

BILKENT UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

CS 299
SUMMER TRAINING
REPORT

Erkin Aydın

22002956

Performed at

İnnova Bilişim

4/7/2022-12/8/2022

Table of Contents

1	Introduction	3
2	Company Information	3
2.1	About the company	3
2.2	About your department	3
2.3	About the hardware and software systems	3
2.4	About your supervisor	4
3	Work Done	4
3.1	Technologies Used	4
3.2	The Project: ConNow!	5
4	Performance and Outcomes	9
4.1	Solving Complex Engineering Problems	9
4.2	Recognizing Ethical and Professional Responsibilities	9
4.3	Making Informed Judgments	10
4.4	Acquiring New Knowledge by Using Appropriate Learning Strategies	10
4.5	Applying New Knowledge as Needed	10
4.6	Awareness About Diversity, Equity, and Inclusion	11
5	Conclusions	11
	References	12
	Appendices	14

1 Introduction

I have done my mandatory summer training in the Ankara office of İnnova Bilişim. The reason I chose this place is that it was a company of Türk Telekom, which I use their technology daily. In addition, İnnova was a relatively big company compared to my many other options, so I thought I had an excellent chance to observe the bureaucracy of such a big company, with its good and bad.[1]

The work I did was a web project that matched the company's standards. I developed a local functioning, meaning that this project did not meet the internet, social media, where users can create posts, comments, like other posts, and so on. I developed backend, frontend, and server-side of this project, meaning that I worked as a full-stack engineer.

Since the project I did was separated from most of the company's work, my work did not have a direct effect on the company, but I've had the chance to work with other interns and collaborate with them to develop my project better and help them to do so.

In this report, I will first give some information about the company I did my internship in, the team I was in, the employees I worked with, and the software and hardware tools I used. Then, I will explain the general structure of my work as much as possible by giving some code and pseudo-code examples, referring to the terms I learned and the tools I used. Later, I will talk about my performance and the outcomes of my work. Lastly, I will give a conclusion concluding what I have learned, experienced, and had the chance to observe.

2 Company Information

2.1 About the company

İnnova is a Türk Telekom group company, mainly focusing on software solutions on various topics, such as finance, insurance, healthcare, education, and on many other topics, both for the private and civil sectors. An example of their product HES (Hayat Eve Sığar) can be shown, which was heavily used during the COVID-19 Pandemic in Turkey to track the disease.[2]

2.2 About my department

The department I worked in was named "Tahsilat ve İş Süreçleri." The name came from the purpose of the software they were developing as engineers rather than what they were dealing with. They are developing and maintaining a software called Payflex İnnova, a software solution for payment and collection of revenues. This product is used in the sector to secure and assure payments.[3]

2.3 About the hardware and software systems

As software, in the backend, Java and Spring Boot (Spring is a popular framework of Java) were used in my department, TTS. In frontend Vuejs, a framework of JavaScript was used. We were using Postman to test the methods we wrote there. I used all these mentioned software in my work, with a couple of additional ones.

One worthy thing to mention in İnnova related to hardware and software was BT(Bakım&Tamir), which was dealing with hardware and software problems of the company's employees.[4] I had to talk to them, primarily when any problem occurred with the hardware given to me, an HP laptop with an i5 intel processor, 8GB RAM, and 128GB SSD.

2.4 About my supervisor

I've had three supervisors, meaning that three company employees checked my work, gave me feedback, and helped me with my internship. One, the team leader, signed my internship report.

Two of them are computer engineers: Mehmet Taha Usta and Mert Suerkan. Mehmet Taha Usta graduated from the Computer Engineering Department of Hacettepe University in the year 2021, and Mert Suerkan graduated from the Computer Engineering Department of TED University in the year 2020. They were relatively new employees of the company. Mehmet Taha Usta was working remotely on my internship, so I had a chance to see how things were working where employees could choose to work remotely or in the office.

The latter supervisor was also the leader of the team I worked with: Çetin Güveme. He graduated from the Mathematics Engineering Department of İTÜ(Istanbul Technical University) in 2000. Çetin Güveme also signed my internship report, as he was the team leader.

3 Work Done

In this part, first, I will talk about the technologies I used and learned. Later, I will explain the structure of my work in the back end.

3.1 Technologies Used

The project I did was a web project, where I used Java, with Spring Boot, in the back end, and Vue.js, a framework for JavaScript, with Vuetify libraries as a framework in the front end. As a database, I used Postgresql. Postman was used to testing the methods and figuring out problems in the code.

Spring Boot

The first and most significant technology I used was Spring Boot. Spring Boot is a microservice framework for Java that creates an ease of production for engineers and developers. It is open-source, meaning that everyone can access its code, use it as they wish, and modify it as they see fit to their needs. In my project, I didn't need to modify the code of Spring Boot; hence I just used it to create a basis for my application and start my project immediately. Spring Boot particularly shines in microservices and web projects, as it helps with its auto-configuration feature where needed. And, where Spring Boot is used, there is no need to use any external servers as it comes with an "embedded server" during initialization. In summary, Spring Boot is a tool developers and software engineers can use to start creating web applications and microservices without losing time and energy with things like configuration and compatibility.[5]

The back end of this project will be explained thoroughly, with references to Spring Boot, as consistency and pattern between classes can be followed.

Vue.js and Vuetify

The second major technology worth mentioning is Vue.js. Vue.js, as predicted by its name, is a JavaScript framework. It is entirely open-source. Hence you can modify it as you see fit. One of the essential features of Vue.js is that it lets you gather all the necessary code for frontend, HTML CSS, and JavaScript in single files. Hence, it reduces the additional overhead and complexity of moving between files. There are plenty of modifications done on default CSS and HTML attributes done by Vue.js, and also many additional features that help engineers, such as loops that can be used to traverse an array of JSON files as an HTML attribute: v-for. Learning Vue.js is a lot easier than learning other possible choices, such as React.js, which was a game changer for me as I had a limited amount of time with my internship. It is also worth mentioning Vuetify, which is an open-source user interface(library) that has Material Components explicitly created for Vue.js. The front-end part of this project will not be explained in detail, as every page is designed differently, and there is no consistency or pattern between them.[6][7]

PostgreSQL

The database that I used for my internship is PostgreSQL. PostgreSQL is a database system developed by Google. As the rest of the technologies are, it is entirely open source. Where this database shines is when the data stored in the database gets big and complex, as it is one of the most complex databases in use, if not the most. Even though I did not have anything to do with the details on how PostgreSQL works, a comparison between PostgreSQL and MySQL can show why PostgreSQL is this good at complexity. For example, while MySQL only supports standard data types, PostgreSQL supports advanced data types, such as arrays. While such properties give PostgreSQL an edge where complexity is present, it also makes PostgreSQL more challenging to troubleshoot, as it is more complex in itself.[8]

Postman

Postman is an application where you can test methods written in an application by sending appropriate requests, like POST, PUT, GET, DELETE, etc., to controller classes. This application was used alongside the IDE, IntelliJ Ultimate, used in this project to test every method written, step by step.[9]

3.2 The Project: ConNow!

ConNow! was designed to be a simple social media-like site where users can create accounts, post text with a title, make comments on posts of other users, and like and dislike posts and comments.

The backend was written with Java and Spring Boot, so it was designed to follow the principles of OOP(Object-Oriented Programming) as much as possible. Later on in the project, I started to fit my classes into an MVC(Model View Controller) design using controller classes. In this part, the backend will be explained by referring to the meaning of code, annotations used to communicate with Spring Boot, and so on. Understanding the functionality of these annotations will get progressively easier. I created seven packages to separate classes with different purposes and group classes with the same goals. These packages are the following: entities, controllers, services, repos(stands for repositories), requests, responses, and lastly, the email package. There is also an incomplete package created explicitly for ApiResponse: common.[10]

The purposes of these packages require explanation. Some of them will be explained here.

Entities

The entities package contained the backbone classes that users see and interact with: User class, Post class, Comment class, and lastly, Like class.[11]

The User class had four properties: a unique id for each user to separate users, an email and a password for registration, login, and security purposes, and a unique username. [12][20]

As can be seen in [20], there are some annotations used both for the class and its properties. These annotations function as tools to communicate Spring Boot and tell it that we need some type of help with configuration and writing our code. For example, the `@Entity` annotation of the user class states that this class is an entity, the `@Table(name = "users")` tells Spring Boot that a table with the name "users" should be created in the local database, and the `@Data` with annotation, Spring Boot automatically configures getter and setter methods for properties, so we don't spend time writing them and simply continue with the rest of the code. There are also written annotations for the id property: The first annotation, `@Id`, tells Spring Boot that this property will function as an ID, and hence it should be treated as. The `@GeneratedValue(strategy = GenerationType.IDENTITY)` annotation configures this id property to be auto-incremented, meaning that in the creation of a User object, the id of the new User will be one higher than the previously created one, and thus it will be unique for sure. At first, in the production stage, the strategy to generate ID was different in my code: it needed to be entered manually. Therefore, some parts of my code related to IDs did not make any sense but still remained there as they were not creating any problems, and I didn't have enough time to change and test the code again.

The Post class had four properties: A unique id, a title for the post, a text explaining the context of the post, and the user that posted it.[13][21]

The different annotation for the Post than the User class is the `@Table(name="post")` annotation, meaning the auto-generated table for the Post class in the database will be named "post."

The IDs of Post objects are created as they are for User objects. The `@Column(columnDefinition="text")` of the text property states that the column name of this property in the Post table generated in the database should be "text."

Annotations above the user property require careful examination. In Spring Boot, to explain the relationship between objects, there are four relations: ManyToOne, OneToOne, OneToMany, and ManyToMany. In this instance, the `@ManyToOne(fetch = FetchType.EAGER)` annotation is used as many posts can be posted by one user. If a user could not post more than one post, then the `@OneToOne` annotation should have been used. The `(fetch = FetchType.EAGER)` part of this annotation tells Spring Boot that the compiler should fetch the references as the application starts. This would add overhead when first booting the application. The other option is to use `(fetch = FetchType.LAZY)`, which was used in the Comment class. If it were used here, it would remove the overhead, but this time since the user is not pre-fetched whenever we need to access the user property of a post, we would have additional overhead.[21]

The other annotation used for the user property is `@OnDelete(action = OnDeleteAction.CASCADE)`, which tells Spring Boot that when the user is removed from repositories, its posts are to be deleted automatically. This prevented me from deleting posts posted by users one by one whenever I deleted a user.

The rest of the entities, Like and Comment entities, work incredibly similarly to User and Post entities. Hence they do not require further explanation. If wished, the Comment class given in [22] can be examined.[14][15]

Repositories

The repos package contained interfaces that could be used to find entities with specified methods.[16]

The most important thing here is that this package had no implementation classes. Thanks to Spring Boot, just writing a prototype of a method was sufficient, as Spring Boot auto-implemented the method in the background concerning the name of the method. Each repository extended `JpaRepository` and had the `@Repository` annotation, telling Spring Boot that this is a repository and its methods should be auto-implemented.

Services

The services package contains classes and interfaces used to interact with entities. Every service connects to the entities through controller repositories, and services are reached through controller classes, which will be explained later.[17]

Every interface in this service package contains the prototypes of necessary methods to interact with entities, and implementation classes of these interfaces contain implementations of these methods. For example, changing the title of a post first requires searching the post in the post repository, and if it is found, its title can be changed. If not, necessary actions should be taken accordingly.

Services, the way they work, and their purposes will be explained through the `CommentService` interface and `CommentServiceImpl` class, the implementation of the `CommentService` interface. The `CommentService` interface contained six method prototypes: `getAllComments`, `getAllCommentsByPostId`, `getOneCommentById`, `createOneComment`, `updateOneCommentById`, `deleteOneCommentById`. The `CommentServiceImpl`, the implementation of the `CommentService` interface, implements these methods. For instance, the implementation of `getAllComments` can be a starting point. This method is used to return a list of all comments. Comments can be obtained through `commentRepository`. Hence, the `findAll` method of `commentRepository`, which returns a list of all comments in the repository, is directly used, and the result is returned.[23]

A slightly more complicated example is `deleteOneCommentById` method. This method takes the comment's id as a parameter and deletes the comment with that id. As every comment has a unique id, no more than a comment is ever deleted with this method. First, it is necessary to check whether there is a comment with the parameter id in the repository because if not, we can't delete any comment. Hence, `commentRepository.findById(commentId).isEmpty()` is used as the condition. If `commentRepository.findById(commentId)` returns a comment with that id, then the `isEmpty()` method will return false; hence the condition will be unsatisfied, and the deletion process will start. If the condition is false, an error message is returned stating that no such comment exists.[24]

As the last example of how services work, `createOneComment` can be examined. The unique thing about this method compared to other examples is that it takes a request (the request package, its purpose, and its classes will be explained later on) that carries the necessary information for creating a comment, `commentCreateRequest`, as a parameter, and proceeds accordingly. To create a comment, the user to post this comment, and the post that this comment will be posted should exist. Otherwise, the comment can't be posted. The first two lines of this method's body check these conditions. It was also found necessary to check whether a comment with the ID carried with the `commentCreateRequest` already exists during coding, but this turned out to be unnecessary as I automatized the strategy of generating ids of entities. If no problem occurs with these constraints, the comment is created and saved to the repository, and a success message is returned.[25]

The rest of the services and their implementations follow the same logic as `CommentServiceImpl`: Check corner cases; if a problem occurs, return an error message explaining why it is impossible to fulfill the duty, and if no problem occurs, fulfill the duty and return a success message.

Controllers

The controllers package is the way to contact the service classes.[18]

This package links the front end with the back end via mappings, such as `@GetMapping`, `@PostMapping`, and such annotations. Controller classes are the classes executed in a local link, such as `"/users"` or `"/comment"`, and each method of them has a unique link that extends the classes' links, such as `"/users/list"` or `"/comment/list"`. These unique links are delivered to Spring Boot via mappings. In my controllers, 4 mappings were enough: `@GetMapping`, `@PostMapping`, `@PutMapping`, and `@DeleteMapping`.

The `UserController` class can be used to give examples of these mappings. `@GetMapping` is used for getter methods, such as the `getAllUsers` method.[26]

`@PostMapping` is used for methods that create an object and save it to repositories, such as the `createUser` method.[27]

`@PutMapping` is used for setter methods, where properties of an already existing object are to be changed, such as `updateOneUser` method.[28]

`@DeleteMapping` is used where the deletion of an object from repositories is to take place, such as `deleteOneUser` method.[29]

These mappings and their request links were used with Postman to test every method and see whether they were working correctly or not.

Requests

Before this package was created, where an object was to be created or updated, either each parameter had to be given separately to service methods, or the object should've been created and passed as a parameter. This created security issues and did not follow Object-Oriented Programming principles. For example, when a user changes his/her password, the service layer does not need his/her id, but if we create a user object with new information, its id is also passed to the service layer. To prevent this from happening, request classes were used.[19]

4 Performance and Outcomes

4.1 Solving Complex Engineering Problems

As far as I saw in my internship, one of the most complex parts of creating reliable and consistent software is not coding but planning. For example, I didn't start my project by laying all the possible problems on the table and figuring out ways to solve them, or I didn't create a plan by using UML diagrams. I just thought about the project and started coding. Later on, this created many problems, like when I couldn't predict not using Request classes would create security issues. When I saw this issue, I had to change my code a lot, and during this change process, the code I wrote looked like mumbo jumbo. There, I experienced the difference between being a developer and an engineer.

One other problem was choosing the right technologies. At first, I wanted to use React.js as my JavaScript framework for the front end of this project. If I had known that the learning curve of React.js was too high, I could've avoided unnecessary complexity. In the middle of the project, I had to change my framework for JavaScript.

Another part of my work, which I didn't fail, was documenting my work. In this project, I have implemented tens of classes and methods. I have recognized early on that if I don't document my work using Javadoc comments, I will have trouble understanding what I have written over time and fixing problems. My usage of Javadoc comments prevented extra complexity I could have if I didn't use them, which I found to be a crucial part of Software Engineering.

A noteworthy example of solving complexity was using service and controller layers. At first, I didn't understand what they accomplished, and I thought everything these two layers have done could be done with one layer. But using two layers made the code much more understandable and simple and prevented further complexity. It also adds one more layer, which means that if an error occurred somewhere in my code, it would be easier to detect its place since these layers are separate.

Lastly, I would like to mention a compatibility issue that created complexity. Due to a problem we couldn't solve with Mert Suerkan, one of my supervisors, I have abandoned using Vue3 and downgraded to Vue2, the previous version of Vue. Later on, it turned out that Vue3 was still buggy and incompatible with one other software I used, Spring Boot, so I did the right thing by choosing the more reliable and compatible version of Vue.js. This showed that theoretical knowledge, using the most up-to-date and performant versions of software, for instance, sometimes created problems, that there is no specific way to solve complexities, and that they are inevitable sometimes.

4.2 Recognizing Ethical and Professional Responsibilities

The first ethical and professional responsibility I recognized was not preventing colleagues from doing their duties. Yes, collaborating is an essential part of work life, but the moment collaboration exceeds its lines and becomes a dependency, it is a problem. I tend to ask questions on topics before searching for answers myself properly. I, unfortunately, might have asked too many questions at times, and this may have prevented employees from doing their job at times. When I realized this, I tried to search for answers by myself more.

One other thing I observed in İnnova is that employees always stuck to the schedule of their teams, and when they couldn't, they reported it to their team leader, and the team leader informed other employees in the team. This not only created an order and a feeling of duty but also created a feeling of responsibility to colleagues. Little things such as coming to the office, or if you are working remotely, being present in the zoom meeting on time were crucial for communication and trust. So, an engineer does not only write code or present reports but also looks up to their team members.

Lastly, not only the employees had responsibilities to the company, but also the company had to take care of their employees. For example, there were periodic checks and presentations about health made by doctors at İnnova. One thing I found interesting is that vending machines were automatized so that they didn't need money to give the snack you wanted. This was taken as a measure to prevent employees from pushing buttons more frequently and spreading COVID-19 during the pandemic.

4.3 Making Informed Judgments

When I first met with the team, I realized that the work I would have done there wouldn't be qualified as a Computer Engineering internship. Hence, I demanded to change my team, which they didn't want to do at first but later accepted. Then, I met the team that I worked with on my project, "Tahsilat ve İş Süreçleri." If I didn't detect that this would be a problem at first sight, my whole internship would be in danger.

In this report and in the mfstaj system, my internship contains 6 weeks, whereas the second week of it was a holiday. However, I also lost the first week of it due to my troubles with my first team. Thus, I did my internship in 4 weeks, in 20 work days, still satisfying the number of mandatory internship days. This is why the report delivered by İnnova has a different beginning date than stated in this report.

4.4 Acquiring New Knowledge by Using Appropriate Learning Strategies

To accomplish my goals in this project, I had to understand the technologies that I didn't know before. There was no textbook or direct way of learning these technologies. The most reliable way to learn them when they were changing on a monthly basis was to read the official documentation. Hence, I have spent my first week not actually coding for the project but reading the documentation of Spring Boot and trying different possibilities on how to use the annotations that I explained in part 3. Where the documentation failed to provide necessary information to my questions, I have used websites like StackOverflow, which usually contained the answers to my questions.

Another way of acquiring new knowledge was by asking my team members, as they were using the same technologies as I did. This allowed me to understand the technology and what I should do with it. The team member next to me has already been in my place once and, thus, understood me and my problems much better than a website.

4.5 Applying New Knowledge as Needed

I didn't need to use everything I had learned from documentation and/or my team members in this project. Nearly always, there were multiple ways to solve a problem I faced, so I had to be selective. My general approach when such a problem appeared

was to use the most up-to-date technology. However, this didn't always work, as I explained in part 4.1.

One example of applying the knowledge I have learned was the usage of annotations. These annotations saved me so much time with their auto-configuration features. Some of those auto-configurations were unimplementable in the given amount of time, and when I saw the need, I used the annotations without wasting time.

The most noteworthy example of applying new knowledge would be the usage of Vue.js. I had to learn Vue.js from scratch for the front end to work. A backend without a front end would mean nothing for my project as I was developing a web application.

4.6 Awareness About Diversity, Equity, and Inclusion

The company where I did my internship, Innova, was in Turkey, and it is Turkey-based. Hence, I did not have much of a chance to observe diversity. What I saw, though, was that there were many women software engineers and computer engineer graduates that were working both in my team and other teams, so there was no exclusive policy based on gender in Innova. Additionally, employees of Innova were quite inclusive to us, interns and newbies. Nobody tried to humiliate or ignore us, and I always got satisfying answers to my questions from them. It didn't matter whether I asked one of my supervisors or someone randomly present in the office. We always communicated on equal ground.

5 Conclusions

I have learned a lot of new technologies in Innova, the most notable one is Spring Boot. I saw the connections between technologies, similarities and differences on how they work. I understood the value of frameworks, as they increased my productivity by multiples. Another thing I found valuable in my internship is that developing a system is not the same as engineering it. Planning ahead and applying certain strategies that help you write code gets more important as the codebase gets bigger. Lastly, I saw the value of constantly learning. Engineering, particularly software and computer engineering, change fast and it is necessary to keep your knowledge up to date.

References

- [1] "İnnova Bilişim Hakkında".
<https://www.innova.com.tr/tr/hakkimizda/innova-hakkinda>. [Accessed: Oct 20, 2022].
- [2] "Webrazzi Ödülleri'nde "Yılın Mobil Uygulaması" kategorisinde "Hayat Eve Sığar" uygulamamıza 2.'lik ödülü".
<https://www.innova.com.tr/tr/hakkimizda/oduller/webrazzi-odulleri>. [Accessed: Oct 20, 2022].
- [3] "Ödeme, Tahsilat, Mobil Uygulama ve Sadakat Çözümleri".
https://www.innova.com.tr/download/files/3002201405tr_payflexgenel_web_5874819891.pdf. [Accessed: Oct 20, 2022].
- [4] "BT Yönetim Çözümleri".
<https://www.innova.com.tr/tr/altyapi-operasyon/it-servisleri-yonetim-sistemleri/bt-yonetim-cozumleri>. [Accessed: Oct 20, 2022].
- [5] "Spring Boot".
<https://spring.io/projects/spring-boot>. [Accessed: Oct 20, 2022].
- [6] "Vue.js - The Progressive JavaScript Framework".
<https://vuejs.org/>. [Accessed: Oct 20, 2022].
- [7] "Vuetify - A Material Design Framework For Vue.js".
<https://vuetifyjs.com/en/>. [Accessed: Oct 20, 2022].
- [8] "PostgreSQL: The World's Most Advanced Open Source Relational Database".
<https://www.postgresql.org/>. [Accessed: Oct 20, 2022].
- [9] "Postman API Platform".
<https://www.postman.com/>. [Accessed: Oct 20, 2022].
- [10] "ConNow".
<https://github.com/Erkin-Aydin/ConNow/tree/master/src/main/java/com/project/questapp>. [Accessed: Oct 20, 2022].
- [11] "ConNow Entities".
<https://github.com/Erkin-Aydin/ConNow/tree/master/src/main/java/com/project/questapp/entities>. [Accessed: Oct 20, 2022].
- [12] "ConNow User Class".
<https://github.com/Erkin-Aydin/ConNow/blob/master/src/main/java/com/project/questapp/entities/User.java>. [Accessed: Oct 20, 2022].
- [13] "ConNow Post Class".
<https://github.com/Erkin-Aydin/ConNow/blob/master/src/main/java/com/project/questapp/entities/Post.java>. [Accessed: Oct 20, 2022].
- [14] "ConNow Comment Class".
<https://github.com/Erkin-Aydin/ConNow/blob/master/src/main/java/com/project/questapp/entities/Comment.java>. [Accessed: Oct 20, 2022].

- [15] “ConNow Like Class”.
<https://github.com/Erkin-Aydin/ConNow/blob/master/src/main/java/com/project/questapp/entities/Like.java>. [Accessed: Oct 20, 2022].
- [16] “ConNow Repos”.
<https://github.com/Erkin-Aydin/ConNow/tree/master/src/main/java/com/project/questapp/repos>. [Accessed: Oct 20, 2022].
- [17] “ConNow Services”.
<https://github.com/Erkin-Aydin/ConNow/tree/master/src/main/java/com/project/questapp/services>. [Accessed: Oct 20, 2022].
- [18] “ConNow Controllers”.
<https://github.com/Erkin-Aydin/ConNow/tree/master/src/main/java/com/project/questapp/controllers>. [Accessed: Oct 20, 2022].
- [19] “ConNow Requests”.
<https://github.com/Erkin-Aydin/ConNow/tree/master/src/main/java/com/project/questapp/requests>. [Accessed: Oct 20, 2022].

Appendices

[20]

```
@Entity
@Table(name = "users")
@Data
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String email;
    private String username;
    private String password;

}
```

Figure 1-User Class.

[21]

```
@Entity
@Table(name="post")
@Data
public class Post {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    public String title;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name="user_id", nullable=false) // user can't be null
    @OnDelete(action = OnDeleteAction.CASCADE) //when a user is deleted, all the posts of its are also deleted
    public User user;

    @Column(columnDefinition="text")
    public String text;

}
```

Figure 2-Post Class.

[22]

```
@Entity
@Table(name="comment")
@Data
public class Comment {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY) //One post can contain many comments
    @JoinColumn(name="post_id", nullable=false) // post can't be null
    @OnDelete(action = OnDeleteAction.CASCADE) //when a post is deleted, all the comments of its are also deleted
    @JsonIgnore
    public Post post;

    @ManyToOne(fetch = FetchType.LAZY) //One user can have many comments
    @JoinColumn(name="user_id", nullable=false) // user can't be null
    @OnDelete(action = OnDeleteAction.CASCADE) //when a user is deleted, all the posts of its are also deleted
    @JsonIgnore
    public User user;

    @Column(columnDefinition="text")
    public String text;

}
```

Figure 3-Comment Class.

[23]

```
/**
 * Gets all the existing comments in commentRepository.
 * @return all the existing comments in commentRepository. If no comments exist, then return an empty list.
 */
public List<Comment> getAllComments() {
    return commentRepository.findAll();
}
```

Figure 4-getAllComments Method.

[24]

```
/**
 * This method is used to delete a comment.
 * @param commentId id of the comment to be deleted
 * @return "Success" if successful, "Failed: No such comment exists!" if no comment found with the id.
 */
public String deleteOneCommentById(Long commentId) {
    if(commentRepository.findById(commentId).isEmpty()) {
        return "Failed: No such comment exists!";
    }
    else {
        commentRepository.deleteById(commentId);
        return "Success";
    }
}
```

Figure 5-deleteOneCommentBy Method.

[25]

```
/**
 * This method is used to create a comment under a post
 * @param commentCreateRequest carries credentials of the upcoming comment.
 * @return "Success!" if it is created successfully, a fail message with its reason if not.
 */
public String createOneComment(CommentCreateRequest commentCreateRequest) {

    Optional<User> user = userService.getOneUser(commentCreateRequest.getUserId());
    Optional<Post> post = postService.getOnePostById(commentCreateRequest.getPostId());
    //If the user to comment, or the post to be commented, does not exist, we can't create te comment.
    if(user.isEmpty()) {
        return "Failed: User does not exist!";
    }
    else if(post.isEmpty()) {
        return "Failed: Post does not exist!";
    }
    else if(commentRepository.findById(commentCreateRequest.getId()).isPresent()) {
        return "Failed: A comment with the same id already exists!";
    }
    //If they exist, then we create the comment accordingly.
    else {
        Comment toSave = new Comment();
        toSave.setId(commentCreateRequest.getId());
        toSave.setText(commentCreateRequest.getText());
        toSave.setUser(user.get());
        toSave.setPost(post.get());
        commentRepository.save(toSave);
        return "Success!";
    }
}
```

Figure 6-createOneComment Method.

[26]

```
/**
 * This method returns the list of users in the userRepository
 * @return List of users in the repository.
 */
@GetMapping("/list")
public UserResponse[] getAllUsers() {
    //return userService.getAllUsers();
    List<User> users = userService.getAllUsers();
    UserResponse[] userResponses = new UserResponse[users.size()];
    for(int i = 0; i < users.size(); i++) {
        UserResponse newUserResponse = new UserResponse();
        newUserResponse.setUserName(users.get(i).getUsername());
        userResponses[i] = newUserResponse;
    }
    return userResponses;
}
```

Figure 7-getAllUsers Method.

[27]

```
/**
 * This method will create a user with the given user information through createRequest. It returns "Success!" if such
 * a user with incoming createRequest.getId() does not exist in the system and creates a user with these credentials.
 * Returns "Failed: User with the given id already exists." otherwise.
 *
 * @param createRequest is the user information of the incoming user
 * @return newUser
 */
@PostMapping("/create")
public ResponseEntity createUser(@RequestBody UserCreateRequest createRequest) {

    try {
        userService.createUser(createRequest);

        ApiResponse apiResponse = new ApiResponse("Success!", true);
        return new ResponseEntity(apiResponse, HttpStatus.OK);
    } catch (Exception e) {
        ApiResponse apiResponse = new ApiResponse("Failed!", false);
        return new ResponseEntity(apiResponse, HttpStatus.CONFLICT);
    }
}
```

Figure 8-CreateUser Method.

[28]

```
/**
 * This method updates the user with the parameter userId with the parameter updateRequest. Returns the
 * @param email is taken as a PathVariable, the email of the user to be updated.
 * @param updateRequest as the new credentials of the user(id can't be updated!!!)
 * @return "Success!" if successful, a fail message if not.
 */
@PutMapping("/{email}")
public String updateOneUser(@PathVariable String email, @RequestBody UserUpdateRequest updateRequest) {
    return userService.updateOneUser(email, updateRequest);
}
```

Figure 9-updateOneUser Method.

[29]

```
/**
 * This method is used to delete a user.
 * @param userId the id of the user to be deleted
 * @return "Success!" if the user exists and thus deleted from the userRepository, "Failed: User with the given id
 * does not exist!" if such user does not exist, "Failed: User does exist, but for some reason it can't be deleted!"
 * if the user exists but can't be deleted.
 */
@DeleteMapping("/{userId}")
public String deleteOneUser(@PathVariable Long userId) {
    return userService.deleteById(userId);
}
```

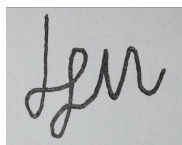
Figure 10-deleteOneUser Method.

Self-Checklist for Your Report

Please check the items here before submitting your report. This signed checklist should be the final page of your report.

- ☒ Did you provide detailed information about the work you did?
- ☒ Is supervisor information included?
- ☒ Did you use the Report Template to prepare your report, so that it has a cover page, has all sections and subsections specified in the Table of Contents, and uses the required section names?
- ☒ Did you follow the style guidelines?
- ☒ Does your report look professionally written?
- ☒ Does your report include all necessary References, and proper citations to them in the body?
- ☒ Did you remove all explanations from the Report Template, which are marked with yellow color? Did you modify all text marked with green according to your case?

Signature:

A square box containing a handwritten signature in black ink. The signature appears to be 'Jen' or similar, written in a cursive style.