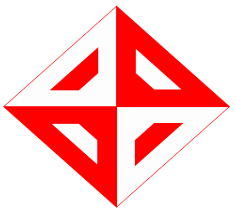
MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING



SUMMER PRACTICE REPORT

CENG 400

STUDENT NAME : Onur ŞİMŞEK

ORGANIZATION NAME : Spark Kalibrasyon

ADDRESS : Üniversiteler, 06800 Çankaya/Ankara

START DATE : 6 July 2020

END DATE : 7 August 2020

TOTAL WORKING DAYS : 20

STUDENT'S SIGNATURE **ORGANIZATION APPROVAL**

TABLE OF CONTENTS

Başlıklar (Biçim > Paragraf stilleri) eklediğinizde, içindekiler tablonuzda görünürler.

TABLE OF FIGURES

1. INTRODUCTION

This report document includes what I have done during my internship at Tübitak Bilgem YTE including the difficulties that I faced with and how I solved them.During my internship, software engineer Ahmed Enis ERKAYA was my mentor.In this report, you can find what my project is about and the technical details of my project.

My internship was in between 30.07.2019 and 13.09.2019.There was a holiday in between these days, and after the holiday we continued to our project .The aim of my internship is to understand what software engineers do during one day , observe the working environment and try to solve the behaviours of the engineers while they are working.

In Tübitak Bilgem YTE, all the interns had the same project about web technologies and there was no team work. In other words, during my internship I didn't have a friend to work with. However, to make our working environment more social and interactive,I and my friends talked about the problems that we have and how we overcome them. After finishing the project that was assigned to me , I made some improvements to create a better and more useful program.

2. INFORMATION ABOUT PROJECT

My project is called the Alerting System Application . It is project where we have a web page including a form. By using this form, we have some information such as name ,url , http method and control period. The main usage of this project is to send request to the websites by using this url field and see the result of this request by the help of a chart. In addition to this, by using control period field , we control the response of the website in every time interval that is given as control period .The field http method is to determine the http request type of the request.

There is also a page which is called user login. The user can directly go to the form or he/she can first go to the login page and then go to the form page.

2.1 ANALYSIS PHASE

While I was creating the Alerting System Application, I have used many tools that are Intellij IDEA,Visual Studio,Postman,Java Spring Boot,PostgreSQL,React JS, start.spring.io, Node.js and Git Bash. Here, you can find brief information about these tools and how I have used them.

2.1.1 Intellij IDEA

Intellij IDEA is a very practical and useful tool for especially Java developers.It helps you understand your mistakes easily before you compile your codes.Also, I used it in a Spring Project where I don't know other ways to run if I don't have Intellij IDEA. By using this tool you can also add plugins that are used in Spring Project such as Lombok.

2.1.2 Visual Studio

I used Visual Studio for the front end side of my project. I used the terminal of Visual Studio to start or run my project.

2.1.3 Postman

Postman is a tool to test your backend side of project without using a frontend side. By using Postman, without having a frontend side, you can send request to the website and understand whether your backend part is working properly or not .

2.1.4 Spring Boot

Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can just run. I used the spring boot to have a connection between my database and my program.Spring boot enables you to connect the database directly without writing any sql commands.

2.1.5 PostgreSQL

PostgreSQL, also known as Postgres, is a free and open-source relational database management system emphasizing extensibility and technical standards compliance. I used the PostgreSQL to understand whether the changes that I made is visible. That is ,if the post operation in my form is successful , I should be able to see the changes inside the related table .

2.1.6 React JS

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex user interfaces from small and isolated pieces of code called components.I used React JS for frontend side of my project.I have created many components for each specified task and call them inside App.js

2.1.7 Start.spring.io

Start.spring.io is a webpage that help us to create Spring projects. In this website, you can specify the version of Spring that you want to use and you can dependencies to your project such as Spring Web Starter,Lombok, Spring Data JPA and PostgreSQL Driver.

2.1.8 Node.js

Node.js is an open-source, cross-platform, JavaScript run-time environment that executes JavaScript code outside of a browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser.

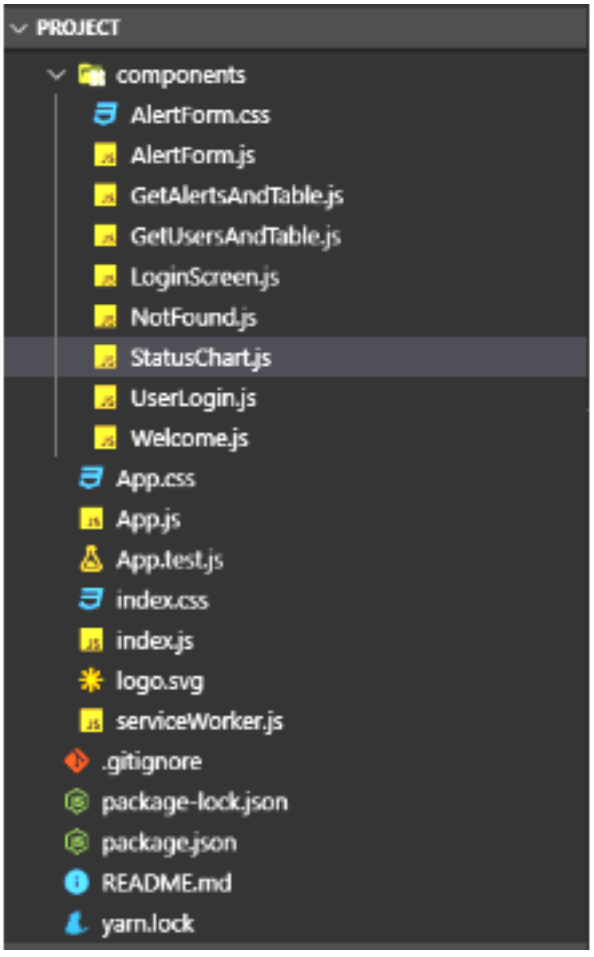
2.1.9 Git Bash

I used Git Bash as a linux terminal on Windows operating system. React applications needs to be started by using the command “create-react-app”.This operation takes some time. In these situations, I use Git Bash as a helper tool and use the Visual Studio’s terminal for other purposes.

2.2 DESIGN PHASE

The implementation and the design phase of my project consist of two parts. One is the frontend part and the other is backend part. Since I use different technologies and tools for these two parts, I will observe the design and implementation part separately.

2.2.1 Fronted Side



As you can see from the image on the left side, I have a folder called components and I have App.js.In addition to these, I also have some .css file to change the appearance of the web site. However, in many cases I tried to give these css format not as a className but as style inside .js files.By observing the requirements, we need to have a form page and a login page to direct the user to the form page. After this we should also have the tables for alerts and users.

Figure 1 : The design of the frontend part

If the users enter a different url, we need to have a “Not Found” component so that we can direct the user to this not found page. Finally, to show the response result of the website, we should have a component for chart. This component will just

create the chart according to the response from websites. To have a better idea, you can observe the figures below where I indicate some images from my project .

Figure 2 : Form page of alerting system

In figure 2 , you can see the place where we can take url and the request type by the help of user interface.Later on, we will use these two as well as the control period in the backend side of our project to send request with the given request type and control period .After adding the alert , since the user might want to see all of the alerts, I also put a button called *ALERT LIST .*

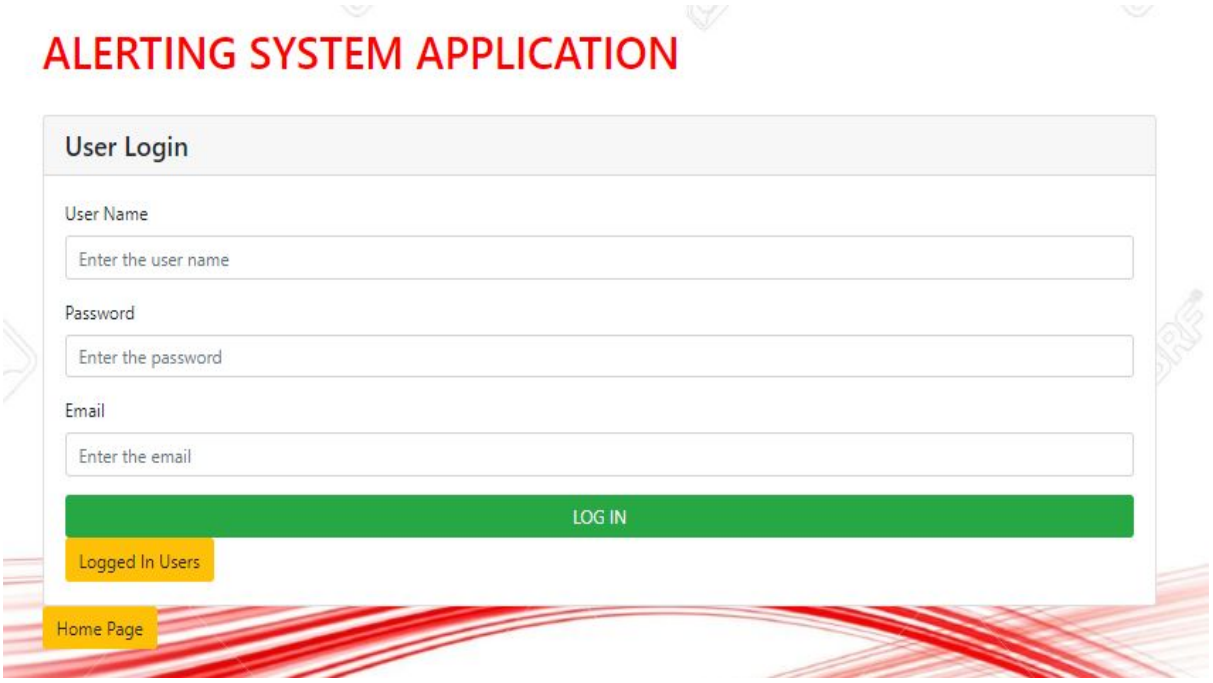


Figure 3 : User login form screen

Figure 4 : The first page that we see when we run project

In figure 3 and figure 4 ,you can see the page from where we can go to form page after log in operation and the first page .

Figure 5 : The table that we show the alert list

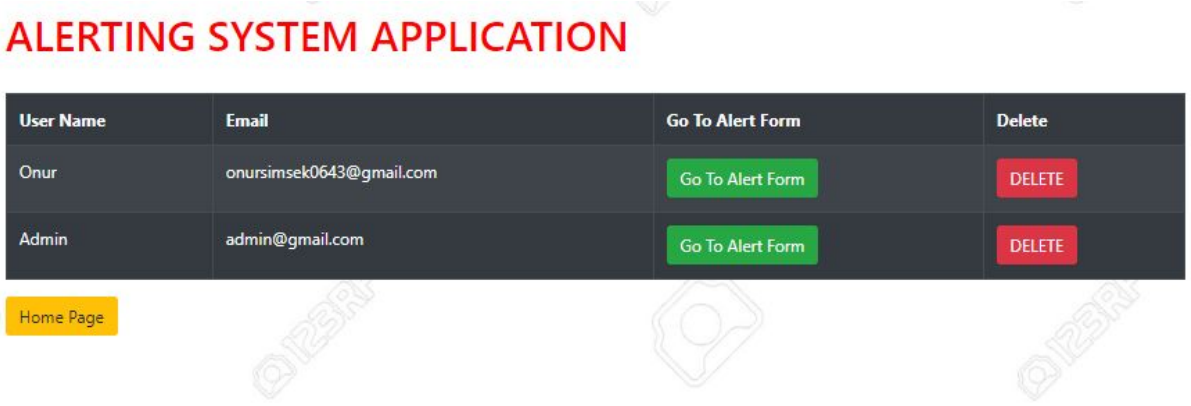


Figure 6 : The table that shows the user list

As you can see from Figure 5 and Figure 6 , there is a button called “GET CHART” and “Go To Alert Form” . That means , after pressing the submit button in the form, you can go to the related chart by using the column in alert table.Similarly, after log in operation , you can go to the alert form as a logged in user .There is also one more column in tables just to delete the alert record or delete the user .In addition, since the user may want to go to the home page from these pages, I put one more button just to directly go to the home page.

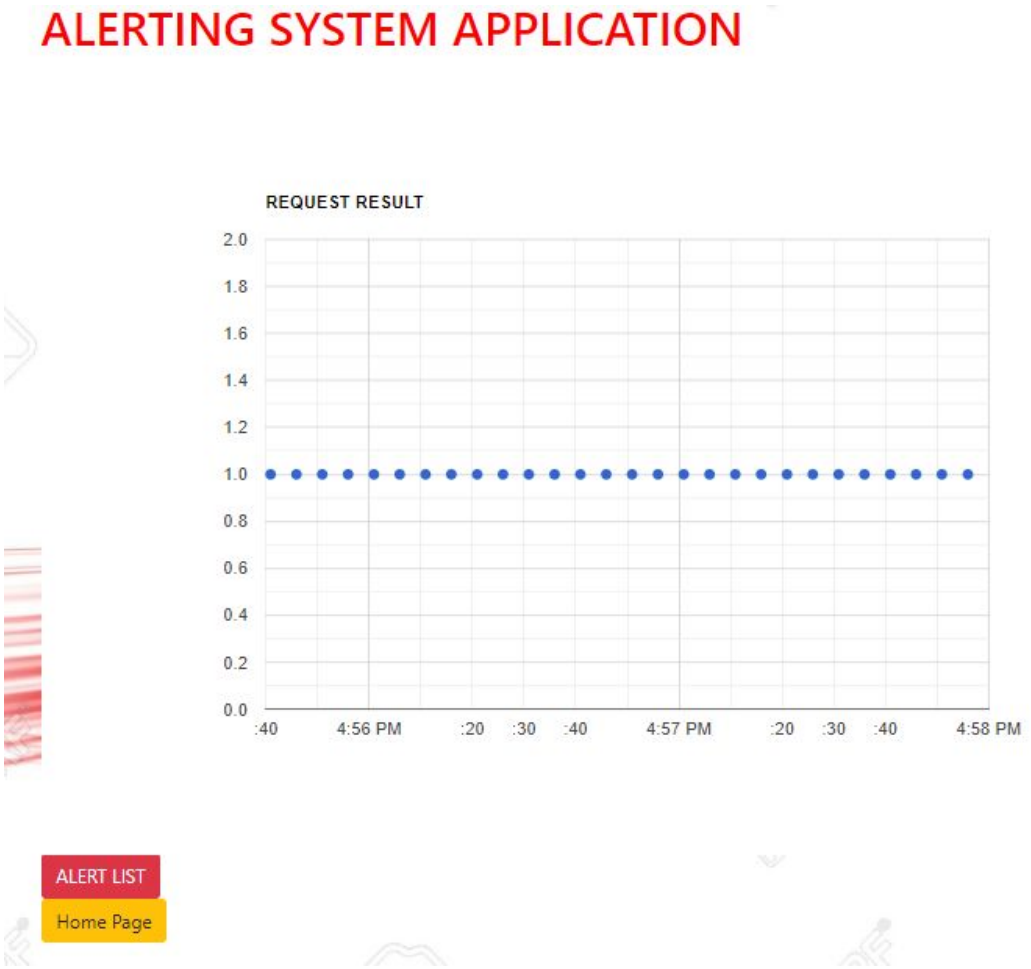


Figure 7 : The chart where we can see the request result

2.2.2 Backend Side

In the backend side , I'm creating java Spring Boot application.There should be four main packages to group the work that you I done. These are :

- Controller
- Model
- Service
- Repository

To have a "Alert" and "User" table in my database ,I should create them as a model in my model package. Inside the repository package, I have an interface that I didn't implement anything. Service package is where I implement my methods by the help of Jpa Repository. Finally, controller package is to specify the mapping address of the request so that each function can have a mapping address and we can check

whether they are working or not. To make it more clear and understand the design of my backend side, you can check the screenshot below .

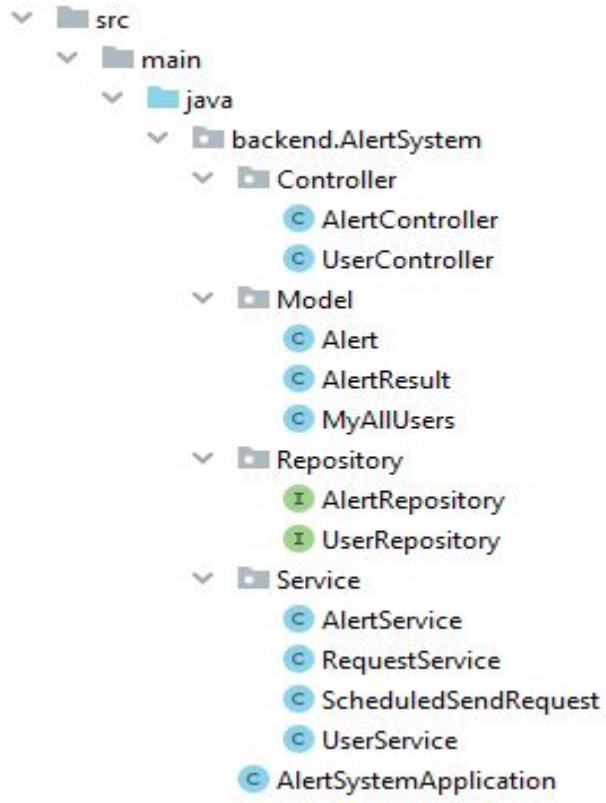


Figure 8 : The design of backend side of the project

2.3 IMPLEMENTATION PHASE

Like in the design part, I will observe the frontend and backend side in different parts.

2.3.1 Frontend Side

In the implementation of the frontend side, I have *App.js* and a package called components. In *App.js*, the components are called and I add *"route"* ,which is defined in *"react-router-dom"* library, to open a new page when we click the components. Also, by using the *"<button>"*, I add a button called *Home Page* to access the home page from all pages of the application.

In the components package, I have six main JavaScript file. To have a useful and practical home page, I have created the *LoginScreen.js* file. In this file, I used *<ul>* and *<li>* to make a list format and *<Link to = "/">* to make them open in another page. Similar to *route*,*link* is also defined in *"react-router-dom"* library. Apart from

this, I have also *AlertForm.js* and *UserLogin.js* where I use the form structure that is defined in *"react-bootstrap/Form"*.Now, since we have a form format in these two files, we need to implement two functions to handle the change on inputs and submit inputs when the user press submit button .These two functions are *submitHandler* and *changeHandler*. In *submitHandler*, I use axios which I import from axios node module. I want to give you a brief information about why I use axios and how I use it.

What is axios and why I use it ?

Axios is a Javascript library used to make HTTP requests from node.js. I have used axios to connect the frontend side and backend side of my project. In other words, there are some methods in the backend side of my project for special tasks. Axios takes the url as a parameter to this method and when we use axios with the given url ,we make request to server.

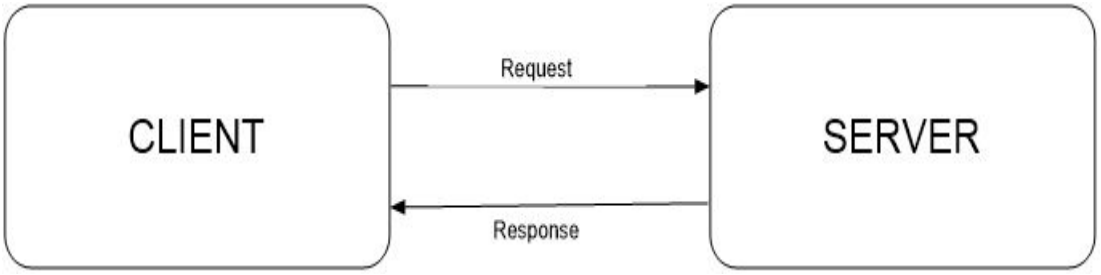


Figure 9 : The relationship between client and server

As you can see from Figure 9 , we make a request to a server from the client , and server gives us a response that shows whether the request that we try is successful or not .There are many request types that you can use depending on what you want to do . I have used *axios.post* , *axios.get* and *axios.delete* to make post request , to get some information from a specific url and to delete the object by giving it's id.

Since we know what is axios, now you can check how I use axios during the implementation of *AlertForm.js* by using the below image .

```
18      changeHandler = (e) => {
19          console.log(e)
20          this.setState({[e.target.name] : e.target.value}) ;
21      }
22
23      submitHandler = (e) => {
24          e.preventDefault() ;
25          console.log("submit handlera girdi")
26          axios.post('http://localhost:8080/addAlert',this.state)
27          .then((response) => {
28
29              console.log(response.data) ;
30              this.setState({
31                  name: '',
32                  url: 'https://',
33                  httpMethod: 'GET',
34                  controlPeriod: ''
35          })
```

Figure 10 : ChangeHandler method and the usage of axios.post

The remaining three main components are the place where I use *axios.get* and *axios.delete* and make tables to list all the users and all the alerts with the all given columns. You can check the *Design Phase* section to see how they look like.In *GetAlertsAndTable.js* , I used *axios.get* to get the information that I post by using *axios.post* .You can check the below code to see how I use *axios.get.*

```
8       state = {
9           alerts : []
10      } ;
11      componentDidMount(){
12          console.log("Get Alert component mounted") ;
13          this.retrieveData() ;
14      }
15      retrieveData(){
16          axios.get('http://localhost:8080/getAlerts')
17          .then((response => {
18              console.log(response.data) ;
19              this.setState({alerts : response.data});
20
21          })) ;
```

Figure 11 : The usage of axios.get

To delete an alert when we press the delete button, we should use *axios.delete.* In the below code, you can see how I use *axios.delete*

```
23      deleteAlert(id){
24          console.log("It entered the delete method."+ id) ;
25          swal ({
26              title : "Are you sure ?",
27              text : "If you delete, you can't reach this alert again!",
28              icon :"warning" ,
29              buttons : true ,
30              dangerMode : true
31          })
32          .then((willDelete)=>{
33              if(willDelete){
34                  axios.delete('http://localhost:8080/deleteAlert/'+id)
35                  .then((response)=>{
36                      this.retrieveData();
37                      swal({
38                          title : "Good Job!",
39                          text : "Alert is deleted!",
40                          icon :"success" ,
41                      })
```

Figure 12 : The usage of axios.delete together with swal

In Figure 12, I used swal just to ask the user whether he/she wants to delete the alert or not . Since the user might delete the alert accidentally, I find this solution to guarantee the deletion operation.

```
let myTableData = this.state.alerts.map((element,elIndex) =>{
    return <tr key = {elIndex}>
        <td>{element.name}</td>
        <td>{element.url}</td>
        <td>{element.httpMethod}</td>
        <td>{element.controlPeriod}</td>
        <td>
            <Link to={"/getAlerts/" + element.id}>
                <button className = "btn btn-success">
                    GET CHART
                </button>
            </Link>
        </td>
        <td>
            <button className = "btn btn-danger"
            onClick = {() => {this.deleteAlert(element.id)}} >
                X
            </button>
        </td>
    </tr>
}) ;
```

Figure 13 : Mapping operation on an array to create the table data

The logic behind *GetUsersAndTable.js* is exactly the same as *GetAlertsAndTable.js* .I will not mention it in here in order not to confuse you.

The last part of the implementation of the frontend side is nothing but *StatusChart.js*

```
18    componentDidMount() {
19        setInterval(() =>{
20            console.log("Result component mounted...") ;
21            this.getChartData();
22        },1000);
23
24    }
25    getChartData(){
26        if(this.props.match.params.selectionId !== null){
27            axios.get('http://localhost:8080/getAlerts/' + this.props.match.params.selectionId)
28            .then(response =>{
29                this.setState({alert : response.data });
30            })
31        }
32    }
```

Figure 14 : Using axios.get to obtain the result of the related alert

Figure 14 shows how the related alert information can be obtained by using axios.get.

```
34        if (this.state.alert !== null){
35            return(
36                <div>
37                    <Chart
38                        width={1000}
39                        height={650}
40                        chartType="ScatterChart"
41                        loader={<div>Loading Chart</div>}
42                        data={[[
43                            {type: 'datetime', label: 'x'},
44                            {type: 'number', label: 'values'},
45                        ]].concat(this.state.alert.alertResult.map((element, index) => {
46                            return [new Date(element.requestDate), element.requestResult]
47                        }))}
48                        options={{
49                            backgroundColor : "white",
50                            title : "REQUEST RESULT" ,
51                            intervals: {style: 'sticks'},
52                            legend: 'none',
53                        }}
54                    />
```

Figure 15 : The implementation of chart that shows the request result

Please keep in mind that we should have *requestDate* and *requestResult* in backend side so that we can save the request date and result of the request .In addition, the map operation is on *AlertResult* which I have described as a set. You can see this when you observe the below section.

2.3.2 Backend Side

I want to start the implementation of the backend side with *Model* package.In *Model* package , I have three java file :

- Alert.java
- AlertResult.java
- MyAllUsers.java

First, I want to explain you the annotations that I used and why I used them .

```
12  @Entity
13  @Getter
14  @Setter
15  @AllArgsConstructor
16  @NoArgsConstructor
17
```

Figure 16 : The annotations used in these three java files

In all these three java files in the *Model* package, I used these annotations. *"@Entity"* annotation specifies that *Alert* will be an entity ,which means if everything goes right in database connection, we should see a table called *"Alert"* . *"@Getter"*, *"@Setter"*, *"@AllArgsConstructor"* and *"@NoArgsConstructor"* are defined annotations in *Lombok* .In order to use the annotations , you should import them by using *Lombok* .As the name implies, *"@Getter"* and *"@Setter"* creates the getter and setter functions in *Java* in order to use them in other files .*"@AllArgsConstructor"* creates an all-args constructor that needs to be filled by every fields of the class.*"@NoArgsConstructor"* generate an error message if such a constructor cannot be written due to the existence of final fields.

```java
public class Alert {
    @Id
    @GeneratedValue
    private Long id;
    private String name;
    private String url;
    private String httpMethod;
    private Long controlPeriod;
    private Long remainingTime = 0L;
    @OneToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinColumn(name = "result_id")
    private Set<AlertResult> AlertResult;
}
```

Figure 17 : Alert model and it's fields

As you can see from Figure 17, there is a "one to many" relation between *Alert* and *AlertResult* , since one alert can have many alert results. Also, I have a set called *AlertResult* that will store the response and date of request. As you might recognise,I used these while I was creating the chart.

I also needed to keep track of remaining time since I should send request when the remaining time is 0 , or in other words when the control period finishes .For example, when the control period is 5, it will control the request result in every 5 seconds. To see this ,it will be enough for you to click on each request and look the date of each . (See Figure 7)

The implementation of AlertResult.java is very similar.

```java
19  public class AlertResult {
20      @Id
21      @GeneratedValue
22      private Long id ;
23      private LocalDateTime requestDate ;
24      private Integer requestResult ;
25
26  }
```

Figure 18 : The implementation of *AlertResult.java*

Alert and *AlertResult* entities were enough to make our job.However, I also wanted to add a new User.java. This can be regarded as a very useful approach since many websites use this login authentication system. What I intended to do was to make my program accessible by only a few people. Other people who aren't registered would be unable to access my website .However , this approach requires a higher level understanding of Spring and Spring Security Boot which I cannot handle by using the Internet. In addition, it also requires a lot of times since these technologies were new to me. Therefore, I decided to create a Java class to handle :

- The login form (See Figure 3)
- The logged in users list (See Figure 6)

In logged in users list ,there is a column to go to the alert form as a logged in user. Because of these reasons, I create an additional *MyAllUsers.java.* I have four fields inside it, namely id ,username,password and email. Since I have my models right now, I want to show the database combined in one figure.

Data Output | Explain | Messages | Notifications

	id [PK] bigint	control_period bigint	http_method character varying (255)	name character varying (255)	remaining_time bigint	url character varying (255)
1	11563	5	POST	googlepost	3	https://www.google.com

Data Output | Explain | Messages | Notifications

	id [PK] bigint	request_date timestamp without time zone	request_result integer	result_id bigint
1	11590	2019-09-05 23:24:29.385161	0	11563
2	11592	2019-09-05 23:24:39.384391	0	11563
3	11593	2019-09-05 23:24:44.390969	0	11563
4	11594	2019-09-05 23:24:49.391636	0	11563
5	11595	2019-09-05 23:24:54.385042	0	11563

Data Output | Explain | Messages | Notifications

	id [PK] bigint	password character varying (255)	username character varying (255)	email character varying (255)
1	11629	deneme	onur	onursimsek0643@gmail.com

Figure 19 : The tables and their rows in database

Until now , I only mentioned about the *Model* package. If you go to Figure 8 , you will see that there is also *Repository* package.Inside it, I have *AlertRepository.java* and *UserRepository.java.* These are the repository files that we don't implement anything , so they should be an interface. Inside these files, we use JpaRepository.

```
7   @Repository
8   public interface AlertRepository extends JpaRepository<Alert,Long> {
9
10
11  }
```

Figure 20 : The implementation of *AlertRepository.java*

Now ,this brings me to the *Service* package which includes the definition of the methods. I will first observe the annotations then the java files that I use .

```
12  @Service
13  @RequiredArgsConstructor
14  @Transactional
```

Figure 21 : The annotations used in *AlertService.java*

"@Service" is to specify explicitly that this file will be a service file .Therefore , you should use this annotation at the top of the class name if you used it inside the *service* package. *"@RequiredArgsConstructor"* generates a constructor with required arguments. Required arguments are final fields. Therefore,when you use this annotation , your fields should be defined with final keyword . I was getting an error related to this, since I didn't use the keyword final. *"@Transactional"* indicates whether a bean method is to be executed within a transaction context.

In *AlertService.js* file one important thing is that there are methods that need to be created. We can achieve this by taking an instance to *AlertRepository.java* and use some methods that are already defined in *JpaRepository.* In this project, we need to add an alert , get all the alerts ,delete an alert and get the specific alert to draw it's chart.Therefore ,for every specific task right here, I have described methods to do these tasks.

```java
public class AlertService {
    private final AlertRepository alertRepository;
    public Alert addAlert(final Alert alert) {
        return alertRepository.save(alert);
    }
    public List<Alert> getAlerts() {
        return alertRepository.findAll();
    }
    public void deleteAlert(Long id) {
        alertRepository.deleteById(id);
    }
    public Optional<Alert> getAlertById(Long id) {
        return alertRepository.findById(id);
    }
}
```

Figure 22 : The implementation of *AlertService.java*

Since I have an instance of *alertRepository* ,I can now use the methods that is defined in *JpaRepository.* These methods are :

- *save()*
- *findAll()*
- *deleteById()*
- *findById()*

This concludes our discussion related to adding ,deleting and getting the alerts. Now,we need to send the request to given url with the given request type and do this task at each scheduled time . Therefore, we need to implement a method to send request and a class to send request at scheduled time interval.This was the most challenging part for me ,since there are many ways of doing this task and the number of sources related to this is very limited.

To send a request , and save the request result to the database we need to check the response code of the connection. A response is called a successful if the response code is in between 200 and 300.Therefore , while doing this task , I take the response code and check whether it is equal to 200 or not . If it is not equal to 200 ,it is not considered as a successful connection.In the case of having the successful connection *requestResult* should be 1.Otherwise ,it should be 0 indicating that this connection is not successful .

RequestResult should be 0 in two cases :

- If you send a request to a bad url
- If the website url is ok , but your request type is wrong .For example , you are trying to post something to Google

```
23 @    public void sendRequest(Alert alert){
24          LocalDateTime localDateTime = LocalDateTime.now();
25          alert.setRemainingTime(alert.getControlPeriod());
26          try{
27              URL url = new URL(alert.getUrl()) ;
28              HttpURLConnection con = (HttpURLConnection) url.openConnection() ;
29              con.setRequestMethod(alert.getHttpMethod());
30              con.connect();
31              int status = con.getResponseCode() ;
32              if (status == 200){
33                  AlertResult alertResults = new AlertResult( id: null,localDateTime, requestResult: 1);
34                  alert.getAlertResult().add(alertResults);
35                  System.out.println(alert.getName() + " Connected");
36              }
37              else {
38                  AlertResult alertResults = new AlertResult( id: null,localDateTime, requestResult: 0) ;
39                  alert.getAlertResult().add(alertResults) ;
40                  System.out.println(alert.getName() + " Not connected");
41              }
42          }catch(Exception e){
43              AlertResult alertResults = new AlertResult( id: null,localDateTime, requestResult: 0) ;
44              alert.getAlertResult().add(alertResults) ;
45          }
```

Figure 23 : The function that sends request to given url

As you can see from Figure 23, we first create a *url* object then open a connection called *con.* After this, to make a request we also need to have request type and have a connection. *getResponseCode()* is a very helpful function that returns us an integer indicating the HTTP status code. This function sends request , create an object by using constructor and save this results by adding these to the *AlertResult* set .

This function does all the task that we need .The only remaining thing is doing this task at each scheduled time interval and doing this not for only one alert but for all the alerts. Therefore, inside another class, we need to :

- Get all the alerts with the help of findAll() inside a list
- Have a for loop to make this job for all the alerts
- “@EnableScheduling” and “@Schehuled” annotations
- Send the request if the control period finishes

```
21      @Scheduled(fixedRate = 1000)
22      public void scheduledRequest(){
23          List<Alert> alertlist = alertRepository.findAll() ;
24          for(Alert alert : alertlist){
25              if (alert.getRemainingTime() == 0L){
26                  requestService.sendRequest(alert);
27              }
28              else {
29                  alert.setRemainingTime(alert.getRemainingTime()-1L);
30                  alertRepository.save(alert) ;
31
32                  if (alert.getRemainingTime() == 0L){
33                      requestService.sendRequest(alert);
34                  }
35              }
36
37      }
```

Figure 24 : Sending request at each fixed rate

Now our work becomes more clear. We have url that we take by the help of *AlertForm.js* and we are sending a request to this url at each *fixedRate* (1 second in my case) and control the response of the server by using the control period .

In figure 24, we are checking whether the control period finishes which means is it a right time for us to send request or should we wait it ?

I have mentioned about *Service* package. The only remaining package is *Controller.* In the controller package we have *AlertController.java* .This file will be very similar to *AlertService.java*. In there, we have taken an instance of *alertrepository* ,but in here we should take an instance of *alertservice* since we are using the methods of *alertservice*. The annotations that I use in *AlertController.java* is different from other annotations that I have mentioned.

```
11  @RestController
12  @RequiredArgsConstructor
13  @CrossOrigin("*")
14
```

Figure 25: The annotations defined at the top of the class name

"@RestController" is a specified version of *"@Controller".* It includes *"@Controller"* and *"@ResponseBody"* annotations.Therefore, it simplifies the controller implementation. *"@CrossOrigin('*')"* is to allow request from anywhere .If you don't use this annotation ,you will get a CORS policy error.

```
17      private final AlertService alertService;
18      @PostMapping("/addAlert")
19      public Alert addAlert(@RequestBody final Alert alert) {
20          return alertService.addAlert(alert);
21      }
22      @GetMapping("/getAlerts")
23      public List<Alert> getAlerts() {
24          return alertService.getAlerts();
25      }
26      @DeleteMapping("/deleteAlert/{id}")
27      public void deleteAlert(@PathVariable Long id) {
28          alertService.deleteAlert(id);
29      }
30      @GetMapping("/getAlerts/{id}")
31      public Optional<Alert> getAlertById(@PathVariable Long id) {
32          return alertService.getAlertById(id);
33      }
34
```

Figure 26 : The methods and annotations used in *AlertController.java*

It can be easily seen from Figure 26 that the method names are the same .We are using the methods of service . On the other hand, we have some annotations to talk about :

- *@PostMapping* : Handles the HTTP POST requests matched with given url. When we say *axios.post* in the backend side, we post the alert by the help of this annotation and it's method.
- *@GetMapping* : Handles the HTTP GET requests matched with given url.
- *@DeleteMapping* : Maps HTTP DELETE requests onto specific handler methods.
- *@RequestBody* : I used this annotation while I was adding alert .In other words, whenever you are using HTTP POST method, you are giving it by using it's body, so you should use it in the parameter of related method .
- *@PathVariable* : I used this annotation when I wanted to delete a specific alert or get the fields of specific alert. In such cases, I have taken the id of the alert as a path and have used this annotation as a parameter to the related method.

2.4 TESTING PHASE

We were not responsible for creating test in this project .It was optional to us. I didn't write any code for testing, but I tried to make some improvements on my project .However, when we think of the job of the program, it is sending a request and getting a response . Therefore , if we go to the request response chart shown in Figure 7, there will be some responses each of which shows the response which is either 1 or 0. As long as we do not stop execution of the backend side of the project ,we should get the response value as 1 or 0 .This fact was what I checked to understand whether the project is working or not .

3. ORGANIZATION

3.1 ORGANIZATION AND STRUCTURE

In 2012, the company has been transformed into an institute under the name of Software Technologies Research Institute (YTE) affiliated with Informatics and Information Security Research Center (TÜBİTAK-BİLGEM).

TÜBİTAK-BİLGEM-YTE develops R&D-focused software solutions that take into account new technologies and innovative approaches to meet the digital transformation needs that provide digital policies of public institutions and organizations.

There are many properties that only YTE has :

- YTE is the one and only institution owning CMMI Level 5 accreditation located in Turkey.
- It is also a member of The Open Group consortium.

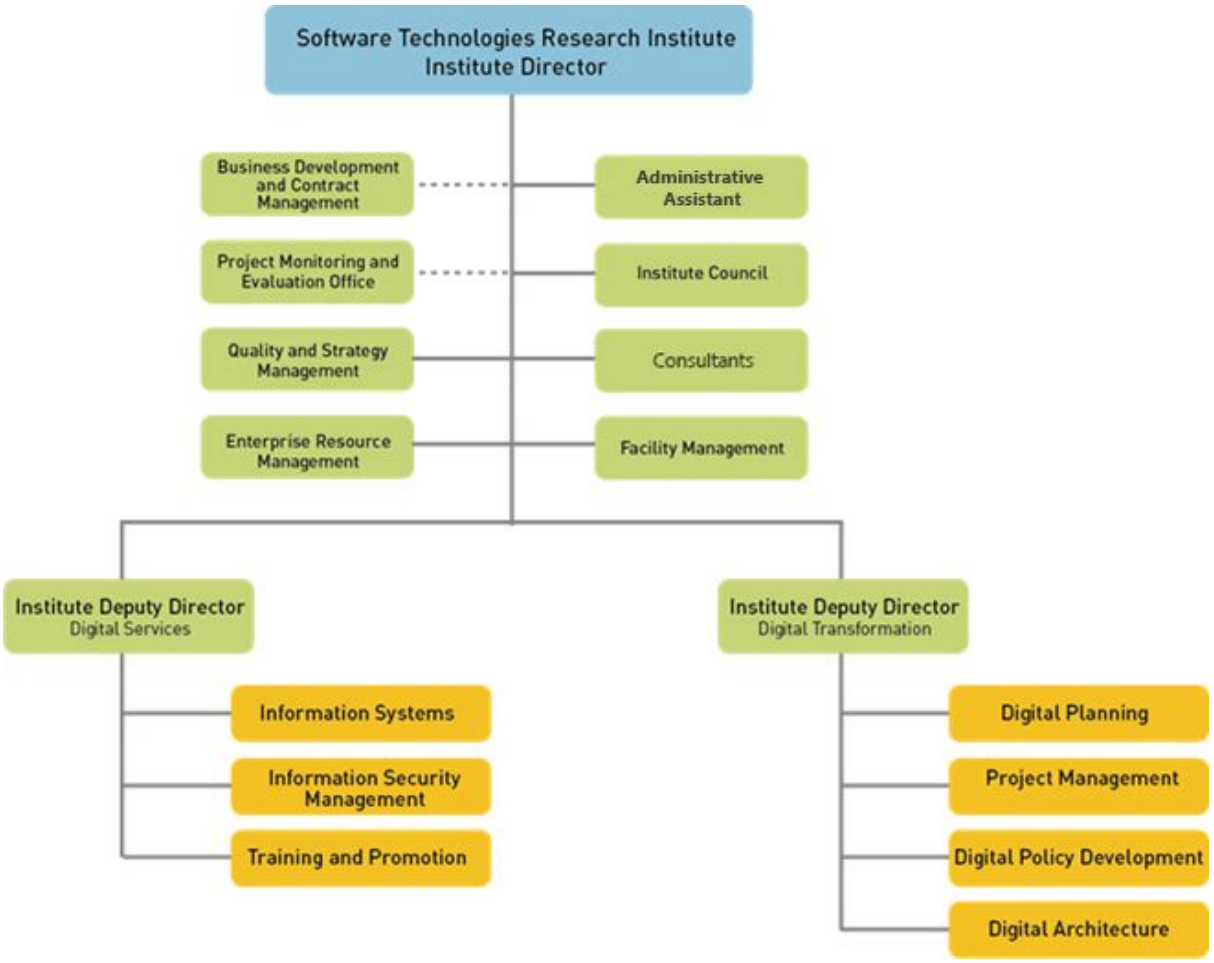


Figure 27 : Organization Structure

3.2 VISION AND GOALS

To become the pioneer research institute that makes Turkey as a reference point by contributing to the development of informatics ecosystem in order to implement effective digital transformation policies.

The company is aimed :

- Conduct strategic, critical and R&D software development projects in digital government policies,
- Develop guideline and reference models with the purpose of building capacities required in informatics ecosystem and to provide guidance,
- Build capacity in new technology areas by following international digital trends and developments,

- Conduct policy researchers to improve the performance and quality of informatics ecosystem.

4. CONCLUSION

In conclusion, after having an internship at Tübitak Bilgem YTE and creating the Alerting System Application, I have learned the technologies that I haven't known before. In addition to technologies, I had a chance to see the working life and meet a lot friends and computer engineers.

During my internship , the times that I have difficulties were :

- The time when I try to send request
- The time when I try to use Spring Security Boot
- The time when I try to create a test

However , I added many features to my project as a personal choice :

- Deleting an alert
- Sending delete and post request as well as get
- User message to make my program useable
- Adding many route to make the connection open in other page
- Adding an user and going to alert form page as logged in user

It was the first time for me to have adventures with React ,Spring Boot,PostgreSQL and Postman.

5. REFERENCES

- Organization Structure.(n.d.). Retrieved from : https://yte.bilgem.tubitak.gov.tr/tr/kurumsal/icerik-organizasyon-yapisi
- Vision and Goals.(n.d.). Retrieved from : https://yte.bilgem.tubitak.gov.tr/en/kurumsal/icerik-vision-and-goals