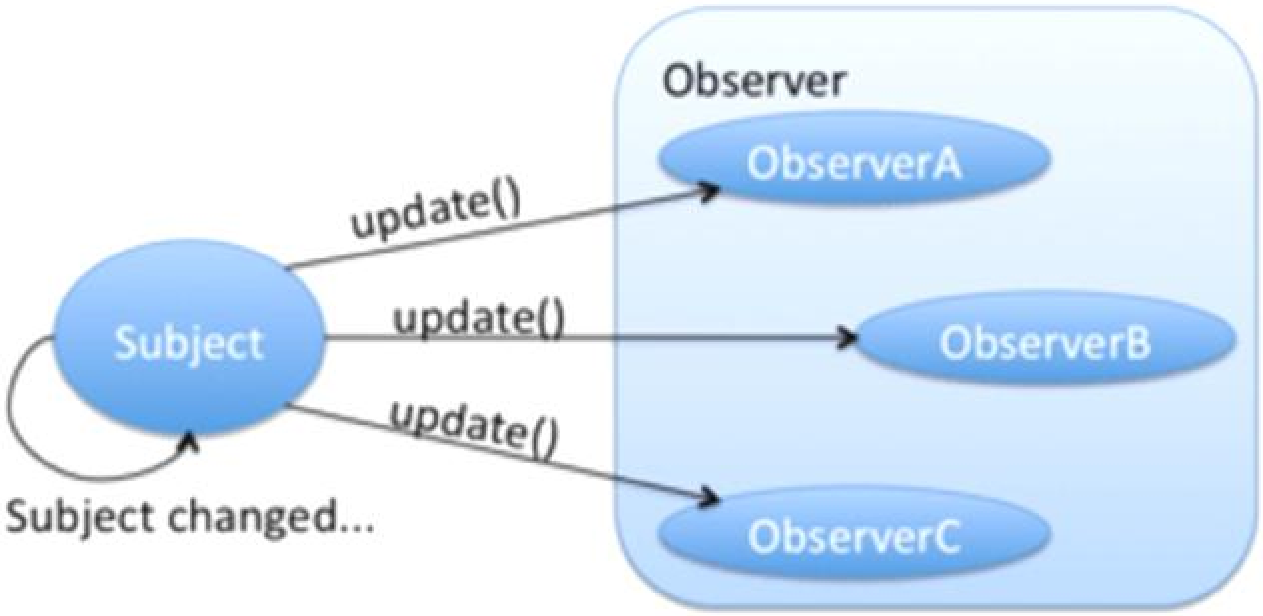
We have followed the extreme programming approach in development of our project:
Pair Programming: After dividing project into modules, two programmers or more worked on the same tasks in order to have better and quicker results and desired functionalities.
Simple Design: In order to avoid faults, software was designed as simple as possible and any extra complexity were removed.
Continuous Integration: Bitbucket is a web-based version control repository for development projects that use Git revision control systems, same as this project. Using Bitbucket made all the programmers on the same branch and eased the Maintenance of concurrent changes and errors detections by automating integrating tasks.
Collective Ownership: Using a repository also made all members able to make changes anywhere in the code anytime.
Testing: Several unit test have been written in order to make sure each unit is functioning as it should.
Coding Standards: Oracle coding standards were accepted by all programmers and was performed in the whole project. Like naming conventions, file organization, commenting conventions and etc. Following these standards made the code more readable and understandable for everyone which leaded us to a more maintainable and sustainable code that will be helpful in our further builds.
Refactoring: After receiving feedbacks of the first build and requirements for the second build, deficiencies were found and refactoring has been applied as were needed, like in design patterns and player model.

Architectural Design:

Observer pattern

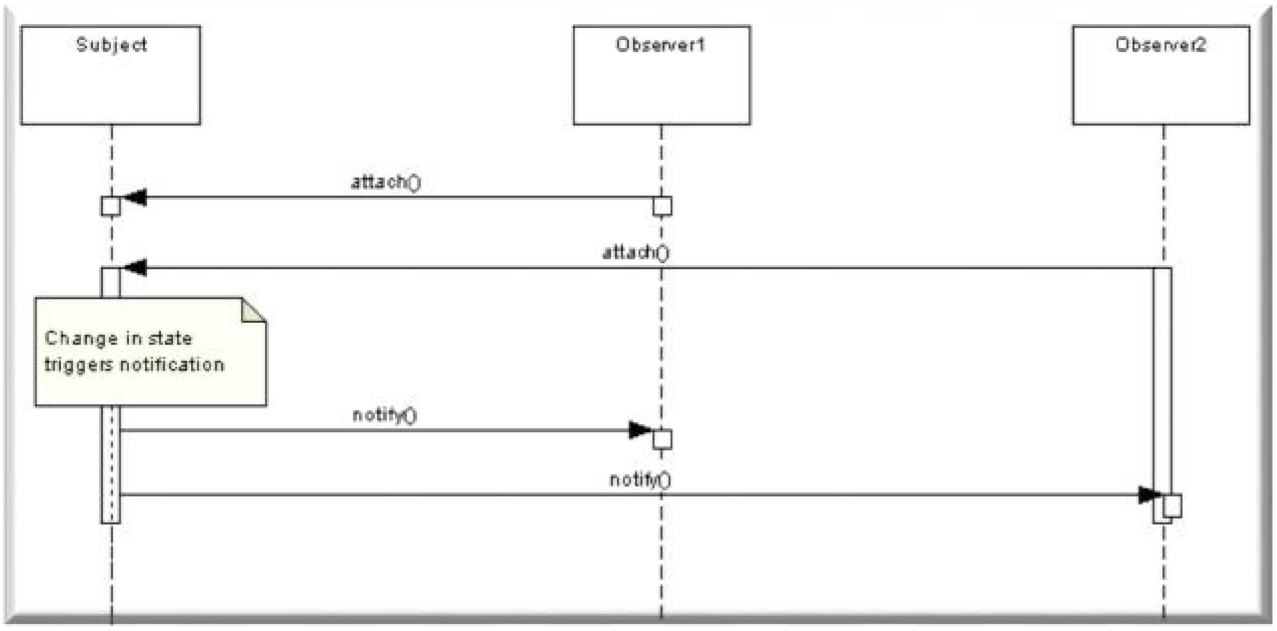
Observer pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. It is also referred to as the publish-subscribe pattern.
In observer pattern, there are many observers (subscriber objects) that are observing a particular subject (publisher object). Observers register themselves to a subject to get a notification when there is a change made inside that subject.
An observer object can register or unregister from subject at any point of time. It helps is making the objects loosely coupled.

When to use observer design pattern:
As described above, when you have a design a system where multiple entities are interested in any possible update to some particular second entity object, we can use the observer pattern.

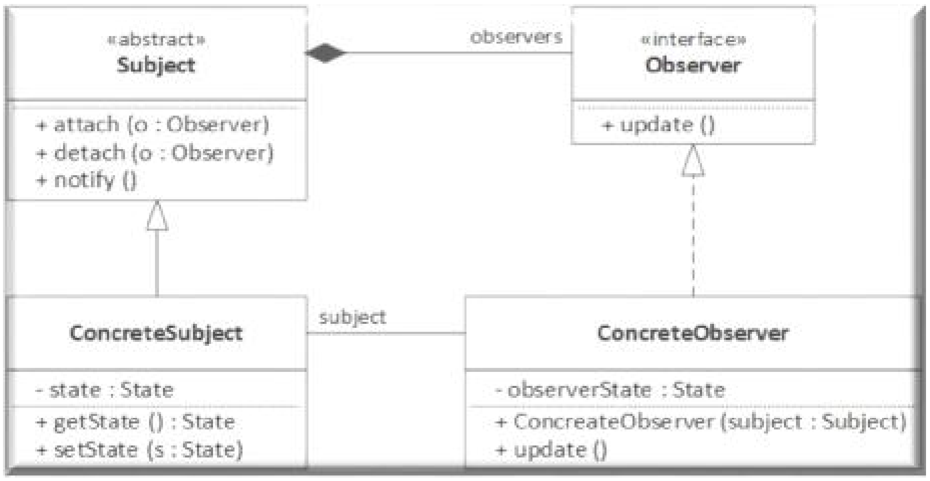


The flow is very simple to understand. Application creates the concrete subject object. All concrete observers register themselves to be notified for any further update in the state of subject.

As soon as the state of subject changes, subject notifies all the registered observers and the observers can access the updated state and act accordingly.



Observer Pattern Architecture

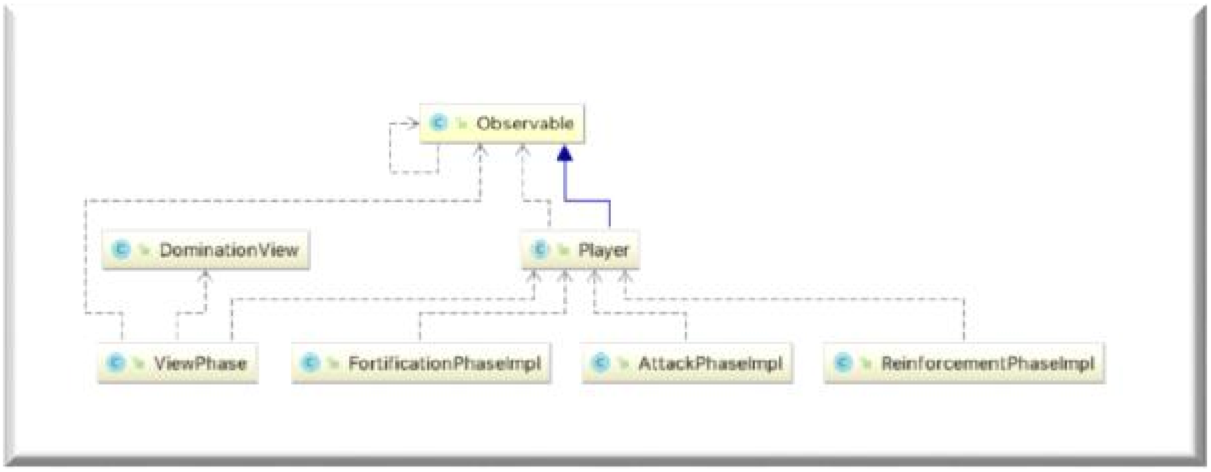


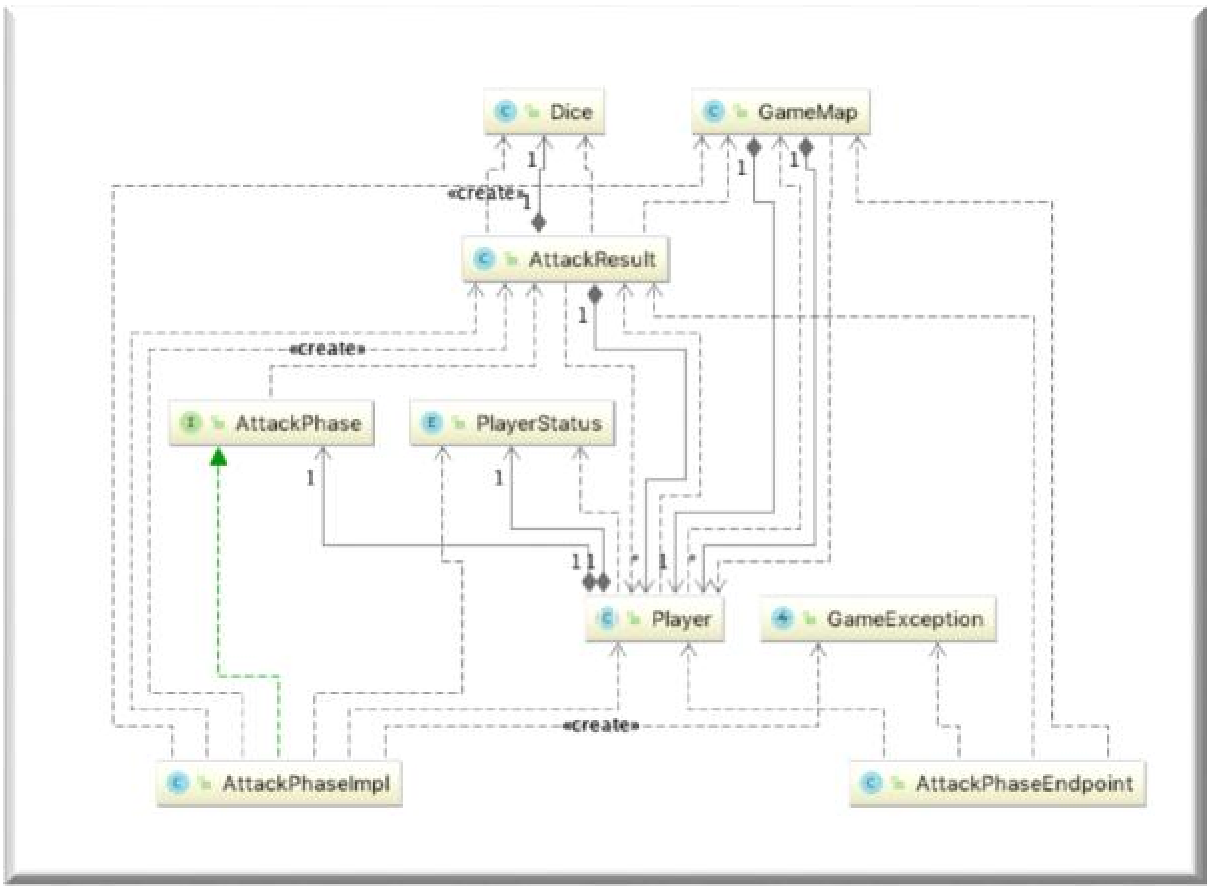
Design participants

The observer pattern has four participants.

- Subject – interface or abstract class defining the operations for attaching and de-attaching observers to the subject.
- ConcreteSubject – concrete Subject class. It maintain the state of the object and when a change in the state occurs it notifies the attached Observers.
- Observer – interface or abstract class defining the operations to be used to notify this object.
- ConcreteObserver – concrete Observer implementations. [1]

Following diagrams are showing how we have implemented observer pattern in this project:



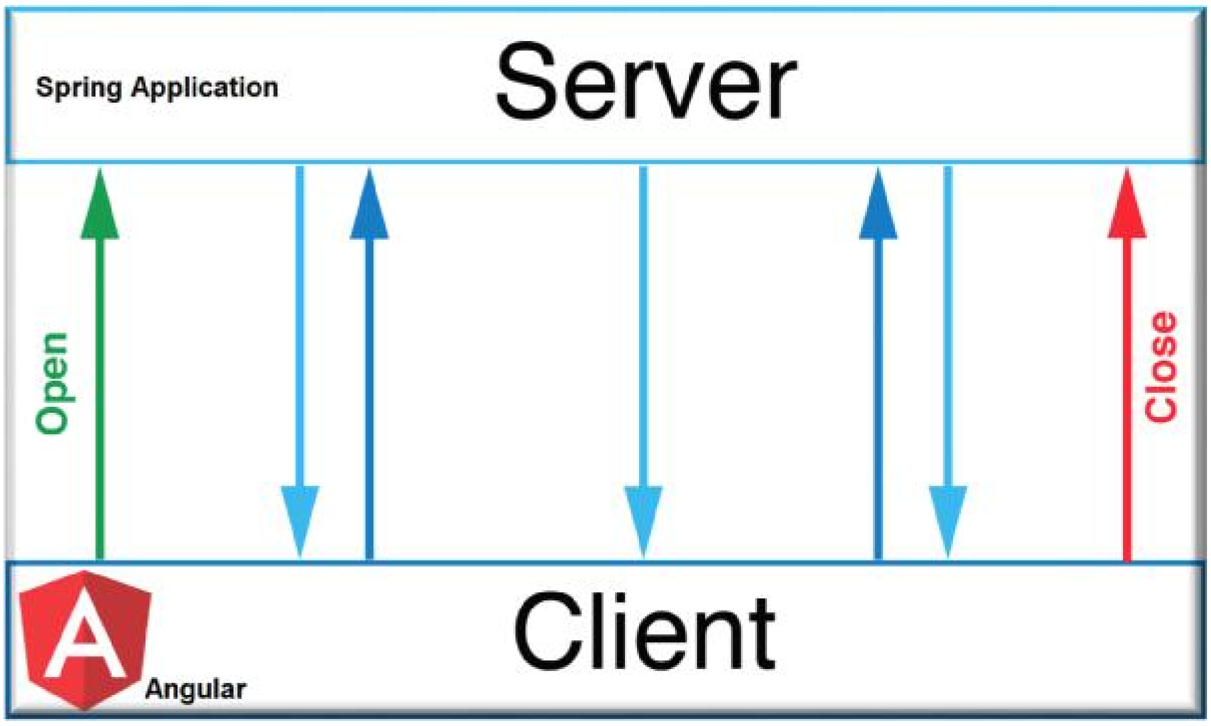


What is a Real Time Application?

According this Wikipedia definition, a real-time application allows information to be received as soon as it is published, rather than requiring a source to be checked periodically for updates. Therefore, this kind of app should give a feeling to users that events and actions happen immediately.

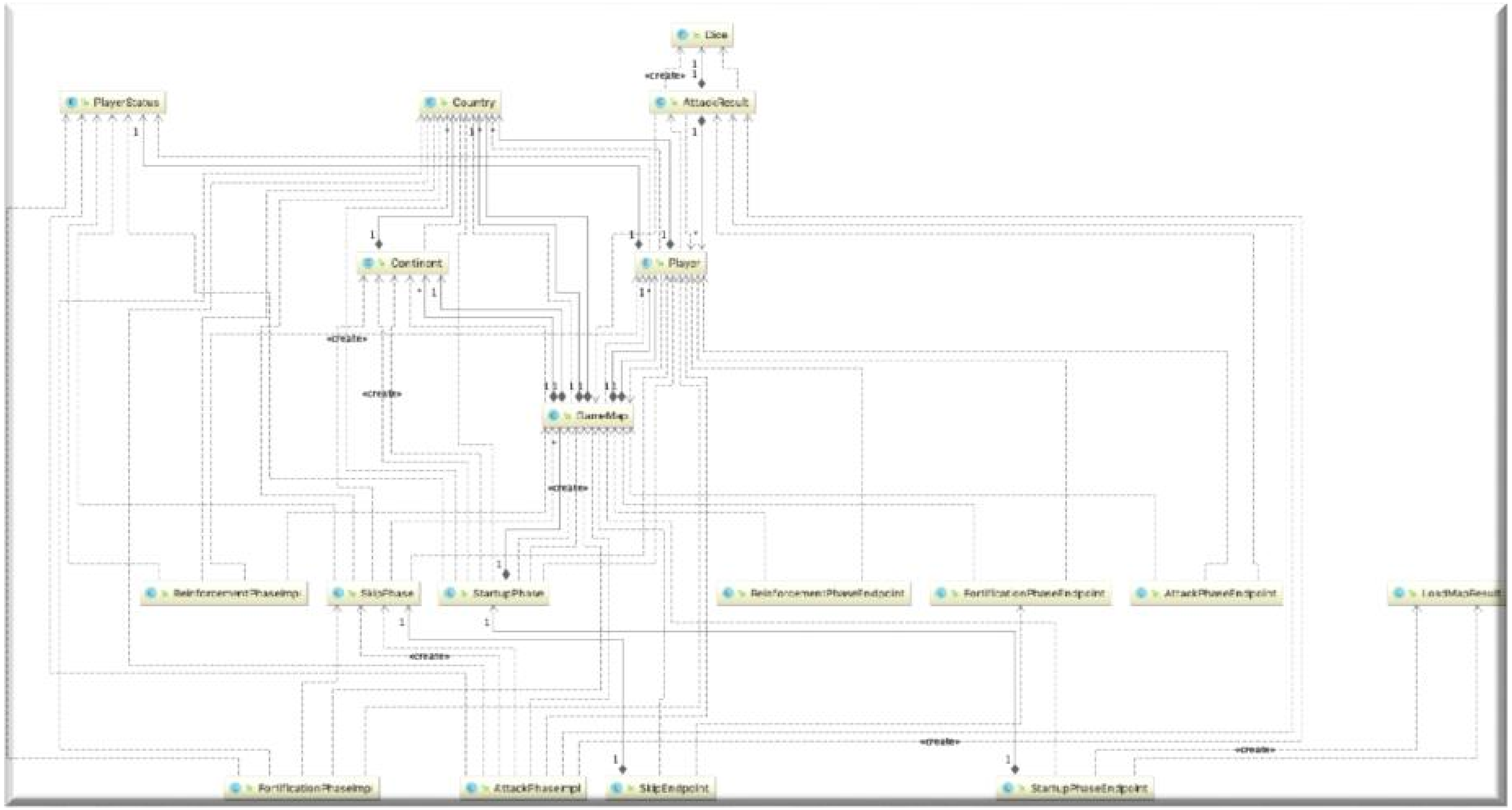
WebSockets

WebSockets is a protocol that provides a bi-directional communication channel. This means that a browser and web server can maintain real-time communications, sending messages back and forth while the connection is open. [2]



The WebSocket protocol provides new capability for web applications: full-duplex, two-way communication. So in the system where the client and server need to exchange data at high frequency and with low latency, WebSocket is the best solution. [3]
WebSocket is a protocol which enables communication between the server and the browser. It has an advantage over RESTful because communications are both bi-directional and real-time. This allows for the server to notify the client at any time instead of the client polling on a regular interval for updates. [4]
We have used webSocket in our project in both backend and frontend in order to apply the observer pattern for player view phase, players world domination view phase and card exchange view.

Following diagram gives you an overall view of the project design.



References

[1] [Online]. Available: Gupta, L. (n.d.). Observer Design Pattern - Observer Pattern in Java - HowToDoInJava. [online] HowToDoInJava. Available at: https://howtodoinjava.com/design-patterns/behavioral/observer-design-pattern/ .

[2] [Online]. Available: Real Time Apps with TypeScript: Integrating Web Sockets, Node & Angular. [online] Available at: https://medium.com/dailyjs/real-time-apps-with-typescript-integrating-web-sockets-node-angular-e2b57cbd1ec1 .

[3] [Online]. Available: grokonez. (n.d.). Angular 6 WebSocket with Spring Boot WebSocket Server | SockJS + STOMP - grokonez. [online] Available at: https://grokonez.com/frontend/angular/angular-6/angular-6-websocket-example-with-spring-boot-websocket-server-sockjs-stomp .

[4] [Online]. Available: [online] Available at: https://www.nexmo.com/blog/2018/10/08/create-websocket-server-spring-boot-dr/ .