Innopolis University

Thesis

Architecture design of the event management application

Author:
Ilgiz Zamaleev

Supervisor:
Victor Rivera

December 7 2017

Abstract

Software architecture always was one of the most important part in information technologies. Initial design of one or the other application directly determines its further development. Even the smallest mistake at the initial stage can lead to large costs in the future. The best way to avoid mistakes is to use the experience of previous developments, such as design patterns or using of modern development tools.

This work focuses on the implementation of modern approaches in the software architecture development of the event management application that called Hikester.

1 Introduction

With the development of the world's first computer was developed the first programming languages. Every year these languages were perfected, becoming more saturated and full. Huge enterprises and companies were created that used complex solutions in the information technology field. From day to day, humanity has found a new and new application for computers and software. With the increasing importance of information technology in everyday life, there was a realization that poorly developed software can cause huge costs of money and time. Accordingly, there was a need for some standards and programming templates that would allow the development of stable, adaptable and reliable software. To this end, a lot of research was conducted in the IT domain, new methods for optimizing the software architecture were searched for.

The main idea of this work is revealing of the most effective development methods for building an architecture of the event management (Hikester) project. Analysis and determination of the most optimal models for building an application architecture on different platforms for the most efficient organization and design of the final architecture. Application of innovative tools in modern development (Firebase, elasticsearch, modern design patterns).

2 Literature Review

From the usual electric toothbrush and electric clock to the largest computers and aircraft, at every turn we are surrounded by information technology creations and each of these devices contains its own software. For the correct working of each gadget, of each electrical appliance, of each application program, should be written the corresponding correct, stable and supported software. To achieve this aim it is necessary to thoroughly think about the entire architecture[1] of the future program, because it will directly determine the development and application of the given software.

The software architecture is primarily a high-level abstraction of the interaction of all components of the application. It allows at the initial stages of development to think about the future structure of the whole project, and to determine what and how it will work. With the right choice of architecture, it is always possible to divide the development into different parts. That separation will provide development in parallel way in a single project. Also a well-constructed software architecture can be used in other projects, thereby using the experience of the previous development [2, 3].

In the field of application architecture development has been done a lot of research, for example from [4, 5] you can clearly deduce what design patterns are, how and for what purpose they are used. Briefly, design pattern is a reusable software architecture solution to the commonly occurring design problem.

One of the most popular pattern in Web is MVC. MVC pattern, that implies the division of the project into 3 parts: Model, View, Controller. The model is the central component of the pattern. It expresses the application's behavior in terms of the problem domain, independent of the user interface. Model directly manages the data, logic and rules of the application. View is responsible for representation of information (model) for user. Controller accepts input(some user interaction) and converts it to commands for the model or view.

MVC design pattern can be applied for the event management project (Hikester).

Another modern way to develop project is following the principles of liquid architecture. This structure which satisfies the requirements of the manifesto described in the article [6]. This article about liquid architecture and describes that in the near future most applications will have a great need

for synchronous data in the context of one application when one user will use different devices for running some app.

If we add the conditions of the liquid architecture to the MVC, then we will add the condition for Model: mobile and web Models will be used a single storage. So all user's interacts with the system will done through a single repository. Such storage could be real-time database - Firebase [7]. Which ensures that synchronization of the full state of application and the data is displayed correctly when using different devices. This allows to users "be able to effortlessly roam between all the computing devices that they have"[6]. More information about how to built liquid software can be found in [8].

Another example of a design pattern is Observer. The Observer pattern is useful when you need to provide some means for dependent objects to adjust their states according to the state of an object they depend on. In this situation, dependent objects are called Observers, and an independent object Observable or Subject. Observable usually contains a pool of Observers that is possible to modify by adding or deleting Observers. Observers in this pool can be notified whenever there is some meaningful change in the Subject, using a special method. Observers, on the other hand, have means for updating their state by getting the state of the Subjects [5]. We can apply it to user notification system in Hikester project, which will notify users about some event creation.

Asynchronous method invocation also is design pattern which described in [9]. That pattern is used for remote call some site. When some request was sent to a server, program doesn't wait a response, just continues to work. This is due to that with the request was sent some callback address, that allow to avoid time wasting for waiting a response.

The servers which use the asynchronous requests could be scalable in more easily way, in contrast to the typical implementation based on the threaded model. For example, in [10] described Node.js. It is one of the newest platforms (introduced in may 2009) based on the V8 engine (translating JavaScript into machine code) that turns JavaScript from a highly specialized language into a general-purpose language. Also node.js and its novel implementation [11] are very good at coping with small data (e.g. sensing data).

In Hikester the server part can be run on node.js. Because most of Hikester application's data, such as geolocation, message, recommendations will small and also opportunity for easy scalability will be good reason to

choose this tool.

Using new software approach in web's front-end - Single page application, described in [12, 13]. That solution is based on the concept of state. Concept of state is taken from state machine. Each element of web page has its own state - set of variable in certain moment of time. And changing state of some web page element allows to change some part of web page without reloading. Based on this paradigm were created several front-end frameworks, one of them is a reactJS [14, 15].

To support liquid architecture, another important criterion for constructing the front-end is responsive design. About this is written in the article [16]. The main concept of responsive design is that the content of web page must be correctly represented on different devices irrespective of the screen size. The last two concepts will be good software solution for front-end part of Hikester.

The VIPER [17] is one of the mobile design pattern. This pattern implies a division of the project into 5 layers: View, Interactor, Presenter, Entity, and Routing. View is responsible for data representation, Interactor layer for business logic, Presenter drives the UI and sends request from user to Interactor, Entity is data layer and Routing is responsible, as the name implies, for routing between other layers.

That pattern also can be used in Hikester as the main design pattern on which will build mobile applications.

2.1 Summary

There are a lot of solutions in the field of software development, and all of them have their advantages and disadvantages. Based on this work, we can conclude that the software solution of event management application Hikester will be implemented in next way: the back-end will be running on Node.js; front-end on reactJS with responsive design; for the repository will be chosen a Firebase real-time database; mobile application's design must be based on VIPER pattern. For the development of each component were chosen a special approach of software design, taking into account the specifics of the whole project. That allows to develop application in quick and high-quality way, as well as simplification of support in the future. The work of the application will be based on the interaction of the user, both with the web version, and with mobile applications, through the Firebase platform. All information will be stored right there, in the Firebase's real time data base. Also,

any user behavior (registration, creation of an event, participation, refusal of participation, location changes, search) will be tracked by the back-end. The front-end and mobile sides will implement the most simple, convenient and user-friendly interface for the user. One of the strengths of this software solution is that in the future it will contribute to the rapid development of the project as a whole.

3 Design

This chapter focuses on implementation of modern approaches in the software architecture development. Revealing of the most effective development methods for building an architecture of the event management (Hikester) project. Analysis and determination of the most optimal models for building an application architecture on different platforms for the most efficient organization and design of the final architecture. Also there is described application of innovative tools in modern development such as: Firebase, elasticsearch, modern design patterns etc. One of the most important thing presented in that chapter is description of the architecture and component interaction of Hikester application.

3.1 Project requirements

To make event management application more convenient and more user-friendly for people there should be applied the approaches that was mentioned in chapter 1:

- *Real time application* - that approach assume all data updates will be shown in real time. F.e. if some person create event, that event will be immediately shown on the screen of all other users that looking in the same location where event was create.

- *Single page application (SPA)* - application which built by that approach should avoid interruption of the user experience between successive pages. That represent the application more like a desktop application. User's browser just once load all necessary code. And all users's interactions with single page application will dynamically change just some part of application without any reloading at any point in the process, nor does control transfer to another page.

- *Cross-platform application* - the application should work correctly on various types of devices such as a PC, tablet and phone. The number of users will directly proportional to the coverage of devices for which this application will be implemented.

3.2 Architecture

3.2.1 Main components

For achieving that goal needs to develop application with user-friendly interface. That application shouldn't restrict users in choosing the device through which they would like to use the Hikester application. This was the reason that the main software design components of Hikester application consists of the following parts:

- *Web interface*
- *Mobile applications (Android & IOS)*
- *Real-time Firebase database[7]*
- *Back-end*

That main components we can divide into 3 groups:

- *The Web and mobile applications - responsible for representation of data for user. Also that components will send to back-end some user interactions.*
- *The back-end - responsible for data control and process the user interaction.*
- *Real-time Firebase database - represent data.*

That was the reason to choose MVC (Model View Controller) pattern as the main design pattern for project in general. That is one of the most popular pattern in modern web development.[18]

In context of that project Web and mobile applications show to users all information which was received from model (real-rime Firebase database). That two project components corresponds to View part of MVC. The back-end handle all users interactions that was sent from View and change the data in accordance with the action which was done by user. It represent Controller in terms of MVC pattern. The real-time Firebase database directly manages the data. Also it is responsible for logic and rules of the writing and deleting the data independent of the user interface. That component is the Model in MVC pattern.

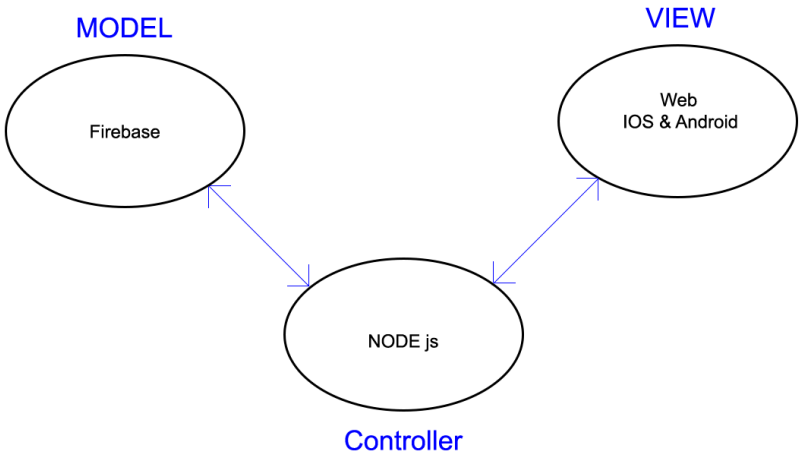

Figure 1: Project components interaction.

Figure 1 illustrate project components interaction.

This structure partially satisfies the requirements of the liquid architecture manifesto. The liquid architecture relate to applications that have several implementations for example it has web and mobile versions. Also that approach assume that the application should have a data synchronization in the context of one application especially when one user will use several devices and as a result several implementations of that application. And each of that implementations should represent user's data in a correct way. It means the date should be the same, but it can be shown in a different ways. All of this requirements are necessary to make the application more convenient for the end-user.

In the Hikester application, all user information and data stored in single repository, namely the Firebase real-time database, which ensures that synchronization of the full state of application and the data is displayed correctly if the user will need to use several devices to use the application. This allows to users "be able to effortlessly roam between all the computing devices that they have"[6].

Another approach that was used to support liquid architecture - responsive web design (RWD). RWD need to render well all web page data on a

different devices regardless of the screen size and type of device. The main idea for the responsive web design was in creating several views for one web page. These web page views depended on the current screen size of the device from which the page display request was came. And the modern stylistics of the web page - Cascading Style Sheets (css) allows you to achieve RWD using a minimum of effort for this.

3.2.2 Auxiliary components

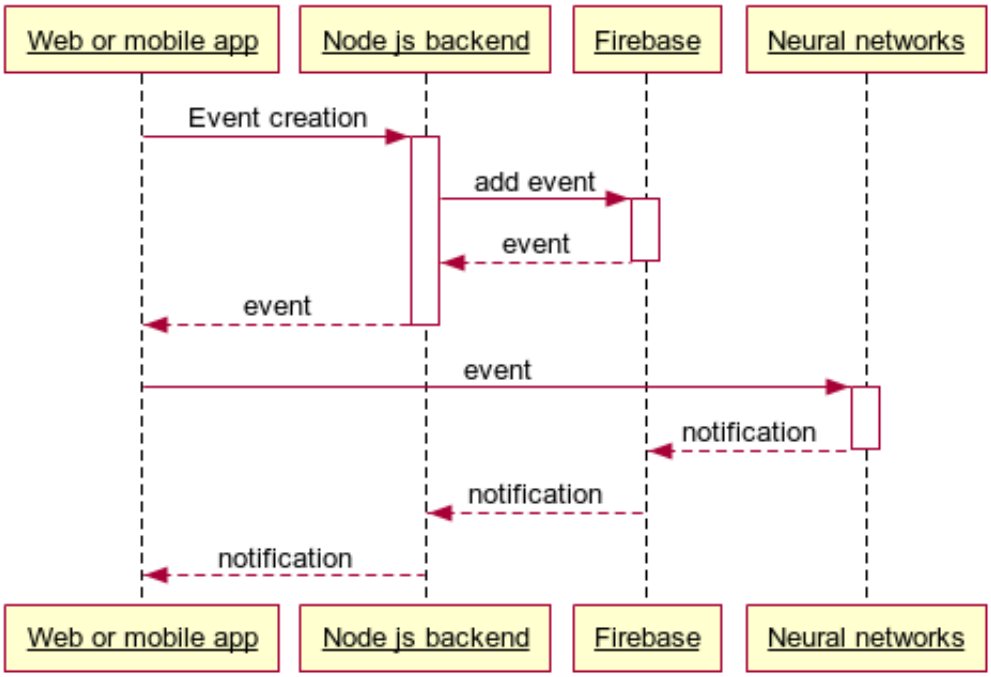

Figure 2: Project modulus interaction sequence diagram.

For more comfortable work with main components and for achieving additional functionalities such as recommendations for users to join for some events and spam recognition system Hikester project also has separate auxiliary services:

- *Elasticsearch*
- *Geofire*

- *Python neural network*

In general, all modulus interactions take place according to following schema:

All user interaction are handled by back-end. After processing the user's input back-end send request to Firebase database. Firebase manage data and send response for back-end. The back-end in its turn send it to View (Web or mobile application). After that manipulation the neural networks is notified about some changes. And after processing data neural networks interact with real-time Firebase database. Then it send it to back-end and then user get some notifications, which was sent by neural network. The figure 2 shows example of that interactions.

3.2.3 Summary

The topic that was described in this chapter is design of the event management application. The main idea was to show what are the main components of this system and how they interact with each other. For the development of each component was used a special approach, taking into account the specifics of the whole project, which allows for quick and high-quality development, as well as simplification of support in the future.

4 Implementation

4.1 Back-end

The back-end of project implemented by using "Node.js". In order to provide a convenient way to interact with front-end and mobile applications the back-end was implemented as an API with Express framework. The main job of back-end is to communicate with front-end and mobile applications via JSON requests.

Node.js is one of the newest platforms based on the V8 engine. That engine translate JavaScript into machine code that promotes to turns JavaScript from a highly specialized language into a general-purpose language. Due to the fact that Node.js use the asynchronous requests and the most part of that platform is implemented in languages C and C++ Node.js provides quite a high speed for input/output (i/o) operations. Also each new connection requires a small memory area in the heap. All this benefits allow to scale the application which based on the Node.js in more easily way, in contrast to the typical implementation based on the threaded model.

However, the node is not doing so well with processing large static files [19], but this problem is becoming less noticeable with the release of new versions. Although in that project case, Hikester application works only with small data, such as geolocation, message, recommendations, but in large amount. For that kind of data the node.js and its novel implementation such as node-DPWS [11] will be one of the best solutions from the range of modern existing software development platforms. In addition the node js server can be integrated in firebase platform as cloud functions that will be covered in firebase subsection. That integration allows to scale application in the easiest way. In that case all computing power will depend only on the payments that will be paid to the firebase platform.

4.2 Firebase

Firebase is a platform for development mobile and web application. Real-time nosql database is one of the main service of that platform through which all components of the event management application will interact. The service provides an API that allows application data to be synchronized across clients and stored in Firebase database. There is enable integration with Android, iOS, JavaScript, Java, Objective-C, Swift, React and Node.js

applications. All of that components are used in Hikester application. The real-time database can be secured by using the special server-side-enforce security rules that provided by firebase.

All requests to firebase real-time database for add or extract data are executed asynchronously. Example of such request illustrated in figure 3. There is shown request from back-end JavaScript Event object to Firebase database to extract certain event by its id. While Firebase will prepare the data for send it back, the back-end able to execute some other part of code. And when the data will be accepted by the Node.js server, it can continue execute the part of code which was interrupted by request to firebase database.

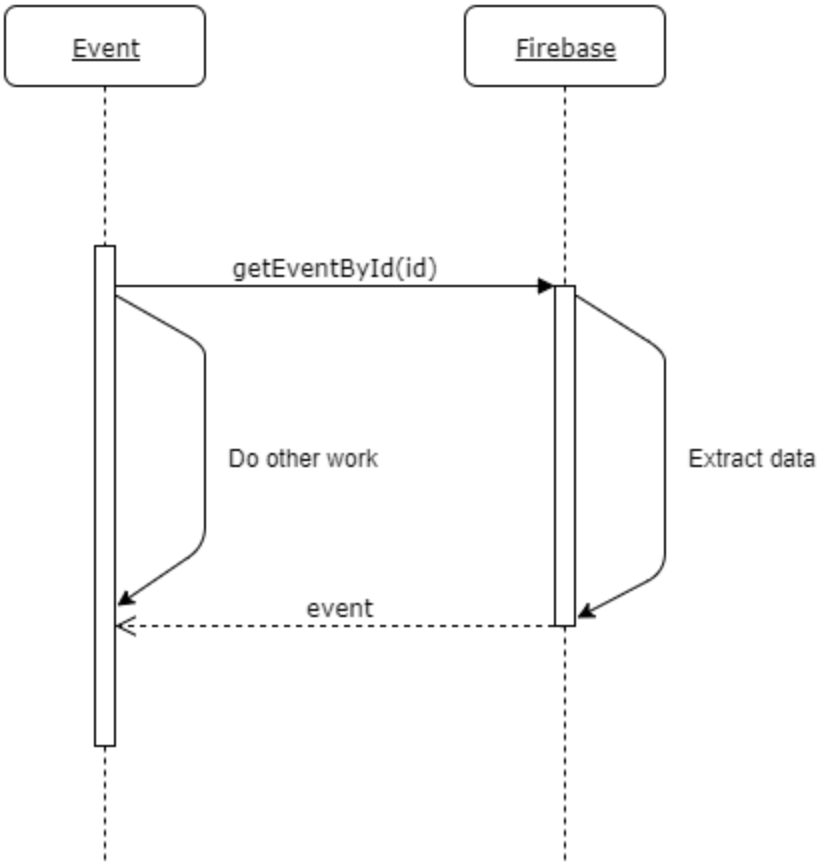

Figure 3: Firebase sequence diagram.

Another useful tool that provided by Firebase is cloud functions. Cloud functions is JavaScript code that execute on special cloud firebase environment, that can handle a HTTP or HTTPS requests. That functions allow to use simple npm modulus and can be written as simple Node.js server. Also they can constitute a trigger for real-time database that executes when the data is changed.

Cloud functions provide a connection between neural networks and the user side components in the context of event management application. For example, when user create an event, the trigger for creating the record executes and the necessary event's data is entered into another special table, which is listened to by neural networks. Having made the necessary data processing, neural networks decide that to do some action. For example as result neural networks will again create a record in the firebase database, for example it can be some notification for the user or detection that there is spam event.

Also, firebase allows you to collect analytics of the entire application. That analytics provides insight into application usage and user engagement. And such kind of analytics data will help to find the vulnerabilities in the application, find which part of application is more convenient to users, which needs to be finalized. Also that data will contribute to the creating a marketing campaign in the future, which in turn will stimulate the activity of old users and should be able to attract new users.

4.3 Front-end web

Web part of Hikester implemented as a single page web application. The logic of the front-end implemented on the JavaScript, using the reactJS framework, and the web application state control system - REDUX. The principle of single page application is achieved thanks to the state model of the ReactJS framework. That model allow to handle changing of some attribute of virtual DOM element in certain moment of time and when it was changed the ReactJS change a state of certain element on web page. So the web page can change each element that contains without reloading. That is the reason why the event management application is single page application.

Front-end communicate with back-end and firebase database through asynchronous request. For more convenient way to work with such kind of request used Promises principle of JavaScript. The connection with the back-end implemented via JSON API using a third-party library - Axios. To

support liquid architecture, another important criterion for constructing the front-end was the availability of responsive design.

4.4 Mobile application (IOS & Android)

The key task of developing a mobile application is to make the most simple, convenient and user-friendly interface for the user.

The main design pattern on which built mobile applications based on is VIPER: View, Interactor, Presenter, Entity, and Routing. That pattern allow to develop applications on high level of abstraction. Separating application on several different and independent part will provide to more easily support in future. The VIPER design pattern allows to achieve such kind of separation.

Figure 4 illustrate the interaction of that components. All interactions with mobile applications follow the next schema:

When user do some action on the View component, it send request to Presenter. Then Presenter recognize what the action was, some input/output operation, confirmation, event creation etc. And then it decides which step will be next. Presenter component asks for Interactor to updates some data or not. If it is necessary Interactor will change the data of Entity component and after all changes will send it back to Presenter. The Entity manage the data in firebase real-time database. When control comeback to Presenter that will change some data or state of View. This is the sequence of the steps that mobile application's do for each response for some action that came from the user. The Router components are responsible for all the interactions between all other components.

As the front-end mobile applications connection with the back-end was maintained using the JSON API.

The mobile applications use Observable design pattern for communication with Firebase real-time database.

That design pattern's UML diagram is represented in figure 5. Observable design pattern consists of 2 interfaces - Observable and Subscriber. Also it has theirs 2 implementations. The main idea is that all observable objects contains some list of subscribers and when there is happen some action with Observable object it will notify all subscribers about it. Also Observable able to subscribe some new subscriber or unsubscribe some old one. Subscribers have onNext(T t) method that allow to get some object as result of update, and then do some computations with that accepted date. So when action is

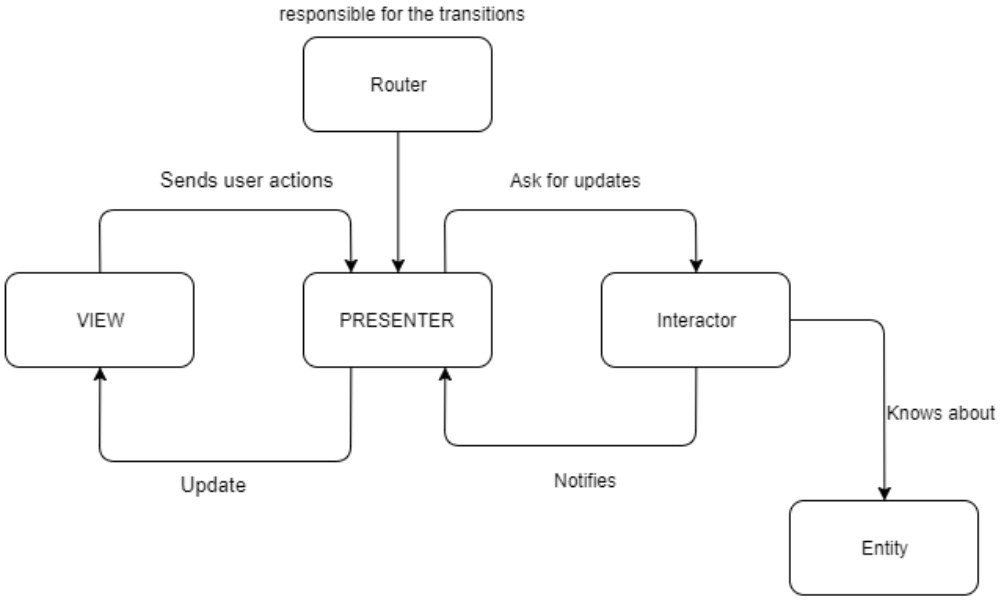

Figure 4: VIPER. Design pattern's modules interaction description.

happen with Observable object it call onNext(T t) method on all subscribers and send to them some new data.

Firebase real-time database represent the observable object and all users request is related to firebase as subscriber. For example if user A is searching some event and filtered the searching by certain tag and time, the application will handle the firebase database for some changes and will extract all necessary data. And if at that moment some user B will create a event with parameters that satisfy the user A searching criteria the firebase real-time database (observable) will notify all users (subscribers) including user A that there was some change. And user A will see in real time that there will appear some new event in which he may be interested without any additional user actions.

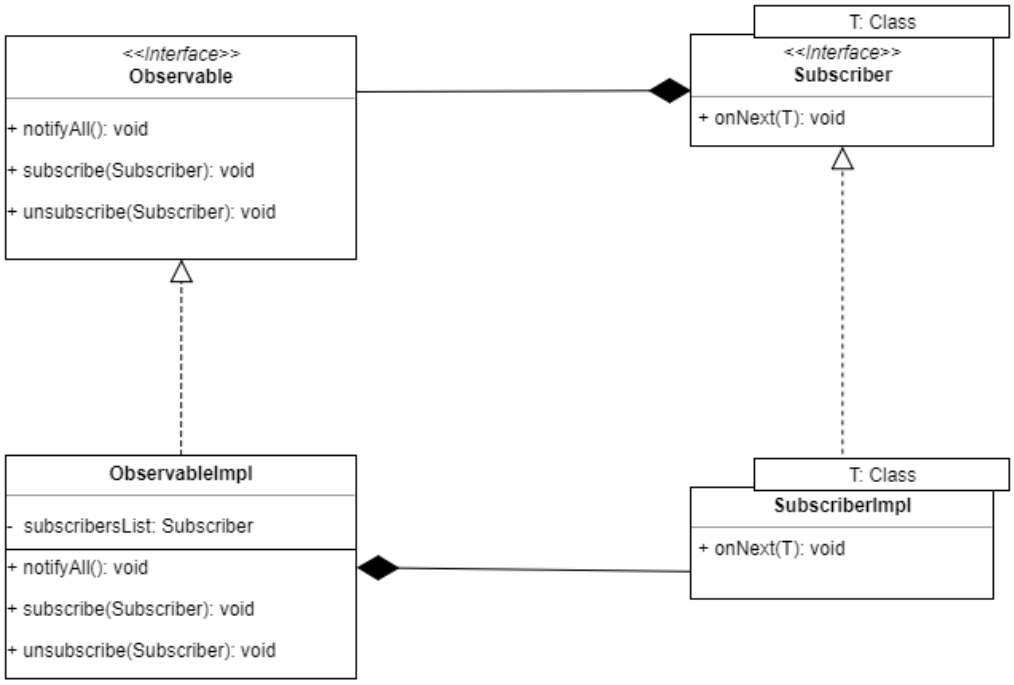

Figure 5: Observable design pattern UML diagram.

4.5 Auxiliary services

4.5.1 Neural networks

The neural networks in Hikester project is represented as a separate service, which is accessed through back-end and firebase cloud functions. The need for neural networks is due to the fact that, for the convenience of the user, it is necessary to create a more user-friendly environment. That will ultimately determine the user's needs by itself, as well as shield users from unwanted content.

There are 3 neural networks:

- *Recommendation system*
- *Spam recognition system*
- *Python neural network*

For implementing all that features were developed neural networks. These networks are designed on Python, with using special neural networks libraries.

4.5.1.1 Recommendation system

This systems is responsible for analyze events which was recently created. The main data process of recommendation system identify the interests of users.

In Hikester project each user is related to some group. That group identified by user social profile and also by behavior in event management application, such as creating certain type of event, accepting or refusal of participation on some events. When user create some event, the neural networks is notified about it and start analyze that data. After that recommendation system send notifications to all users which may be interested on event which was recently created.

4.5.1.2 Spam recognition system

This systems is responsible for analyze the content of event which was recently created. Spam recognition is reasonably important because prevent users from malicious people who want to spoil the application and from seeing a spam in the application is one of the most relevant part in each project. It is very uncomfortable to use any application when there are a lot of low-quality content.

4.5.1.3 Parameter optimizing system

The Parameter Optimizing System suggest to users, who create some new event, which parameters (date, time, location, tags) is more relevant to choose, for achieving the bigger activity and greater participation of people on that event. The system able to do some decision which parameters should be chosen in specific case. That decision based on processing the data of previous events.

4.5.2 Elasticsearch

The Elasticsearch search engine is used to implement a full-scale search in noqsl data base. This tool provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents.

4.5.3 Geofire

Geofire allows to use real-time location queries with firebase real-time database. This tool needs for the display of certain events in a given radius from the current user's or given location.

4.6 Summary

The work of the application will be based on the interaction of the user, both with the web version, and with mobile applications, with the firebase platform. All information will be stored right there, in the fireabase's real time data base. Also, any user behavior (registration, creation of an event, participation, refusal of participation, location changes, search) will be tracked by the back-end cloud functions. The back-end, will pass this information through neural networks, which in turn will determine to which category the user belongs, what he prefers, and what events can interest him. Referring to this, neural networks will send notifications to users that there are certain events near them, about which, perhaps, they would like to be notified. Also, this information will let to sort the events in a unique order for each user, taking into account his preferences. The spam recognition system allows to user avoid interaction with some useless events. And the parameter optimizing system provide to users some useful suggestions for more convenient searching.

5 Evaluation and Discussion

This chapter will be devoted to describing the results achieved regarding the requirements described in Chapter 3. Also a description of the shortcomings and future developments.

5.1 Results

The requirements which was mentioned in chapter 3.1 was done in the current state of application. This was facilitated by the fact that the main platform which was chosen for developing the application was the Firebase platform. The greatest advantage of using this platform is the availability of a Firebase real-time database. That allowed to built real-time application in the most easiest way. This kind of user interaction with the system makes it possible to provide a very convenient user interface. Thus ensuring a very fast and comfortable using of Hikester application.

Also event management application satisfy to requirement for single page application. The ReactJS framework provided state approach for that project. Each user interaction change just part of web page. User only need to download once all the data - HTML, JavaScript and css. After that the application become like desktop application. Another feature of that approach is if there will be some problems with domain or hosting but other components of application will work and user will have downloaded page in browser, the application will work like there is no any problem.

For achieving the cross-platform approach there were developed two mobile applications for iOS and Android operation systems. That two applications are very similar. The main differences between them only in styles. That approach provide the coverage to a wider audience. So user can use any modern device, leading in the field of mobile devices. Also responsive web design provide that opportunity in more easily way, if user want to use just web site he can do it without any restrictions on device.

5.2 Problems

The main problem that should be solved as quick as possible is the design of web site. The appearance of the site always produces most of the influence on the first impression of the project. No matter what convenient functionality was not implemented in the application, if its appearance is not particularly

pleasing to the eye, it is unlikely large number of people will be use that application.

5.3 Future work

As was said to improve the appearance of event management project there should be developed the website design. That will imply the improving a user interface and as result applications will be more convenient for user and will provide the more user quantity that will use the Hikester application.

References

[1] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[2] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[3] David Garlan and Mary Shaw. An introduction to software architecture. Technical report, Pittsburgh, PA, USA, 1994.

[4] PhD Aleksandar Damnjanovic. *METHODS OF EFFORT ESTIMATION IN SOFTWARE ENGINEERING*. 06 2011.

[5] Barry W. Boehm, Clark, Horowitz, Brown, Reifer, Chulani, Ray Madachy, and Bert Steece. *Software Cost Estimation with Cocomo II with Cdrom*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[6] A. Taivalsaari, T. Mikkonen, and K. Systä. Liquid software manifesto: The era of multiple device ownership and its implications for software architecture. In *2014 IEEE 38th Annual Computer Software and Applications Conference*, pages 338–343, July 2014.

[7] Barry Boehm, Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby. Cost models for future software life cycle processes: Cocomo 2.0. *Annals of Software Engineering*, 1(1):57–94, Dec 1995.

[8] Andrea Gallidabino and Cesare Pautasso. The liquid.js framework for migrating and cloning stateful web components across multiple devices. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 183–186, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee.

[9] R. Greg Lavender and Douglas C. Schmidt. Active object – an object behavioral pattern for concurrent programming, 1995.

[10] S. Tilkov and S. Vinoski. Node.js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80–83, Nov 2010.

[11] K. Fysarakis, D. Mylonakis, C. Manifavas, and I. Papaefstathiou. Node.dpws: Efficient web services for the internet of things. *IEEE Software*, 33(3):60–67, May 2016.

[12] Michael Mikowski and Josh Powell. *Single Page Web Applications: JavaScript End-to-end*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2013.

[13] A. Mesbah and A. van Deursen. Migrating multi-page web applications to single-page ajax interfaces. In *11th European Conference on Software Maintenance and Reengineering (CSMR'07)*, pages 181–190, March 2007.

[14] A. Horton and R. Vice. *Mastering React*. Community experience distilled. Packt Publishing, 2016.

[15] Reactjs documentation. `https://facebook.github.io/react/docs/hello-world.html`.

[16] Tommi Mikkonen, Kari Systä, and Cesare Pautasso. *Towards Liquid Web Applications*, pages 134–143. Springer International Publishing, Cham, 2015.

[17] Architecting ios apps with viper. `https://www.objc.io/issues/13-architecture/viper/`. Accessed: September 2017.

[18] Rober Morales-Chaparro, Marino Linaje, Juan Preciado, and Fernando Sánchez-Figueroa. Mvc web design patterns and rich internet applications. 01 2007.

[19] Ioannis K. Chaniotis, Kyriakos-Ioannis D. Kyriakou, and Nikolaos D. Tselikas. Is node.js a viable option for building modern web applications? a performance evaluation study. *Computing*, 97(10):1023–1044, Oct 2015.