|  |  |
| --- | --- |
|  | untitled |

**DEVELOPER CONTACTS**

Bachelor’s Thesis

Batuhan Kurt

216SE2016L

Supervised by

Emine EKİN

December 2020

**DEVELOPER CONTACTS**

Abstract

My Developer Contacts system design is quite simple, understandable but very functional. This web-site provides a platform where users can access anywhere and can share anything with each other. Developers can easily find solutions and answers, according to their projects, also they can share their CV’s and find a job that related to their specifications with my functional system design. After the design stage, I can decrease the time required the implementation.

This project is a cooperation platform where users can access anywhere and can share anything with each other. The main purpose of this project is to find solutions and answers to their projects, also they can share their CV’s and find a job that related to their specifications. This website, which contains several profiles of developers, can provide blogs and useful comments to developers. Developer can create an account, a profile and blogs. In addition, they can like or dislike their comments. Other specification of this project is that developers can look at other developer’s github repositories, or their social media accounts such as facebook, twitter, linkedin and youtube. Also, a developer can add software skills to his/her own profile. As a result, this system helps the users to achieve correct answer to their project and add a contact from software world.

As I said in RAD, the main aim of this project is to find ideal solutions to developers projects, also they can share their information and development skills to find a job that related to their specifications. This website, which contains several profiles of developers, can provide blogs and useful comments to developers. Developer can create an account, a profile and blogs. In addition, they can like or dislike their comments. Other specification of this project is that developers can look at other developer’s github repositories, or their social media accounts such as facebook, twitter, linkedin and youtube. Also, a developer can add software skills to his/her own profile. As a result, this system helps the users to achieve correct answer to their project and add a contact from software world.

Acknowledgements

Thanks to aids and comments of my supervisor who is Emine EKİN, my project is implemented organized, successful and quality.

**TABLE OF CONTENTS**

[ABSTRACT](#_Toc419915328)

[ACKNOWLEDGEMENTS](#_Toc419915329)

[TABLE OF CONTENTS](#_Toc419915330)

[LIST OF FIGURE](#_Toc419915331)

[DEFINITIONS, ACRONYMS AND ABBREVIATIONS](#_Toc419915332)

[1. Introduction 8](#_Toc419915333)

[2. Literature Review 14](#_Toc419915334)

[3. Proposed System 15](#_Toc419915335)

[3.1 Overview 16](#_Toc419915336)

[3.2 Functional Requirements 17](#_Toc419915337)

[3.3 Nonfunctional Requirements 18](#_Toc419915338)

[3.4 System Models 21](#_Toc419915339)

[3.5 Project Schedule 53](#_Toc419915340)

[3.6 System Decomposition 54](#_Toc419915341)

[3.7 Hardware Software Mapping 56](#_Toc419915342)

[3.8 Persistent Data Management 58](#_Toc419915343)

[3.9 Access Control and Security 59](#_Toc419915343)

[3.10 Global Software Control 60](#_Toc419915343)

[3.11 Boundary Conditions 61](#_Toc419915343)

[3.12 Subsystem Services 62](#_Toc419915343)

[4. Implementations Details, Tests and Experiments 66](#_Toc419915344)

[5. Conclusions and Future Work 67](#_Toc419915347)

[References 68](#_Toc419915348)

**LIST OF FIGURES**

[Figure 3. 4. 1. Use Case Diagram](#_Toc417256554) 47

[Figure 3. 4. 2. Object Model Diagram](#_Toc417256555) 48

[Figure 3. 4. 3. Admin Dynamic Model](#_Toc417256556) 49

[Figure 3. 4. 4. Admin Activity Diagram](#_Toc417256557) 50

[Figure 3. 4. 5. Developer Activity Diagram](#_Toc417256558) 51

[Figure 3. 5. 1. Project Gantt Chart](#_Toc417256449) 52

[Figure 3. 6. 1. System Decomposition](#_Toc417256449) 54

[Figure 3. 7. 1. Hardware Software Mapping](#_Toc417256449) 56

[Figure 3. 8. 1. Persistent Data Management](#_Toc417256449) 59

[Figure 3. 9. 1. Access Control and Security](#_Toc417256449) 60

[Figure 3. 12. 1. Create Developer Profile Interface](#_Toc417256449) 62

[Figure 3. 12. 2. View Developers Interface](#_Toc417256449) 63

[Figure 3. 12. 3. Posts and Like & Dislike Interface](#_Toc417256449) 64

[Figure 3. 12. 3. Comments Interface](#_Toc417256449) 65

**DEFINITIONS, ACRONYMS AND ABBREVIATIONS**

The abbreviations and definitions contained in the document are given below:

* **Developer Contacts:** Contacts of a developer that are added with thanks to my application
* **Admin:** Developer Contacts system admin
* **User:** Role of a developer
* **Model:** A schematic description of a system that accounts for its known or inferred properties
* **System:** Any interacts by the application are considered to be done by the system
* **Efficiency:** The properties of an algorithm, which is the amount of computational resources used by the algorithm
* **Service:** Service is a keyword. Purpose of the service is to provide portfolio sharing among to developers
* **Bottleneck:** The capacity of an application severely limited due to having a single component crowded out.
* **GUI:** Graphical User Interface
* **Interface:** The entity allowing for the user to communicate with the system
* **Resources:** The entities that support the system which are limited e.g. space, data, time
* **OOP:** Object Oriented Programming
* **POP:** Procedural or Produce Oriented Programming
* **MongoDB:** Mongo Database
* **HTML:** Hypertext Markup Language
* **CSS:** Cascading Style Sheets
* **MIT License:** Massachusetts Institute of Technology License
* **SDD:** System Design Document
* **MERN Stack:** MongoDB , Express.js, React, Node.js
* **JWT:** JSON Web Tokens

chapter one

IntroductIon

* 1. **Purpose of the System**

This project is a cooperation platform where users can access anywhere and can share anything with each other. The main purpose of this project is to find solutions and answers to their projects, also they can share their CV’s and find a job that related to their specifications. This website, which contains several profiles of developers, can provide blogs and useful comments to developers. Developer can create an account, a profile and blogs. In addition, they can like or dislike their comments. Other specification of this project is that developers can look at other developer’s github repositories, or their social media accounts such as facebook, twitter, linkedin and youtube. Also, a developer can add software skills to his/her own profile. As a result, this system helps the users to achieve correct answer to their project and add a contact from software world.

* 1. **Scope of the System**

My system is called Developer Contacts. Developer Contacts can be used anywhere and anytime as it is a web-based application. It so simple to use, register to the system, create your profile and ask questions that are related to your project, instantly. This web-site is also a good information source for developers. Any person can register my application. This system collects the people who want to find solutions to his/her project or help to other people who are in the same roof.

* 1. **Objectives and Success Criteria of the Project**

The success of the system depends on providing the given main set of arguments:

* Use GitHub repository effectively.
* The design of register and login systems.
* Association between database and system design.
* Make the general design of system in efficient way.
* The demo of system should ensure good success rate.
* Implementation should be understandable, clear, basic and efficient.
  1. **Overview**

In introduction, I have described a good explanation of scope purpose, criteria, definitions and other explanations which help other people who want to read document. In proposed system, I have described the advantages of Developer Contacts Application and what makes Developer Contacts Application to unique. In functional requirement, I have described that the functions are open to end users. Functional requirements states for explaining scope of the system.

In non-functional requirement section, I have described our system’s reliability, performance and others to make the system accessible to end users. In Scenarios section, I have described use case’s descriptions and explanations. In Use case model, I have stated or use case diagram. In Object model, I have described the programming objects which will implemented later on. In Dynamic model, I have described the flow of system in time with operations and function of actors. In glossary section, I have described dictionary and unknown words in RAD.

* 1. **Design Goals**

The purpose of my project is to fulfill the requirements of software engineering completely. It is to specify all functional and nonfunctional functions together. With the definition of the functions, I have defined all the requirements for my Developer Contacts project and have prepared an infrastructure for future versions or new projects. In my Developer Contacts system design, I provide to my users or my visitors easy access to my website, to find new jobs, to hear about updated news from software world, to reach correct information and to create groups for cooperation with each others. The features my system evaluates based on non-functional functions are as follows:

* **Dependability**

One of the most significant non-functional requirement is system security. The user security is at forefront in my Developer Contacts system. In addition to security and safety, I paid attention to robustness, reliability, availability and fault tolerance criteria to make a complex system.

* **Maintenance**

The maintenance of my Developer Contacts system is periodically performed by the system administrator. Of course, while I am creating to my system, shortcomings such as extensibility, modifiability, adaptability, portability and readability were taken into consideration.

* **End User Criteria**

On my Developer Contacts website, users and visitors can sign up, sign in, view their CV’s or other information, and create groups for cooperation. My system efficiently stores and retrieves user data in a dynamic manner. In addition, I noticed that utility and usability factors are important for me. The Developers system supports Microsoft and MacOS operating systems because it is a browser based application.

* **Performance**

My Developer Contacts system is responsive and it can accomplish a maximum number of tasks easily. The memory space of my system is available for speed optimizations. As I mentioned the response time, through memory criteria are significant for my system.

* **Cost**

I try to accomplish optimal level for cost of my system while I am developing it. In addition, this cost is not only for design considerations, but also executive as well. To maintain backward compatibility with a previous system, I can add to the development cost while reducing the transition cost. By the way, I handle the development cost, deployment cost, updating costs, maintenance cost and administration cost.

* 1. **Glossary**

**Developer:** Developer who register to system.

**Admin:** The system administrator.

**Create Profile:** Developer can create profile with this functionality.

**Create Group:** It is where the developer can create private group.

**Like & Unlike Comment:** It is where the developer like or dislike a comment.

**Database:** A collection of specific data.

**View and Edit Dashboard:** It is where the developer can view and edit his/her portfolio.

**Delete Account:** It is where the admin can delete any account, or a developer can delete his/her own account.

**Post Feed & Comment:** It is where the developer can share posts or comment to posts.

**Delete Groups:** It is where the admin can delete any private group.

**Functionality:** An action or control supported by our system for any of the users to carry out a task.

**Log in:** Being entered to the Developer Contacts System with a specific registered account.

**Log out:** Being exited from the Glass Booking System with a specific registered account.

**Use case:** A methodology used in system analysis to identify, clarify, and organize system requirements.

**Registered User:** A user who has an account on the Developer Contacts System.

**Password:** A secret word or phrase that must be choose of all developers when they are register.

**Profile:** A section that includes the developer’s information.

chapter TWO

LITERATURE REVIEW

1. **Current System**

My system is a very agile, intelligent and useful system and, I designed the system for the web browsers. My system differs from other sites, other sites may help to developers with a solution, however they do not provide CV sharing or showing github repositories or do not present details of social media accounts at the same time.

Other difference of my system is that developers can create very detailed portfolio, they can add their software skills and create a group with their contacts. The communication between the system administrators and users (developers) works in a synchronous manner. The platform is running synchronously, the host and system administrator are very quick to inform. As a result, I used the MVC (Model, View, Controller) architecture style for my Developer Contacts system. Because, MVC is well suited for interactive systems, especially when multiple views of the same model are needed.

In addition, I was implemented with the client-server architecture to this application. In this system, there is a client which is the app, and there is a server. The application makes API calls to this server and returns the results back to server. At the end of all this, there exists a database which is on the cloud. The database retrieves any user related data. The client sends request to the server and the server returns results.

chapter THREE

PROPOSED SYSTEM

1. **Proposed System**

Documents the requirements elicitation and the analysis model of the new system. The Developer Contacts system is web based. My project will be very useful for developers who are looking for a new jobs or projects.

I am going to use the MVC architecture for the new system. The Model View Controller (MVC) design pattern requires that an application is made up of a data model, presentation information, and control information. Each of these need to be separated into different models. The MVC mostly associates to the UI / interaction layer of an application.

I still need to have a business logic layer, and some service layer and data access layer. The model contains only the pure application data, it contains no logic describing how to present the data to a user.

The View displays the model’s data to the user. The view knows how to access the model’s data, but it does not know what this data means or what the user (developer) can do to change it. The controller exists between the view and the model. It listens to events triggered by the view and executes the appropriate response to the events. In most cases, the response is to call a method on the model. So, the view and the model are connected through a notification mechanism, the result of this action is then automatically displayed in the view.

* 1. **Overview**

In my Developer Contacts system, I have designed subsystems based on software engineering requirements, working more efficiently, rapidly and working together. This provides me a coherence. I divided to my system some subsystems that are; create groups interface, developer interface, create portfolio interface, create post interface, the web-site main page, the main-page without sign in home page interfaces.

MVC stands for Model, View and Controller. MVC separates application into three components ; Model, View and Controller. It is well suited of applications with a user interface which is my developer interface.

Model maintains the data of an application, it also represents shape of the data and business logic. Model objects are responsible for retrieving and storing model state in a database.

The view represents the user interface. It is where commands are taken and triggers events. It views the display data to the user also allows them to modify the data.

Controller serves as the link between the client and the system. Client requests are handled by the controller. Usually, client interacts with the View which will then create appropriate URL request, this request is handled by a controller.

Advantages of using an MVC include:

* Ability to provide multiple views
* Modification doesn’t affect the entire system
* Faster development process
* Support for asynchronous technique

Disadvantages of using an MVC include:

* Increased complexity
* Performance bottleneck
* Inefficiency data access in view.
  1. **Functional Requirements**

***Developer Functional Requirements***

* **Sign Up**

The developer registers on the system to the web-site.

* **Sign In**

The developer logs in to the system with the username and password from the homepage.

* **Create Group**

A developer can create a private group and other developers can join this group.

* **Create Profiles ( Portfolios )**

A developer can create his/her own profile according to his/her personal information.

* **Add Experience Information**

A developer can add his/her experience information into the profile.

* **Add Education Information**

A developer can add his/her education information into the profile.

* **Add Bio and Skill Set Information**

A developer can add his/her biography information and software skill sets into the profile.

* **Add Github Repository Information**

A developer can add his/her own github repository information into the profile.

* **Add Social Media Accounts Information**

A developer can add his/her social media accounts information such as facebook, twitter, linkedin and youtube into the profile.

* **View and Edit Dashboard**

A developer view to his/her own portfolios, he/she can edit ( insert or delete ) profile information.

* **Delete Account**

A developer can delete his/her profile.

* **Post Feed & Comment**

A developer can post an entry and he/she can comment an entry.

* **Like, Dislike and Delete Comments**

A developer can add like or dislike a comment. Also, he/she can delete his/her own comments.

* **Log Out**

Developer exit the system when they finished the operations on the system.

***Admin Functional Requirements***

* **Sign In;**

Admin part is different from Developer Sign in. Admin should enter the different UML. In this page, admin can login on system with username and password. Password is given by the system database.

* **Delete Accounts**

Admin can delete inappropriate accounts from the system, these accounts do not shown on the application anymore.

* **Delete Groups**

Admin can delete the any group that are created by a developer.

* **Delete Posts & Comments**

Admin can delete inappropriate posts and comments from the system, these entries do not shown on the application anymore.

* **Log Off**

Admin exit the system when they finished the operations on the system.

* 1. **Nonfunctional Requirements**

*Usability:* Using the Developer Contacts website is quite simple, there is a design that everyone can understand and a global interface. Our interface is based on portfolios, blogs and comments, so my priority to use our system without any confusion. Developer Contacts continues to work to create a more effective interface.

*Reliability:* Developer Contacts system has almost all security requirements that is %90.

*Performance:* Developer Contacts uses Mongo DB as its database system, which has a permanent storage space. So, it reacts quite fast and at the same time about 5000 parallel users can enter my system.

*Supportability:* The maintenance of the Developer Contacts system is a secure portal that is open only for administrator.

*Implementation:* The Developer Contacts system is implemented in the JavaScript ( Node.js, Express.js and React.js ) programming language with excellent OOP features using Visual Studio Code and Mongo DB Cloud ( [cloud.mongodb.com](https://cloud.mongodb.com) ) programs and platforms. Developer Contacts uses Mongo DB database system, and Node.js environment in back-end, In-Front-End, Developer Contact uses React.js framework. You can access it via www.developercontacts.com.

*Interface:* The interface of the Developer Contacts system is designed with using React.js components and required CSS files for interface implementation. In addition, despite the use of Interfaces, third party applications and services used in the Developer Contacts system and disclosed to users may not be used. There are no inherited projects that present services or interfaces.

*Packaging:* The system packaging will be with future versions.

*Legal:* Developer Contacts is licensed with MIT License. It is provided "as is" without warranty of any kind, including, but not limited to, marketability, manufacturability, software, certain fitness for purpose, and any infringement of rights. In any unusual circumstances, administrators shall not be liable for any damage or other liability arising out of the author, authors or copyright holders, from a software contract, tort or otherwise.

* 1. **System Models**

Here, I show the high level functionalities of my system and how the users of my system interact with the system. For the object models I depict our classes using UML diagrams and show the relation between them. In the dynamic model, I showed interactions between objects within a use case using sequence diagrams and showed how state machines are used to show the behaviour of that single object.

### Scenarios

**Scenario 1**

|  |
| --- |
| **Scenario name:** Sign up |
| **Participant actor instances:**  Batuhan: Admin, Burak: Developer |
| **Flow of events:**   * Firstly, if a visitor enters to Developer Contacts web-site. * Then, he wants to register Developer Contacts web site. * He clicks to "Sign Up" button on the main page. * He fills the all fields and clicks Sign Up. * If information are correct, and the passwords that are entered by him matches,   he can access his account.   * Finally, user account is created. |
|  |

**Scenario 2**

|  |
| --- |
| **Scenario name:** Log In |
| |  | | --- | | **Participant actor instances:**  Batuhan: Admin, Burak: Developer | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * Then, if he is a developer, he wants to create a profile or, if he is an admin, he wants to delete an account. * So, He clicks to "Log In" button on the main page. * He fills the username and password fields, then clicks sign in. * If information are correct, he will sign in, but if the password is not correct, he canclicks to "Remember Password" button to reset his password. * Finally, he signs in and if he wants to log out, he clicks to "Log out" button to log on main page. |

**Scenario 3**

|  |
| --- |
| **Scenario name:** View Developers List |
| |  | | --- | | **Participant actor instances:**  Batuhan: Admin, Burak: Developer | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * Then, if he is a developer or an admin, he can see the “Developers” button on the main page. * So, He clicks to "Developers" button on the main page. * He will see the list of developers who registered to Developer Contacts System. * He can look at the name, surname, title and software skills of the registered developers. |

**Scenario 4**

|  |
| --- |
| **Scenario name:** View Developers Profiles |
| |  | | --- | | **Participant actor instances:**  Batuhan: Admin, Burak: Developer | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * Then, if he is a developer or an admin, he can see the “Developers” button on the main page. * So, He clicks to "Developers" button on the main page. * Then, he wants to look at a developer profile. So, he clicks the “View Profile” button which belongs to a developer who registered to system. * Finally, he can see the “profile” page which belongs to a developer. On the profile page, he can see the avatar picture, biography, education, experience, skill sets, github repositories and social media accounts of related developer. |

**Scenario 5**

|  |
| --- |
| **Scenario name:** View Dashboard |
| |  | | --- | | **Participant actor instances:**  Burak: Developer | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * Then, he can see the “Dashboard” button on the main page. * So, He clicks to "Dashboard" button on the main page. * Then, he can see his developer profile. * Also, he can edit his profile with “edit profile” button, add an experience with “add experience” button. |

**Scenario 6**

|  |
| --- |
| **Scenario name:** Dashboard Edit Profile |
| |  | | --- | | **Participant actor instances:**  Burak: Developer | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * Then, he can see the “Dashboard” button on the main page. * So, He clicks to "Dashboard" button on the main page. * Then, he can edit his profile, he clicks the “edit profile” button. * Finally, he can update his information with filling the related field which are on the edit profile page. Also, he can update the social media accounts information on this page. |

**Scenario 7**

|  |
| --- |
| **Scenario name:** Dashboard Add Experience |
| |  | | --- | | **Participant actor instances:**  Burak: Developer | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * Then, he can see the “Dashboard” button on the main page. * So, He clicks to "Dashboard" button on the main page. * Then, he can add an experience to his profile, he clicks the “add experience” button. * He can add an experience with filling the “Company”, “Job Title”, “Location”, “From Date”, “To Date” and “Job Description” fields. * Finally, he can clicks “Submit” button to adding experience. |

**Scenario 8**

|  |
| --- |
| **Scenario name:** Dashboard Add Education |
| |  | | --- | | **Participant actor instances:**  Burak: Developer | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * Then, he can see the “Dashboard” button on the main page. * So, He clicks to "Dashboard" button on the main page. * Then, he can add an education information to his profile, he clicks the “add education” button. * He can add an education information with filling the “School”, “Degree or Certification”, “Field of Study”, “From Date”, “To Date” and “Program Description” fields. * Finally, he can clicks “Submit” button to adding experience. |

**Scenario 9**

|  |
| --- |
| **Scenario name:** Delete My Account |
| |  | | --- | | **Participant actor instances:**  Burak: Developer | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * Then, he can see the “Dashboard” button on the main page. * So, He clicks to "Dashboard" button on the main page. * Then, he can delete his account with clicking the “delete my account” button. * Finally, he can clicks “delete my account” button to delete his account. |

**Scenario 10**

|  |
| --- |
| **Scenario name:** Delete Experience and Education Information |
| |  | | --- | | **Participant actor instances:**  Burak: Developer | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * Then, he can see the “Dashboard” button on the main page. * So, He clicks to "Dashboard" button on the main page. * Then, he can delete his experience information with clicking the “delete experience” ,   and education information with clicking the “delete education” button. |

**Scenario 11**

|  |
| --- |
| **Scenario name:** Post Feed |
| |  | | --- | | **Participant actor instances:**  Burak: Developer | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * Then, he can see the “Post Feed” button on the main page. * So, He clicks to "Post Feed" button on the main page. * Then, he can post anything on this page with filling the post field. * Finally, he clicks the “Submit” button to create post. |
|  |

**Scenario 12**

|  |
| --- |
| **Scenario name:** Post Comment |
| |  | | --- | | **Participant actor instances:**  Burak: Developer | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * Then, he can see the “Post Feed” button on the main page. * So, He clicks to "Post Feed" button on the main page. * Then, he can see the all posts on this page. He can comment to post with choosing a post and clicking “Comments” button. * Finally, he fills the comment field and clicks the “Submit” button to comment. |
|  |

**Scenario 13**

|  |
| --- |
| **Scenario name:** Like Comment |
| |  | | --- | | **Participant actor instances:**  Burak: Developer | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * Then, he can see the “Post Feed” button on the main page. * So, He clicks to "Post Feed" button on the main page. * Then, he can see the all posts on this page. * He can like the comments, if he thinks the post is correct. * Then, he clicks to like image. |
|  |

**Scenario 14**

|  |
| --- |
| **Scenario name:** Dislike Comment |
| |  | | --- | | **Participant actor instances:**  Burak: Developer | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * Then, he can see the “Post Feed” button on the main page. * So, He clicks to "Post Feed" button on the main page. * Then, he can see the all posts on this page. * He can dislike the comments, if he thinks the post is incorrect. * Then, he clicks to dislike image. |
|  |

**Scenario 15**

|  |
| --- |
| **Scenario name:** Delete Comment (Developer) |
| |  | | --- | | **Participant actor instances:**  Burak: Developer | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * Then, he can see the “Post Feed” button on the main page. * So, He clicks to "Post Feed" button on the main page. * Then, he can see the all posts and comments on this page. * He can delete his own comment with clicking to cancel image. |
|  |

**Scenario 16**

|  |
| --- |
| **Scenario name:** Delete Accounts |
| |  | | --- | | **Participant actor instances:**  Batuhan: Admin | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * He log in to the system as an admin. * Then, he can see the “Developers” button on the main page. * So, He clicks to "Developers" button on the main page. * Then, he can delete any account with clicking to “delete account” button. If he thinks this account is inappropriate. * Finally, the deleted account is removed from the system. |

**Scenario 17**

|  |
| --- |
| **Scenario name:** Delete Comments and Posts ( Admin ) |
| |  | | --- | | **Participant actor instances:**  Batuhan: Admin | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site. * He log in to the system as an admin. * Then, he can see the “Post Feed” button on the main page. * So, He clicks to "Post Feed" button on the main page. * Then, he can delete any post with clicking to “delete post” button or delete any comment with clicking to “delete comment” button. If he thinks this post or comment is inappropriate. * Finally, the deleted post or comment is removed from the system. |

**Scenario 18**

|  |
| --- |
| **Scenario name:** Log Off |
| |  | | --- | | **Participant actor instances:**  Batuhan: Admin, Burak: Developer | |
| **Flow of events:**   * A visitor enters to Developer Contacts web-site, and log in the web-site with his username and password. * Then, he wants to exit Developer Contacts web-site. * He clicks to “Log off” button. * Then system immediately logs him out and redirects to him on the home page. |

### Use Case Model

**Use case 1**

|  |
| --- |
| **Use case name:** Sign Up |
| **Participant actors:** Admin, Developer |
| **Flow of events:**   * Actor enters to the site. * Actor clicks "Sign Up" button. * Site responds by displaying the Sign-up screen. * Actor enters all required fields. * Actor clicks "Sign Up" button. * System inserts the information to the database which is provided by User. * System redirecting actor to login page. |
| **Entry Condition:** Actor visits the URL page of the system. |
| **Exit Condition:** Actor clicks "Sign-up" button and entered information should be correct. |
| **Quality Requirements:**  1. If Actor leaves one or more fields empty, system displays a warning message, like "This area cannot be empty.”  2. The information that user entered is checking and verified by the system in real-time. |

**Use case 2**

|  |
| --- |
| **Use case name:** Log In |
| **Participant actors:** Admin, Developer |
| **Flow of events:**   * Actor goes on sign-in page. * System responds by displaying the login screen of the application. * Actor enters username and password to the login fields. * Actor clicks "Log in" button. * If the username and password match with the username and password which is stored in database. User authenticates. * System fetches the dashboard of user. |
| **Entry Condition:** Actor visits the URL page of the system. |
| **Exit Condition:** Actor clicks "Log In" button and entered information should be correct. |
| **Quality Requirements:**  1. If Actor leaves one or more fields empty, system displays a warning message, like "This area cannot be empty.”  2. If the information’s checked from database are not true, system displays a warning message, like "Wrong username or password, please retry.” |

**Use Case 3**

|  |
| --- |
| **Use Case Name:** Create Developer Profile |
| **Participant actor instances:** Developer |
| **Flow of events:**   * This event starts with Developer’s clicking Dashboard page. * User can create developer profile with clicking “Edit Profile” button. * System shows to related field such as name, surname, career information, city, software skills, additional descriptions. * Developer clicks any field and types into it, then developer clicks to submit button. * System creates the profile in database. |
| **Entry Condition:** Developer should log in to the system. |
| **Exit Condition:** Profile created. |
| **Quality Requirement:**  **1.** System response to the actor arrives in a few seconds. |

**Use Case 4**

|  |
| --- |
| **Use Case Name:** View Developer Lists |
| **Participant actor instances:** Developer, Admin |
| **Flow of events:**   * This event starts with actors enters to Developer Contacts web-site. * Then, if actor is a developer or an admin, actor can see the “Developers” button on the main page, and actor click to "Developers" button on the main page. * Actor will see the list of developers who registered to Developer Contacts System. * Actor can look at the name, surname, title and software skills of the registered developers. |
| **Entry Condition:** Actor should log in to system. |
| **Exit Condition:** Developer profiles listed. |
| **Quality Requirement:**  **1.** System response to the actor arrives in a few seconds. |

**Use Case 5**

|  |
| --- |
| **Use Case Name:** View Developer Profiles |
| **Participant actor instances:** Developer, Admin |
| **Flow of events:**   * This event starts with actors enters to Developer Contacts web-site. * Then, if actor is a developer or an admin, he can see the “Developers” button on the main page. * So, actor clicks to "Developers" button on the main page. * Then, actor wants to look at a developer profile. So, he clicks the “View Profile” button which belongs to a developer who registered to system. * Actor can see the “profile” page which belongs to a developer. On the profile page, actor can see the avatar picture, biography, education, experience, skill sets, github repositories and social media accounts of related developer. |
| **Entry Condition:** Actor should be logged. |
| **Quality Requirement:**   * System response to the actor arrives in a few seconds. |
| **Exit Condition:** Developer profile is viewed. |

**Use Case 6**

|  |
| --- |
| **Use Case Name:** View Dashboard |
| **Participant actor instances:** Developer |
| **Flow of events:**   * This flow starts with clicking to "Dashboard" button on the main page. * Developer can see his/her developer profile. * Also, developer can edit his profile with “edit profile” button, add an experience with “add experience” button and add an education information with “add education” button. |
| **Entry Condition:** Developer should be logged. |

**Use Case 7**

|  |
| --- |
| **Use Case Name:** Delete Account |
| **Participant actor instances:** Developer |
| **Flow of events:**   * This event starts with clicking to "Dashboard" button on the main page. * Then, developer can delete his account with clicking the “delete my account” button. * Developer can clicks “delete my account” button to delete his account. |
| **Entry Condition:** User should have a developer account. |
| **Exit Condition:** Developer account is deleted. |
|  |

**Use Case 8**

|  |
| --- |
| **Use Case Name:** Delete Experience and Education Information |
| **Participant actor instances:** Developer |
| **Flow of events:**   * This event starts with clicking to "Dashboard" button on the main page. * Developer can delete his experience information with clicking the “delete experience” and education information with clicking the “delete education” button. |
| **Entry Condition:** Developer should have an experience or education information. |
| **Exit Condition:** Experience or education is deleted. |
| **Quality Requirement:**  **1.** System response to the actor arrives in a few seconds. |

**Use Case 9**

|  |
| --- |
| **Use case name:** Post Feed |
| **Participant actors:** Developer |
| **Flow of events:**   * Developer clicks to the "Post Feed" button on the main page. * Then, developer can post anything on this page with filling the post field. * Developer clicks the “Submit” button to create post. |
| **Entry Condition:** Developer should signed-in. |
| **Exit Condition:** Post is created. |
|  |

**Use Case 10**

|  |
| --- |
| **Use Case Name:** Delete Comment |
| **Participant actor instances:** Developer |
| **Flow of events:**   * This event starts with clicking to "Post Feed" button on the main page. * Then, Developer can see the all posts and comments on this page. * Developer can delete his own comment with clicking to cancel image. |
| **Entry Condition:** Developer should have a comment before. |
| **Exit Condition:** Comment is deleted. |
| **Quality Requirement:**  **1.** System response to the actor arrives in a few seconds. |

**Use Case 11**

|  |
| --- |
| **Use Case Name:** Delete Accounts |
| **Participant actor instances:** Admin |
| **Flow of events:**   * This event starts with clicking to "Developers" button on the main page. * Then, Admin can delete any account with clicking to “delete account” button. If admin thinks this account is inappropriate. * The deleted account is removed from the system. |
| **Entry Condition:** There should be an account in the system. |
| **Exit Condition:** Account is deleted. |
| **Quality Requirement:**  **1.** System response to the actor arrives in a few seconds. |

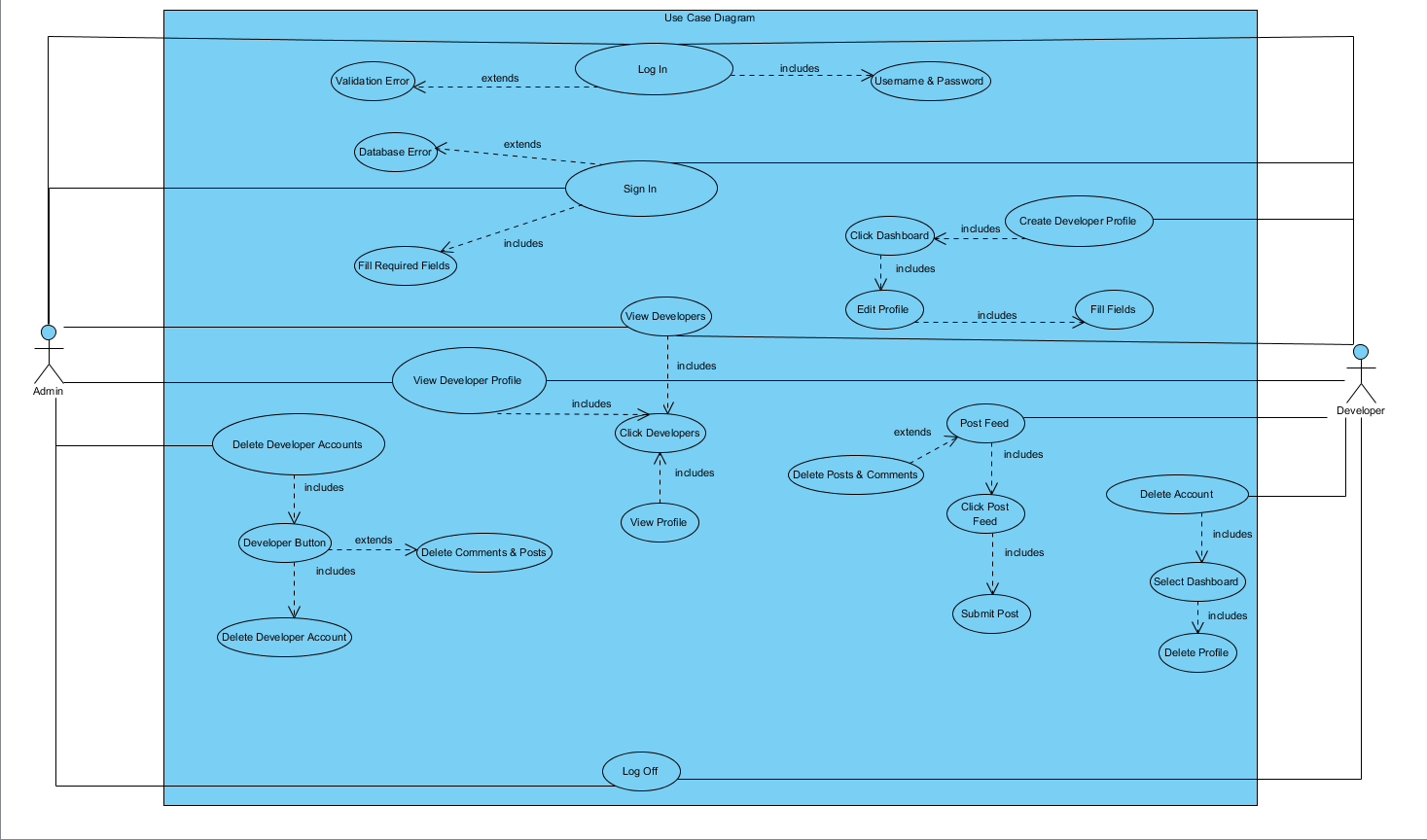
**Use Case 12**

|  |
| --- |
| **Use Case Name:** Delete Comments and Posts |
| **Participant actor instances:** Admin |
| **Flow of events:**   * This event starts with clicking to "Post Feed" button on the main page. * Admin can delete any post with clicking to “delete post” button or delete any comment with clicking to “delete comment” button. If Admin thinks this post or comment is inappropriate. * The deleted post or comment is removed from the system. |
| **Entry Condition:** There should be comments and posts in the system. |
| **Exit Condition:** Comments or posts are deleted. |
| **Quality Requirement:**  **1.** System response to the actor arrives in a few seconds. |

**Use Case 13**

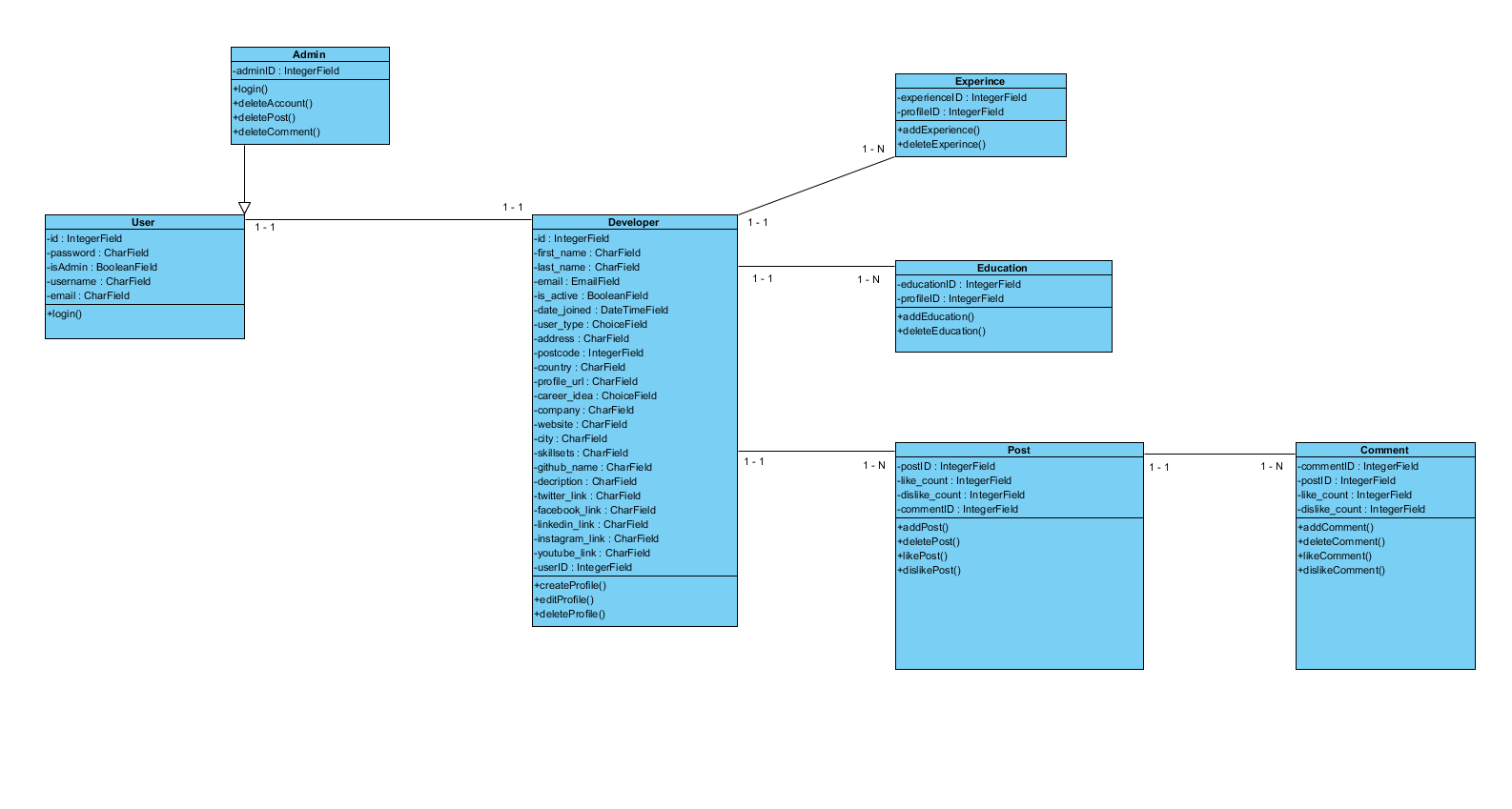
|  |
| --- |
| **Use Case Name:** Log off |
| **Participant actor instances:** Developer, Admin |
| **Flow of events:**   * This event starts with entering to Developer Contacts web-site, and log in the web-site with his username and password. * Actors want to exit Developer Contacts web-site. * Actors click to “Log off” button. * Then system immediately logs them out and redirects to them on the home page. |
| **Entry Condition:** Actors should be logged in the system. |
| **Exit Condition:** Actors redirect to home page. |
| **Quality Requirement:**  **1.** System response to the actor arrives in a few seconds. |

* *Use Case Diagram*



**Figure 3. 4. 1 Use Case Diagram**

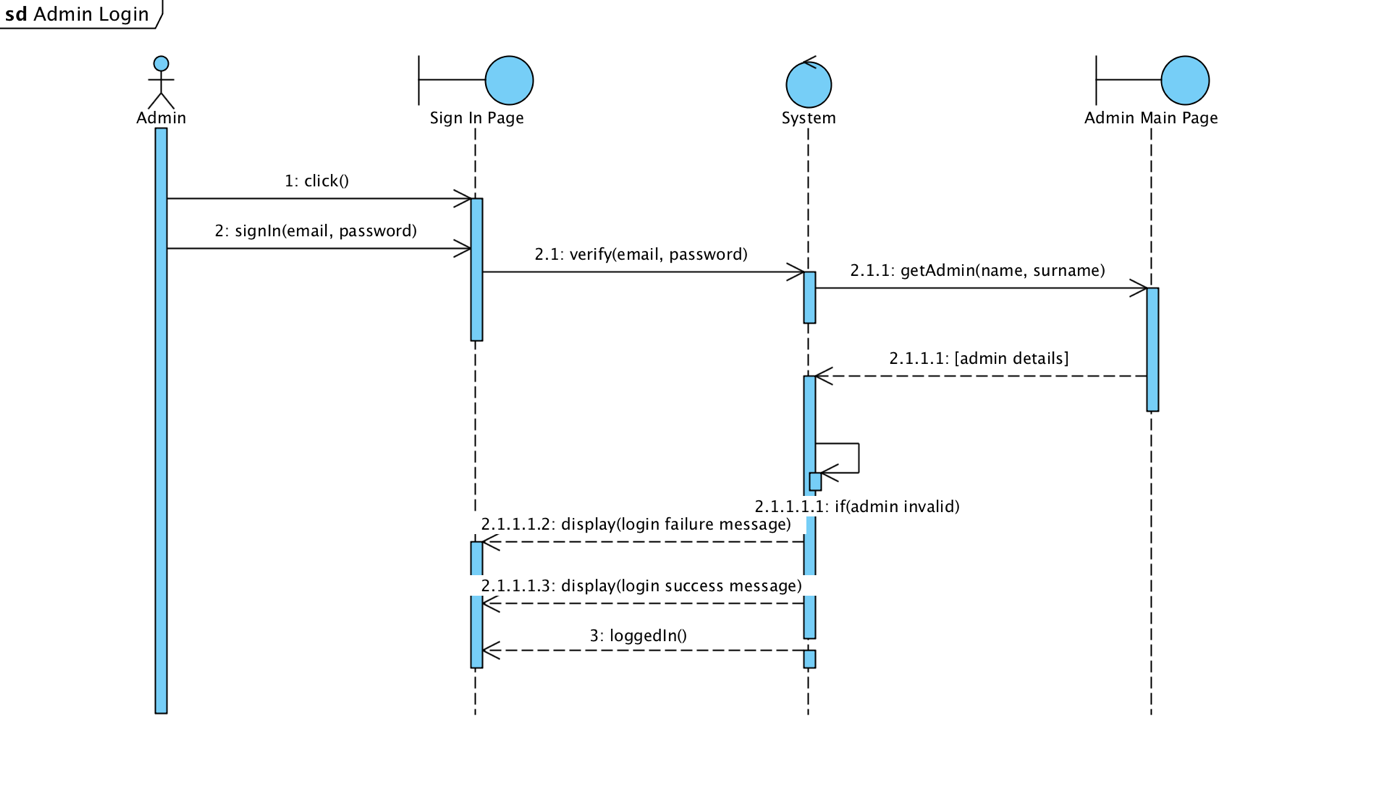
* *Object Model*



**Figure 3. 4. 2 Object Model Diagram**

* *Dynamic Model*

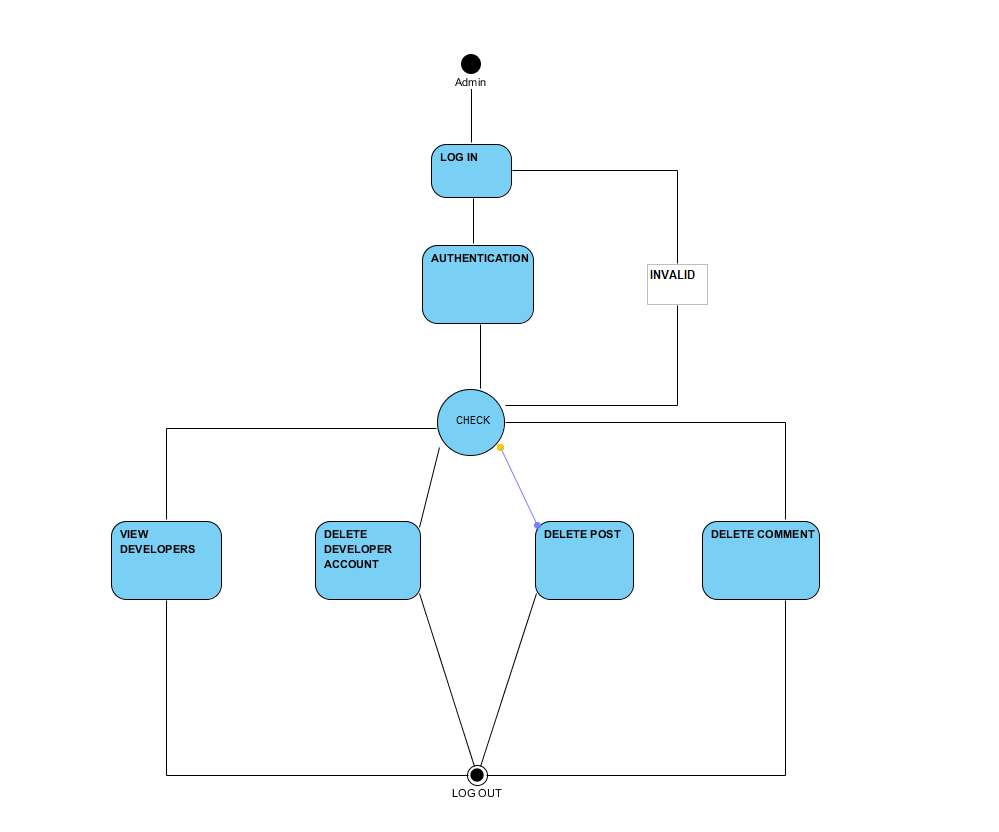
Admin Dynamic Model



**Figure 3. 4. 3 Admin Dynamic Model**

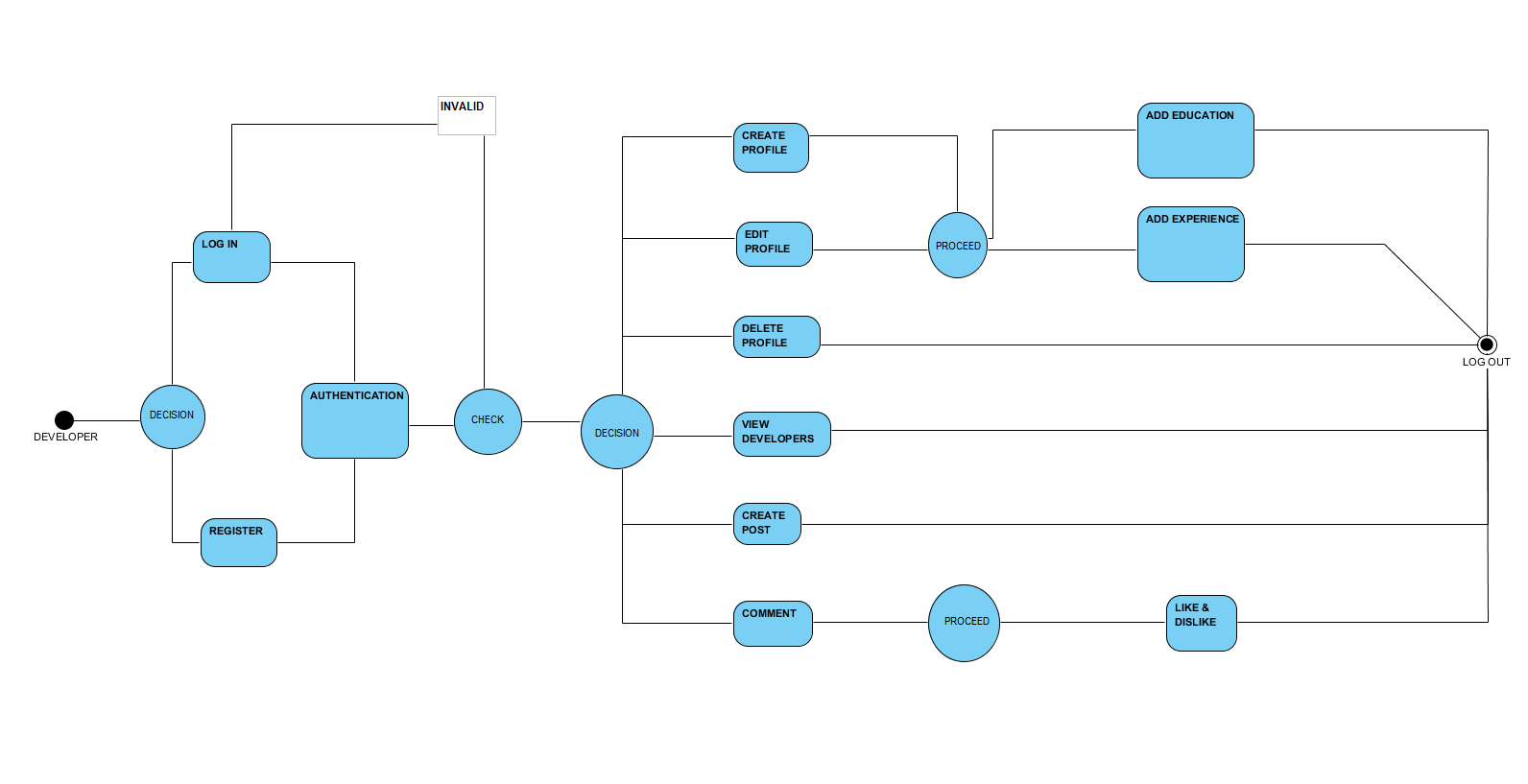
* *Activity Diagrams*

Admin Activity Diagram



**Figure 3. 4. 4 Admin Activity Diagram**

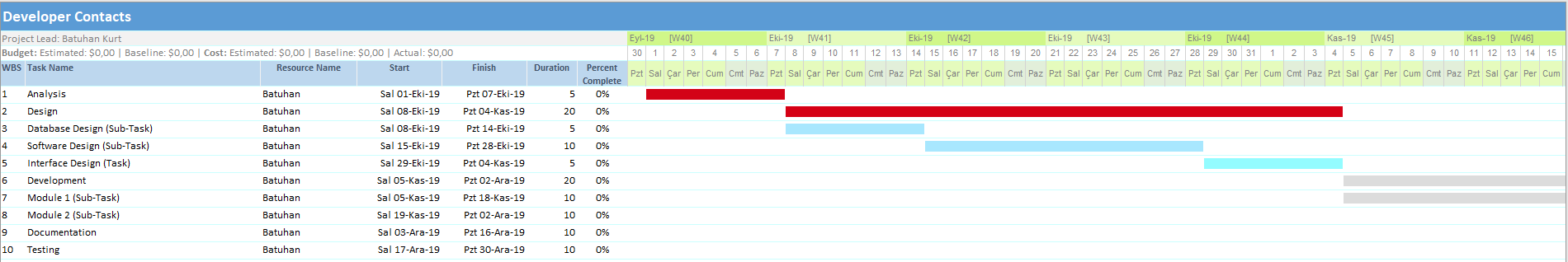
Developer Activity Diagram



**Figure 3. 4. 5. Developer Activity Diagram**

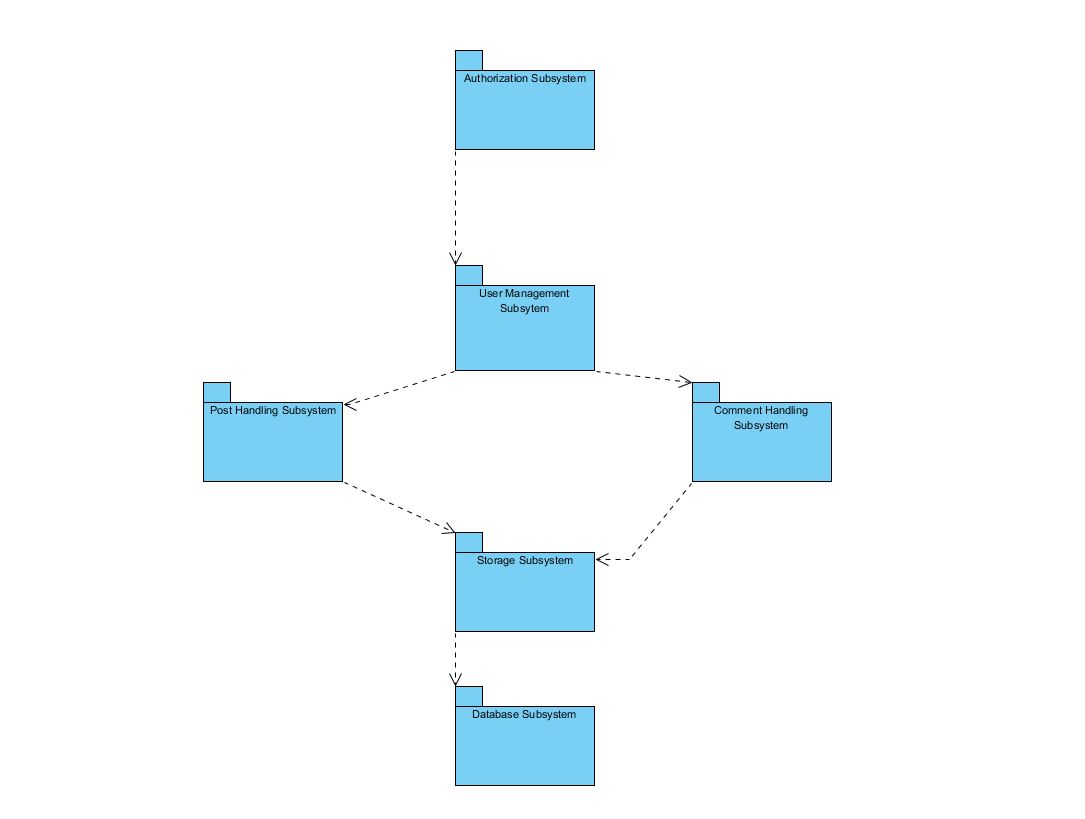
* 1. **Project Schedule**

Project Gantt Chart



**Figure 3. 5. 1. Project Gantt Chart**

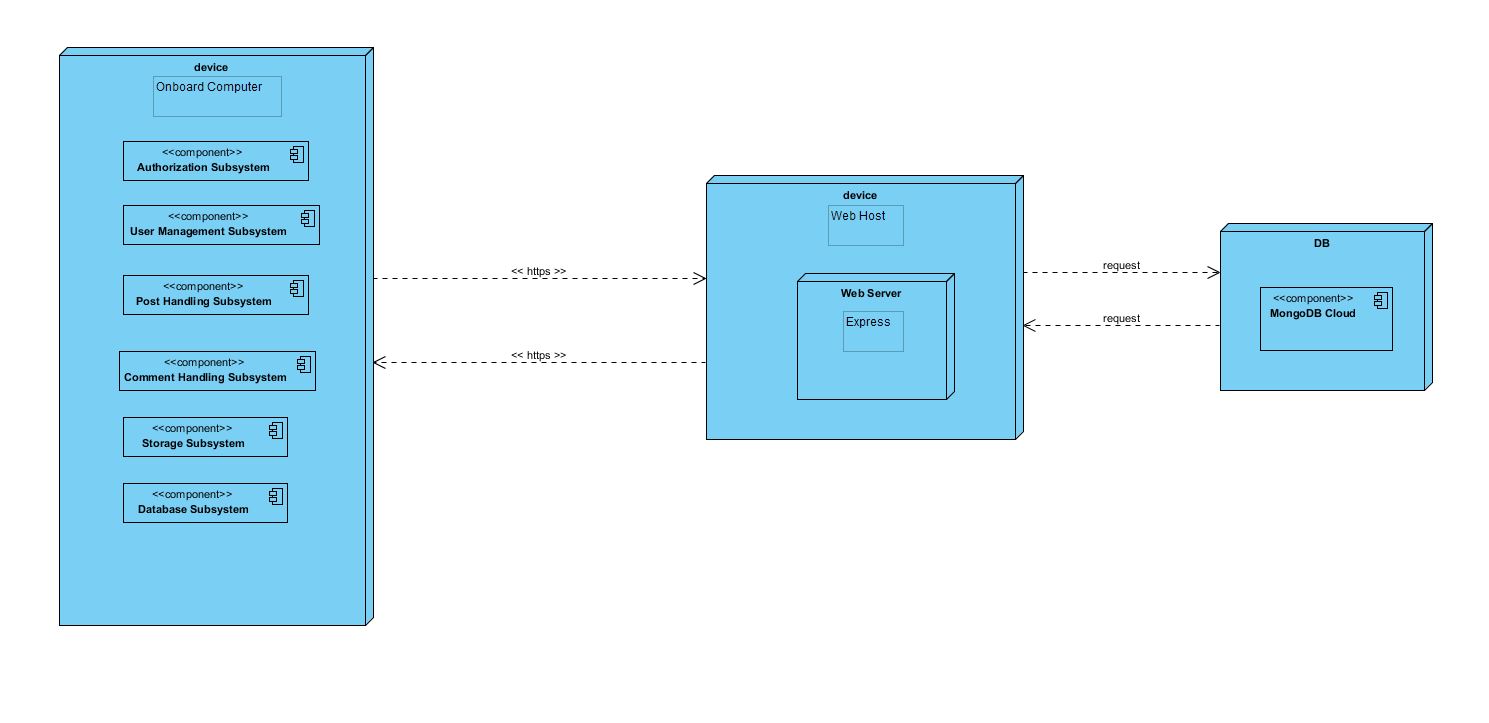
* 1. **System Decomposition**



**Figure 3. 6. 1. System Decomposition**

I have six subsystems in my system. Each one has a unique set of features and functionalities. I achieved this by decomposing my domain model and using the object model. Then, I filtered which were most crucial in achieving the tasks my system requires. My subsystems are as follows:

* *Authentication Subsystem:* This subsystem is fully responsible for user login and register. Authorized persons can access the system through this subsystem. Developers can register with the system through this system.
* *User Management Subsystem:* This subsystem manages user (developer) interactions with the system. It is responsible for allowing the user to create profile, edit profile, delete profile and share portfolio. It is also responsible for adding experience information, deleting experience information, adding education information, deleting education information and creating private developer groups.
* *Post Handling Subsystem:* This subsystem manages user (developer) posts interactions with the system. It is responsible for allowing the user to add post, delete post, like post and dislike post.
* *Comment Handling Subsystem:* This subsystem manages user (developer) comments interactions with the system. It is responsible for allowing the user to add comment, delete comment, like comment and dislike comment.
* *Database Subsystem:* This subsystem is responsible for storing all my data. It is setup in such a way that a storage accesses the database and all other subsystems contact the storage for any modifications or additions to data assets. This is done in an effort to reduce dependency on the database and for security breach issues.
* *Storage Subsystem:* This subsystem is the bridge between the other subsystems and the database. If any subsystem wants to access database for data retrieval or update, the storage provides the service and then it accesses the database itself.
  1. **Hardware Software Mapping**



**Figure 3. 7. 1. Hardware Software Mapping**

I described how subsystems are assigned to hardware and off-the-shelf components. It also lists the issues introduced by multiple nodes and software reuse.

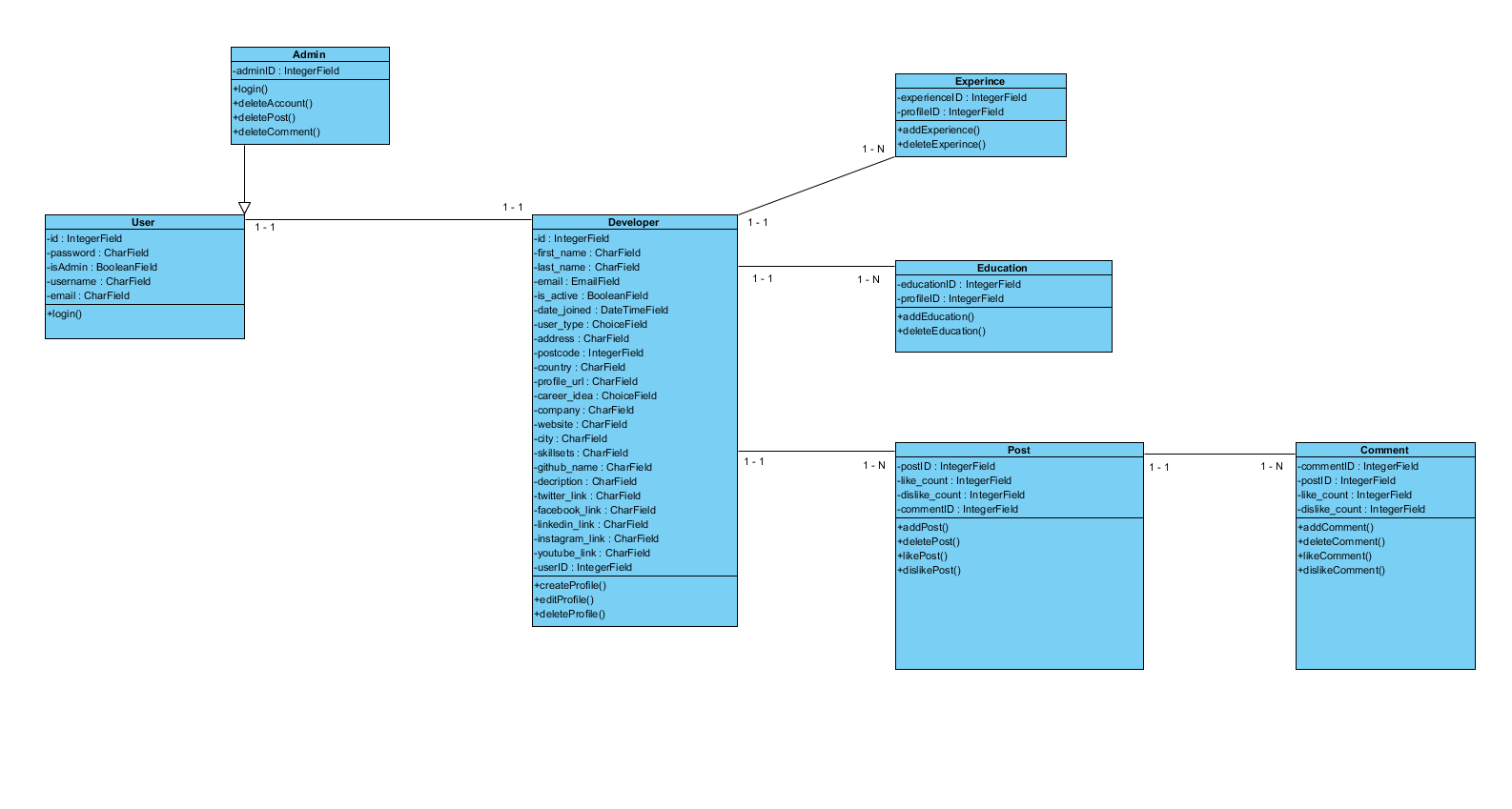
Developer Contacts is a web-based online cooperation system. Developer Contacts is connecting to web server when user (developer) tries to visit and login or register to the system. Developer Contacts system connects to web host with http protocol. Multiple users can access web server simultaneously. Web host have web server. The system connects to database by using JavaScript (Express.js) library.

The Users who is a developer can create portfolio with using the Developer Contacts system. These users communicate with Developer Contacts system with using https protocol in their personal computers web browser. In addition, Developer Contacts uses MongoDB database. This database is running on the cloud. MongoDB is a non-relational database management system. MongoDB is a popular choice for local / client storage applications such as web browsers. It is arguably one of the most preferred deployed database engine, as it is used today by several widespread browsers, operating systems, and embedded systems.

In addition, for the User Interface, I used React.js because it is very popular, agile and open source JavaScript library.

* 1. **Persistent Data Management**

My system uses the Express and Mongoose libraries. These frameworks help to my system to communicate to the MongoDB database. This will allow the database to be easily integrated with and accessed by the rest of the system. The database will retain user (developer) information for functions such as viewing career information of a developer. My database structure is seen below with entity field’s relations etc.



**Figure 3. 8. 1. Persistent Data Management**

* 1. **Access Control and Security**

My system is a multiuser application so it consists of 2 types of users which are developer and administrator. Because of this, the application will provide different interfaces for each user type.

First, the administrator can connect to the system with the membership interface, and will do the administrator's duties, such as adding, updating, editing, deleting etc. with using MongoDB admin interface. As a summary, the administrator does not have to register because he/she is already registered in the Mongo DB database.

The system will store all the information in the database. Then, it will use them by collecting data from the database. The information in the database will use for the confirmation of user types. All types of users must log in to the system with their username and password.

During registration, field filling does not require access to the database, while completion of the process requires the data to be written to the database, which requires read and write access to the database. In that case, the required database fields will be blocked and simultaneous access of multiple users will be denied.

For some situation like updating or deleting an information, it is necessary to update one of the tables in the database in its phase of completion. Therefore, this case must be handled with more care since, several users can be the cause of updating the table at the same time. This will also be avoided by blocking.

Finally, viewing the information or lists again requires read-only access to the database. Therefore, multi-user access does not impose problems and new restrictions.

As a result, the usernames and passwords of users (developers) will be stored in the user table. No one else except the administrator can have access to this information. Authentication interfaces are different for each type of user and will be directed to their own main pages after the login process.

|  |  |  |  |
| --- | --- | --- | --- |
| Actors/  Classes | Admin | Developer | DB Connection |
| Admin | deleteAccount()  deleteComment()  deletePost()  deleteGroups()  viewProfile()  deleteProfile() |  | login()  logout() |
| Developer |  | register()  createProfile()  createGroup()  post()  comment()  like()  dislike()  deleteAccount()  deletePost()  deleteComment()  addExperince()  addEducationInformation() | login()  logout() |

**Figure 3. 9. 1. Access Control and Security**

* 1. **Global Software Control**

My system has MVC (Model - View - Controller) software architecture. Developer Contacts is thread safety but also multithreaded program either because my system must provide many users at the same time to cooperation among the developers.

I have decided to go with event-driven control due to the requirements and nature of my system. The sequencing of actions in my system are directed by an external factor or event generated by an actor to achieve a goal. This makes change in my control structure easy to change and well suited for my system. Because many users can access change and access data in the system and resources are shared, this may creates concurrency issues.

* 1. **Boundary Conditions**

**Initialization**

Startup: Go to system URL and login

Shut Down: Click log out and close browser

**System Failure**

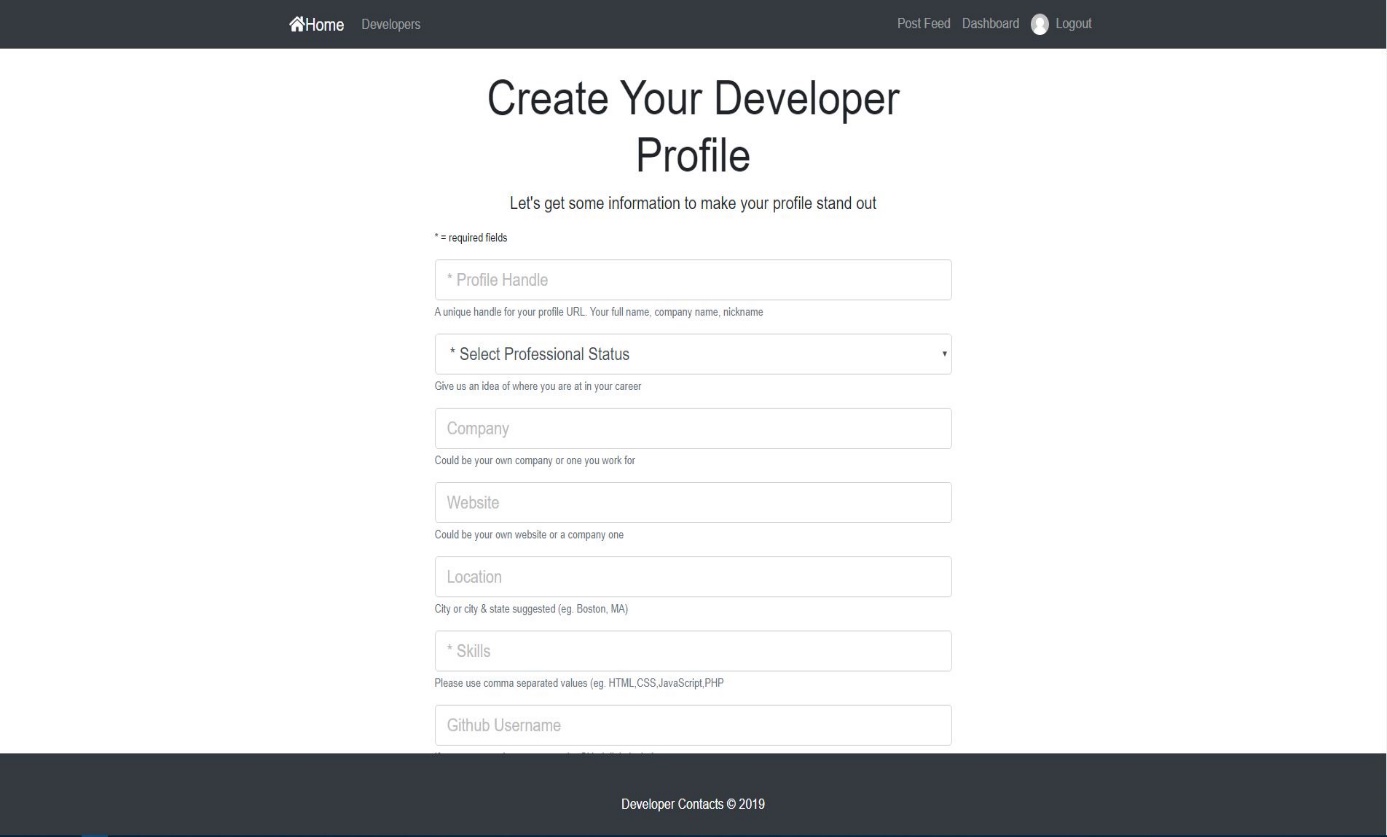
* If database connection is lost, there is a reconnect mechanism which attempts to reestablish the connection to the database.
* Checkpoint system can be applied to prevent database connection problem. System is connected to database and required data is written regularly. If a problem occurs, system will be returned to its previous errorless state.

**Error Conditions**

* Logging in:
* Username or password field is blank.
* Password is not 6 characters long or more.
* Password and username don’t match.
* Username is wrong or does not exist.
* Welcome screen does not appear after logging in.
* Developers Data
* Developer information can not be not exist.
* Developers information can not be blank.
  1. **Subsystem Services**

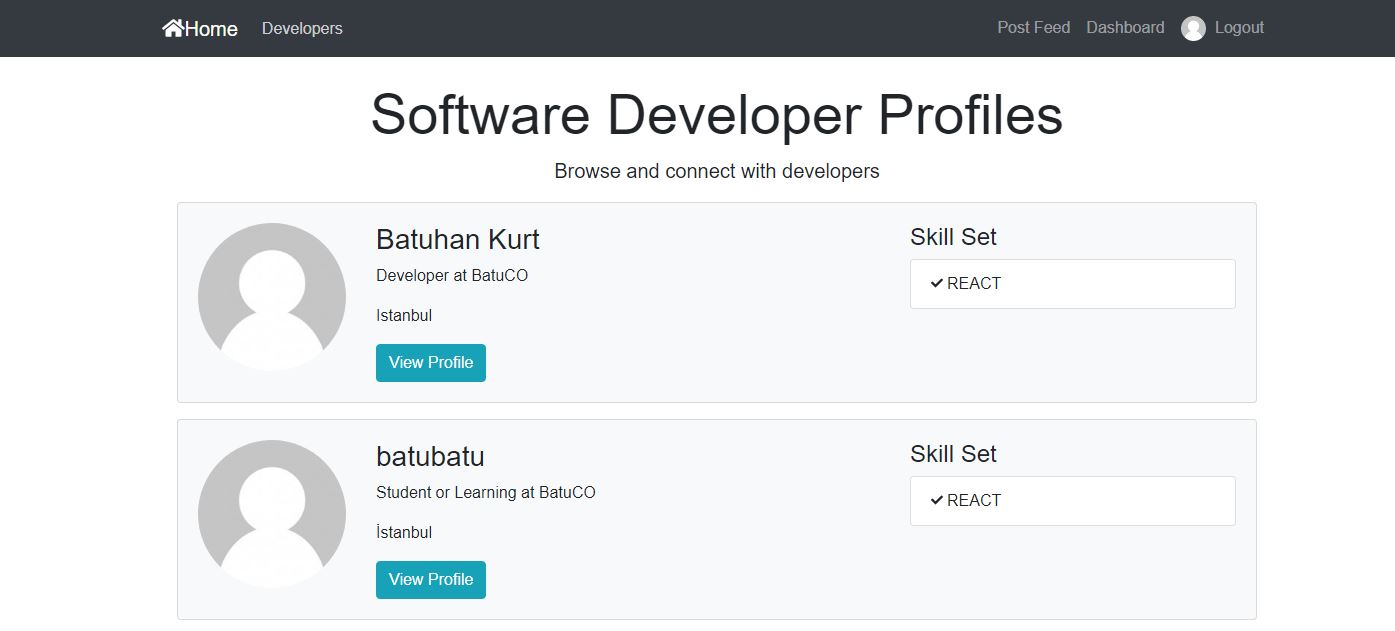
I preferred to divide my system to subsystems to adapt each other themselves and, provide the whole system functionalities.

* **Create Developer Profile Interface**



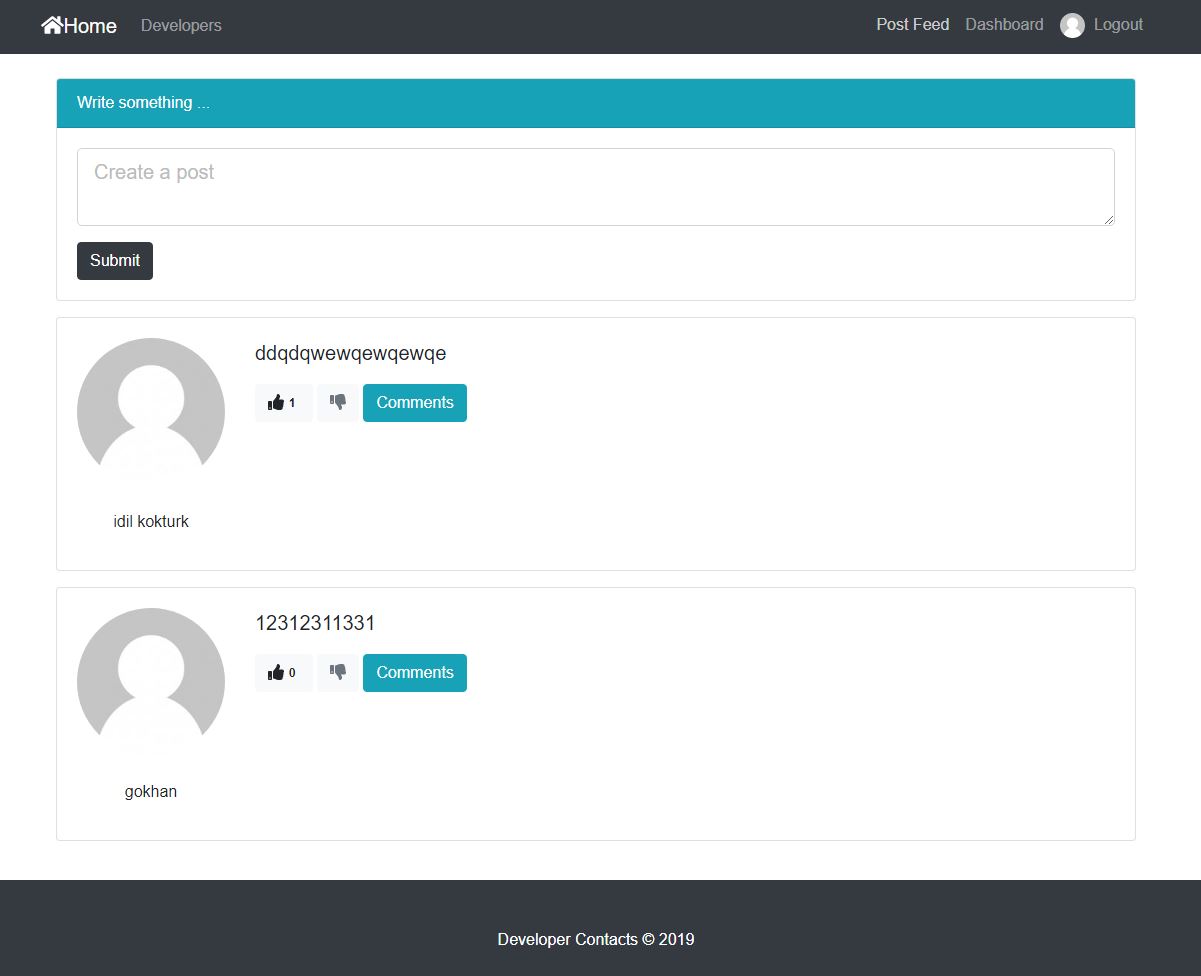
**Figure 3. 12. 1. Create Developer Profile Interface**

* **View Developers Interface**



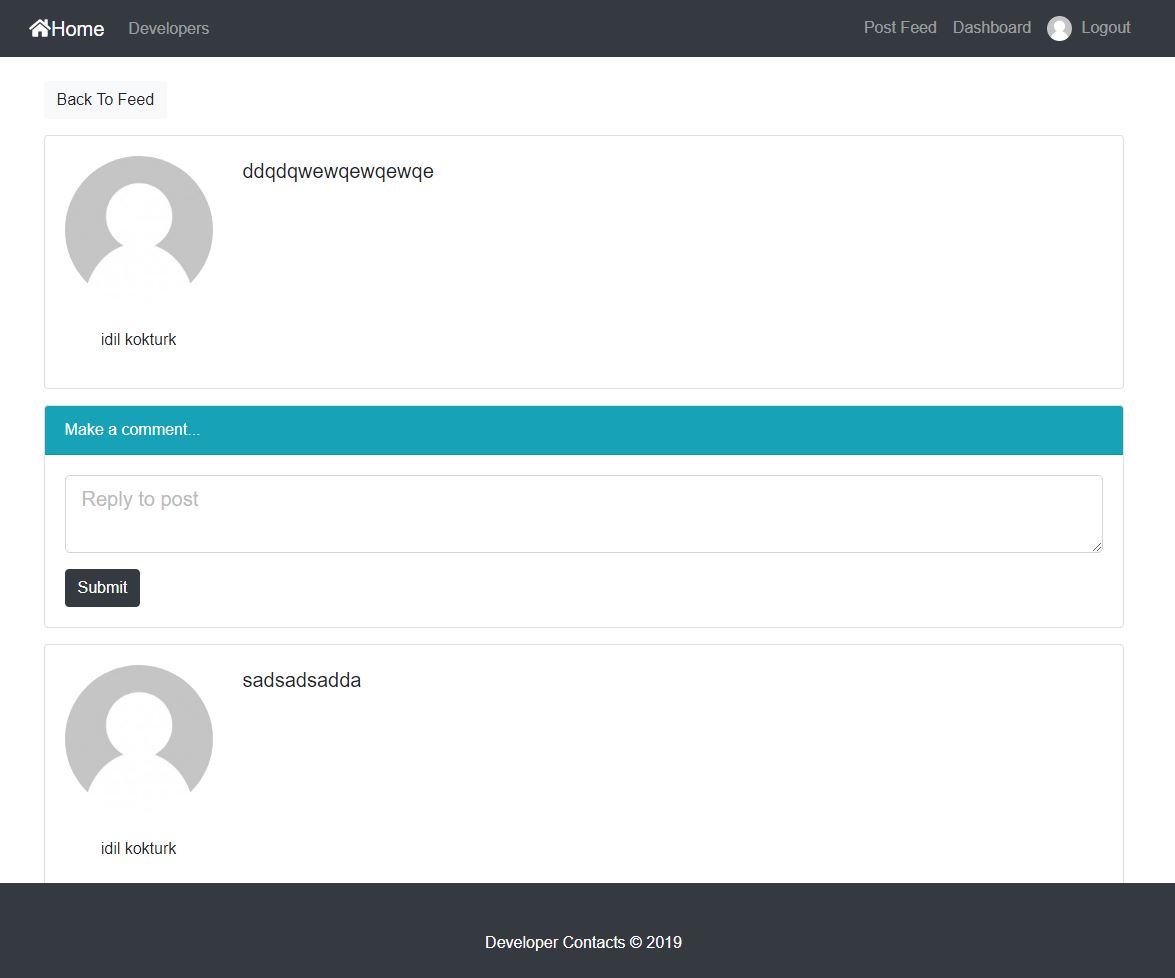
**Figure 3. 12. 2. View Developers Interface**

* **Posts and Like & Dislike Interface**



**Figure 3. 12. 3. Posts and Like & Dislike Interface**

* **Comments Interface**



**Figure 3. 12. 4. Comments Interface**

chapter FOUR

IMPLEMENTATION DETAILS, TESTS, and EXPERIMENTS

Before I started to this project, I have to plan the detailed structure of my database tables and backend logic. So, I want to use Mongo DB for database, because it is one of the fastest and non-relational database. Also, my database works on the amazon web servers on the cloud (cloud.mongodb.com). After that, I have to choose the technologies for backend and frontend developments. I want to create a MERN Stack Application which means MongoDB, Express.js, React and Node.js. I setup the Mongo DB Atlas as first step.

My development platform is Visual Studio Code, so I have installed some dependencies such as mongoose, JSON Web Tokens and basic express setups to this platform.

I used React.js for frontend development, I have learned ES6 notations, Redux and Redux Dev Tools (Redux Chrome Extensions), and React lifecycle methods. Also, I have learned component structure in React. There are HTML and CSS files for pages design. I used Express router for API routing.

Also, I used JSON Web Tokens for authorization. I began to Developer Contacts project with “Sign-in”, “Sign-up” and “Log-out” scenarios. After that, I developed request bodies and validation issues. “Register” scenario is developed for second step. Other steps are that creating dashboard page, post & comment page, portfolio creation page and developers list page.

I used Express.js for server side improvements. Also, I used Mongoose to facilitate Mongo DB operations by creating a model according to the developed application. Mongoose uses “connect” method to reach Mongo DB.

In addition, I create my database models that are posts, users and profiles for using node.js. These are my database tables and of course, these have fields such as ID, name, surname,

e-mail. After that, I create my APIs for CRUD operations to my database.

I used the github for version control system. Last version of my project is deployed to the github. As a result, I used postman for extensive API testing, Google Chrome Developer Tools for UI and API testing and Redux Dev Tools for React Redux developments testing.

chapter FIVE

conclusions and future work

My project which I explained in many ways with this document, gave me valuable information about software production methods in all aspects. Since it does not reach many users now, it was managed with an easier process.

I anticipate that when we reach the number of users like current business to customer applications, performance and quota-based problems will occur. Also, when I was developing this project, I faced the database network access problem. So, I changed the Mongo DB access settings to global. In addition, MERN Stack Application is a new technology, therefore It was difficult to find the tutorials.

It took a long time to find the right answers where I got errors. The main idea of my project was that developers found the right answers by discussing them. Perhaps, thanks to my project, developers will be able to find the answers they are looking for, quickly.

I will add new functionalities such as messaging, live broadcasting etc. to my Developer Contacts project in the future. In addition, I want to design mobile version of my project. In conclusion, I always glad to help people who want to improve themselves, therefore this project was a step for me to achieve my goal.

# 

# REFERENCES

1. Bruegge B. & Dutoit A.H.. (2010). *Object-Oriented Software Engineering Using UML, Patterns, and Java*, Prentice Hall, 3rd ed.
2. <https://www.uml-diagrams.org/composite-structure-diagrams.html>
3. https://www.geeksforgeeks.org/mvc-design-pattern/
4. Enes Taylan, Huseyin Guler, Alperen Eraslan, Omer Durmus (2009). *Billiard Project, Object Oriented Software Engineering.*
5. <http://www.tutorialsteacher.com/mvc/mvc-architecture>
6. <https://www.ibm.com/developerworks/architecture/library/ar-analpat/> (IBM)
7. <https://mindmajix.com/react-js-tutorial#component-api>
8. <https://github.com/facebook/react>
9. <https://mlab.com/>
10. <https://github.com/expressjs/express>
11. <https://github.com/topics/mern-stack>
12. Requirements Analysis Document (15.11.2019)
13. System Design Document (15.12.2019)