College of Engineering

COMP 491 – Computer Engineering Design Project

Final Report

Ülgen: Residence Activity Monitor and OR Solutions to Disasters

Participant information:

Kaan Türkmen, Can Usluel, Halil Doruk Yıldırım, Bumin Aybars İnci

Project Advisor
Didem Unat

Spring 2023

Table of Contents

1. Abstract

The earthquakes that struck on February 6th, centering Kahramanmaraş and impacting multiple provinces, have left all of us in a state of devastation [1]. The aftermath of these earthquakes clearly showed a lack of preparedness, communication, and handling of necessary regulations, which in turn cost in lives of thousands and dislocation of millions. This tragic incident was the primary driving force and motivation behind our project, Ülgen. In our project, we focused on a particular issue which is often disregarded in current disaster related applications; the fact that access to the internet, messaging systems, or mobile devices are hindered during times of emergency and crisis. As a result of our efforts during our semester, we have created a functional and applicable solution which leverages local network connectivity data of users to estimate the presence of individuals within an area during the time of disaster, eliminating any need for user input or interaction. This data is then used within our clustering algorithm to create optimised routing and resource distribution solutions, enhancing the efficiency of rescue and resource operations. We implemented a robust architecture to ensure system resilience, security and scalability as we are aware these matter greatly in applications where such sensitive data is processed. Testing showcased impressive response times, as we discuss further in our report. Ülgen has been implemented as an Android application, due to its overwhelming popularity in Turkey [2], with user authentication, local network monitoring, heat mapping and optimal routing solutions, and a table displaying recent earthquakes. All our functionalities were tested thoroughly using server-side microservices. Overall, our efforts have proven successful in meeting the desired objectives, resulting in a viable solution to aid in disaster management scenarios regarding communication, information, and coordination.

2. Introduction

Ülgen is the supreme sky god of Turkic mythology. He is revered as a wise and benevolent god who protects humanity and ensures the well-being of communities. We named our project Ülgen because we wanted to create a lifesaving and impactful application which aims to protect those who need it during times of disaster. When we explored the existing projects and applications which aim to aid those who need it during times of disaster, we noticed a commonality between all of them; they all expect the user to have access to the internet, messaging systems, or at the very least their mobile device [3,4]. This expectation is highly optimistic, as when disaster strikes, people are usually caught off guard and may not reach their devices. Also, as we witnessed in the devastating earthquakes which occurred on the 6th of February centred in Kahramanmaraş, communicative infrastructures may get

damaged and as a result, access to messaging systems or the internet may be hindered which can also be seen in Figure 1.

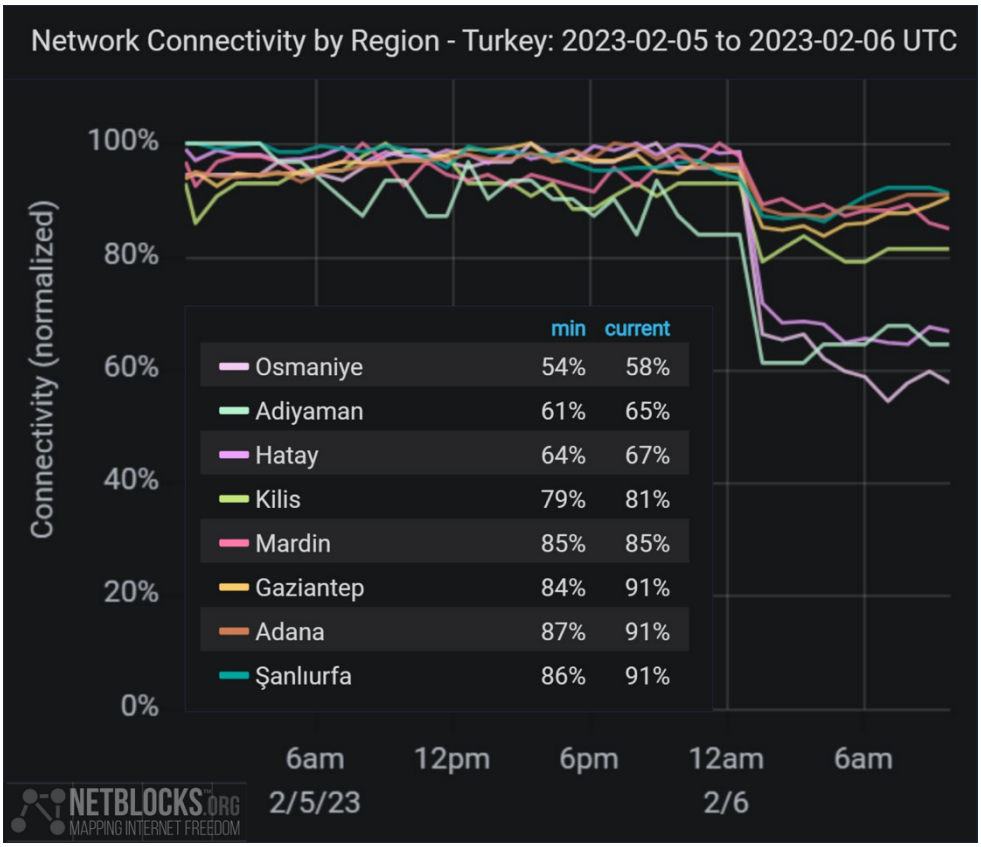

Figure 1. Network connectivity during the time of disaster. [5]

The vulnerability Turkey has towards earthquakes both geographically and infrastructurally is evident from both the Turkey Earthquake Hazard Map published by AFAD, in Figure 2, and from the disastrous conclusions of the aforementioned earthquakes, resulting in the loss of over 45,000 lives and the displacement of millions. It is apparent that an effective solution, which improves communication and coordination, is necessary for disaster management scenarios.

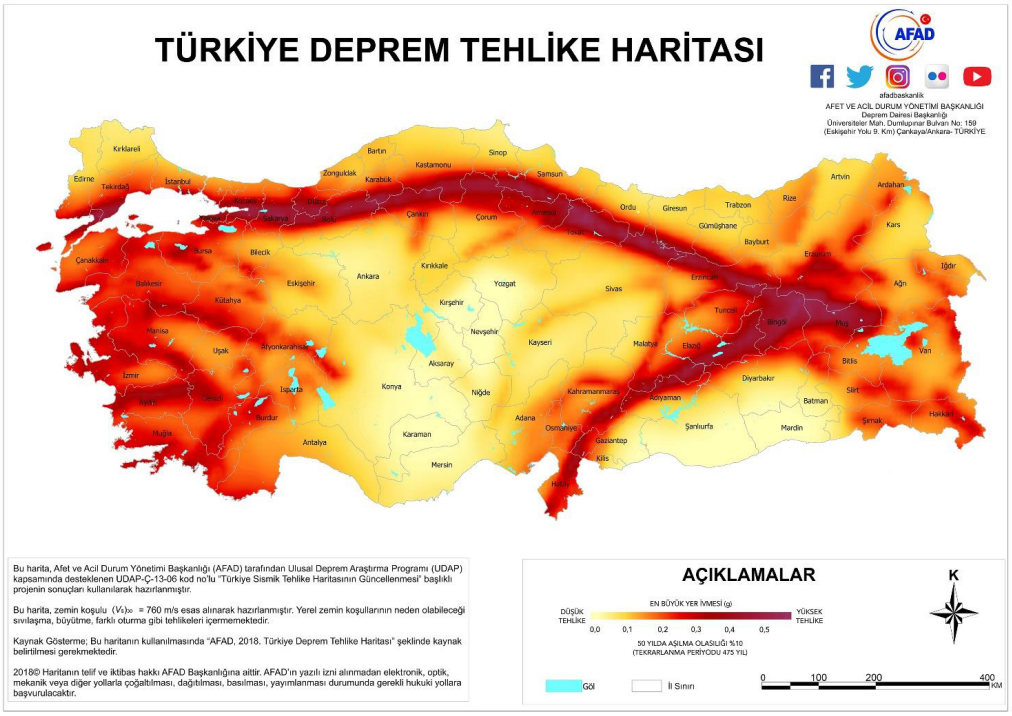

Figure 2. Turkey Earthquake Hazard Map. [6]

Considering all of this, we wanted to create an application that will aid in locating potential earthquake and disaster victims and which ensures the rescue operations are carried out as optimally as possible, with the specialty of not needing any input from users. Thus, even if infrastructures are down, or if the device of the victim is unreachable or damaged; relevant data regarding the whereabouts of potential victims can be obtained and used for rescue efforts. We achieve this by implementing a robust application which periodically gathers the data of users regarding their local network connectivity, to then estimate the amount of people within the household during the time of the disaster. This data is then used within our clustering algorithm which enables us to create routing and optimization solutions aimed to assist rescue and resource distribution operations.

3. System Design

Ülgen had few limitations in the design stage, three of which were deployment costs, 10-week time limit, and testing constraint. As all engineering projects, we have estimated costs, and reported this process in each progress meeting to TAs. We have determined we will spend approximately 105 US Dollars on the cloud infrastructure and requested this funding from Dean's Office. After this request was approved, we had two major decisions to make. We want to produce a high-end application which is possible to complete in a 10-week period, however, we did not want to offer any

provisional code. For instance, Ülgen could have a feature to notify users about incoming help, or collaboration of routing vehicles, however, this would add so much complexity that we cannot produce production level tested code to the users. For such sensitive aid applications, we think delivering non-tested code is unethical, that's why we always wanted to offer fully working functionalities. Furthermore, we also wanted to create trend topic analysis with Twitter Developer API to detect anomalies, nonetheless, after the project started, we realised that Twitter has changed their terms and conditions to use the API, and this feature is no longer possible.

For any aid application, toleration of failures is an important factor. If one of your services fails, this should be handled both in server-side and client-side. Thus, Ülgen is developed using microservice architecture to provide resiliency to the system. In total there are 4 microservices that we have developed: Member API, Producer API, Consumer API, and Routing API. While separating code into different microservices, we have thought about the development process of those API's. For instance, we always seek the answer to the question: is this process large enough to turn into a microservice on its own? Furthermore, we have containerized every microservice with Docker, and software we used to prevent ease of deployment and version conflicts. Docker Compose was responsible for the management of containers and their interaction, and if any container fails, Docker Compose was handling the restart. In the next paragraphs, we will be explaining each microservice, and devote the last paragraph to the security of Ülgen.

Member microservice is developed to authenticate and authorize requests and serves as an entry point to the Ülgen system. Using this API, we can perform all user operations, as well as communicating with other microservices. To establish a connection to this microservice, we have obtained a custom domain called "ulgen.app", and we have created a subdomain called "api.ulgen.app" to use it as a REST API. Furthermore, due to the sensitive data transferring, we ensured that encryption is applied. Thus, we setup a secure connection to member microservice by obtaining a certificate using Let's Encrypt. These certificates are inspected and renewed when expired by Certbot.

Producer microservice is responsible for producing data to Kafka. Kafka is a distributed stream-processing platform. One can think that it is a database for real-time streaming events with high throughput. We have used and tested Kafka's ability to handle large streaming loads, which we will be also mentioning in the later section. Producer microservice is being called from member microservice, and the communication is handled by Feign Client in Spring Boot.

Consumer microservice is implemented to create consistency of Kafka. We are using batch processing of data and storing them in our classic PostgreSQL database. This microservice's main purpose is to manipulate and configure the data and alter batch processing options if the applications are developed even further, or the number of active users increases.

Routing microservice is the place where we offer operational research solutions. We used Google-OR Vehicle Routing algorithm as a boilerplate, however, we have changed inputs by manipulating distance matrix. Before running the routing algorithm, we are running an unsupervised clustering algorithm called DBSCAN. This algorithm requires an epsilon value that determines the sizes of clusters, and right now we are using epsilon to be 0.02. This epsilon is not an optimal value, and we do not think optimal value exists due to the difference between building types between cities, or even between zones in the same city. After clusters are determined, we are getting centroid data, and priority of each cluster. Furthermore, we are building a priority matrix. At the end, we used the following mathematical approach to update our distance vector: $d' = mp + nd$, where m and n are arbitrary constants that sum up to 1, and p and d are a priority and distance vector, respectively. By using d' instead of d, we can change Google-OR's routing algorithm to satisfy our needs. Moreover, the question we frequently got in the progress meetings as well as the demo day was the adaptation of routing. We are aware that roads and other constraints change significantly during and after disasters. Ülgen relies on Google Maps API to handle those changes. Google Maps API dynamically analyses and updates their routing solutions by retrieving data regarding the current conditions of the roads and traffic.

The hardest part between building and deploying microservices was to obtain security in a way that protects user's sensitive data, while keeping availability of applications as high as possible. The first action we made was removing secure connections between microservices and achieving protection using Firewall. Since we do not want anyone to directly access any of our microservices other than Member API, we forbid access to all the ports except the Member API. This approach decreases the time it takes to communicate between each microservice dramatically. The second action was using JWT, to authenticate and authorize the users. We also keep track of JWT tokens in our database, which slowed us down, however, gave us the ability to revoke user tokens in case of detection of fraud. The third action was to forbid access to the HTTP (Port 80) to the forgot password and verification website that Ülgen has. With NGINX, we have been watching all the traffic that URL obtains and mapping them to the HTTPS (Port 443). By doing this, we ensured that people who are not experienced with technology were not able to click phishing links and were not targeted by the

man in the middle attacks. Lastly, even though we were in the virtual private cloud in AWS, we still used environment variables to inject our sensitive configuration strings such as API keys, and passwords to our Docker containers. Furthermore, we separated the front end of Ülgen website with the back end to isolate both from each other.

4. Analysis and Results

While designing the architecture of Ülgen, we wanted to build a production-level product. With our load testing with K6, we proved that architecture was not only concrete, but also software such as Kafka were a perfect choice.

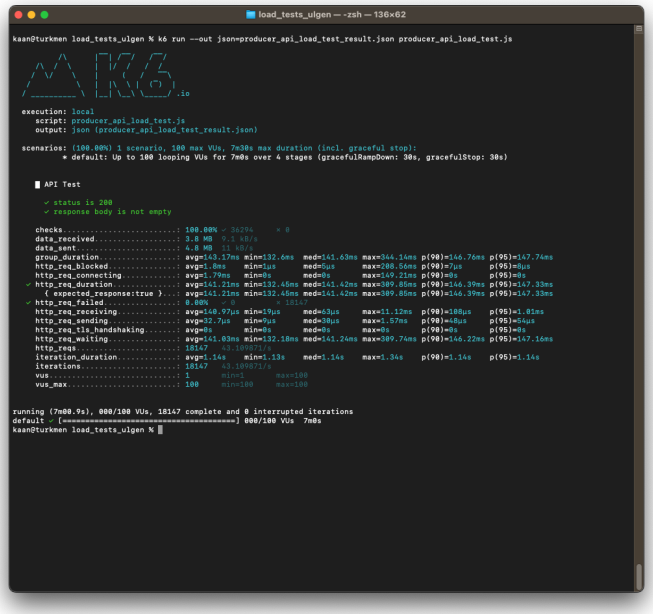

Figure 3. Load Test of Producer API.

As seen in Figure 3, when we run a load test for 7 minutes and approximately 100 concurrent users send requests to our microservice, we are getting an average response time of 141.21 milliseconds. If we used a traditional database instead of Kafka, it was not only impossible to get such results, but it would also cost much more to us in a cloud environment.

The Android mobile application development process for Ülgen was primarily focused on ensuring functionality and usability. The process began with the creation of the application's user interface in Figma, a popular UI design tool. Designing in Figma allowed us to create a UI that was intuitive and eye-catching. After the design is finished, it is exported and integrated into the application using Android Studio and Kotlin, Model-View-Controller (MVC) architecture is

followed.

Testing of the application took place on both an Android emulator and an actual Android device. This ensured the compatibility of the app across different devices and versions of Android, confirming its robust performance in various scenarios.

The application's functionalities such as user authentication, activity monitoring, disaster alerts, heatmap, display of recent earthquakes, and optimal route guidance were tested thoroughly. Each function communicated effectively with the respective server-side microservices - Member API, Producer API, Consumer API, and Routing API.

During user authentication, the Member API successfully verified the user's credentials and returned a JWT token, which was securely stored on the user's device after they successfully logged in. This token was effectively used for all subsequent API interactions, validating the session security process. We also assigned unique avatars to each user to create a personalised and enjoyable experience.

For the activity monitoring feature, the application effectively detected significant changes in the user's activity through device sensors and communicated the data to the Producer API, validating the real-time activity tracking functionality. We send the user's location information and MAC addresses which are located in the user devices' local network to our Producer API and use this information on other features such as heatmap and disaster alerts. We send this information to our back end periodically, using Android's Worker library so that if the user's connection to the internet is cut due to a disaster, we could estimate the location and the number of users near the device. Furthermore, if the user is secured by the authorities, they can stop sending their data by clicking the "I am Safe" button in our app, which can also be seen in Figure 10.

The application successfully received and displayed disaster alerts from the Consumer API in real time, affirming the effectiveness of our disaster alert system. In addition, the application has a heatmap feature where the affected areas and the number of users affected are represented in a map. This map's data is generated via sending user's data to the Producer API and the map itself is drawn using Google Maps API. The results can be seen in Figure 5.

In the vehicle information screen, we are obtaining vehicle count, depot location, selection of affected cities, and slider to determine the priority count or distance to the affected users. Using these data, they can create a routing data according to their preference. This process can be in Figure 6. The optimal route guidance feature, powered by the Routing API, accurately provided users with the safest and quickest routes to predetermined safe zones during a simulated disaster scenario, demonstrating the efficiency and practicality of the application in emergency situations. While doing this, we are only transferring data of affected areas to enhance user privacy. Furthermore, users cannot use disaster-time functionalities if no disaster is reported. The result of the optimal route guidance feature can be seen in Figures 7 and 8.

In our app, we also provide a feature where users can observe the recent earthquakes that happened in Turkey. We use Kandilli Rasathanesi API for fetching the data and displaying it in our app. The result of this can be seen in Figure 9.

5. Conclusion

In conclusion, our project Ülgen aimed to deliver a secure, reliable real-time data processing system for effective disaster management. The successful implementation and testing of Ülgen's architecture and its Android application have demonstrated its robustness and its ability to handle high data loads in various scenarios. However, despite achieving our design goals, there is still room for improvement. Right now, we have Member API as an entry point, however, this creates a single point of failure. The solution is straightforward: using Kubernetes and configuring multiple replica deployments. However, this adds a huge cost to the system, that’s why we did not implement Kubernetes for this project. In addition, since the data we send is highly sensitive, Android restricts the access to the MAC addresses which reside in the device's current local network. Therefore, when we acquire the MAC addresses, they might not be 100% accurate because of Android’s strict policy. To overcome this problem in the future, we came up with a solution which includes making arrangements with service providers in order to improve the accuracy of our app. While developing Ülgen, we wanted to ensure that it fits in a 10-week period of implementation time and testing end-to-end to produce production-level application. Thus, Ülgen could have more features and better architecture, however, tests would leave incomplete, and we did not want untested features for such a sensitive product. Furthermore, we are optimistic about future improvements to enhance Ülgen's architecture, making it even more reliable and invaluable during crises.

6. References

1. Communications and Publishing, “The M7.8 and M7.5 Kahramanmaraş Earthquake Sequence struck near Nurdağı, Turkey (Türkiye) on February 6, 2023.” usgs.gov https://www.usgs.gov/news/featured-story/m78-and-m75-kahramanmaras-earthquake-sequence-near-nurdagi-turkey-turkiye

2. Market share of mobile operating systems in Turkey. statista.com https://www.statista.com/statistics/1316551/turkey-market-share-of-mobile-operating-systems/

3. AKUT, “Güvendeyim.” akut.org.tr https://www.akut.org.tr/guvendeyim

4. AFAD, “AFAD Acil Mobil Uygulaması.” afad.gov.tr https://istanbul.afad.gov.tr/afad-acil-mobil-uygulamasi

5. Network connectivity by Region - Turkey. netblocks.org https://twitter.com/netblocks/status/1622544305331077120

6. “Turkey's New Earthquake Hazard Map is Published.” afad.gov.tr https://en.afad.gov.tr/turkeys-new-earthquake-hazard-map-is-published

7. Appendix

As developers, we have open sourced all implementation, deployment, and additional resources such as Javadoc websites in our Github repository. They can be accessed by https://github.com/UlgenApp

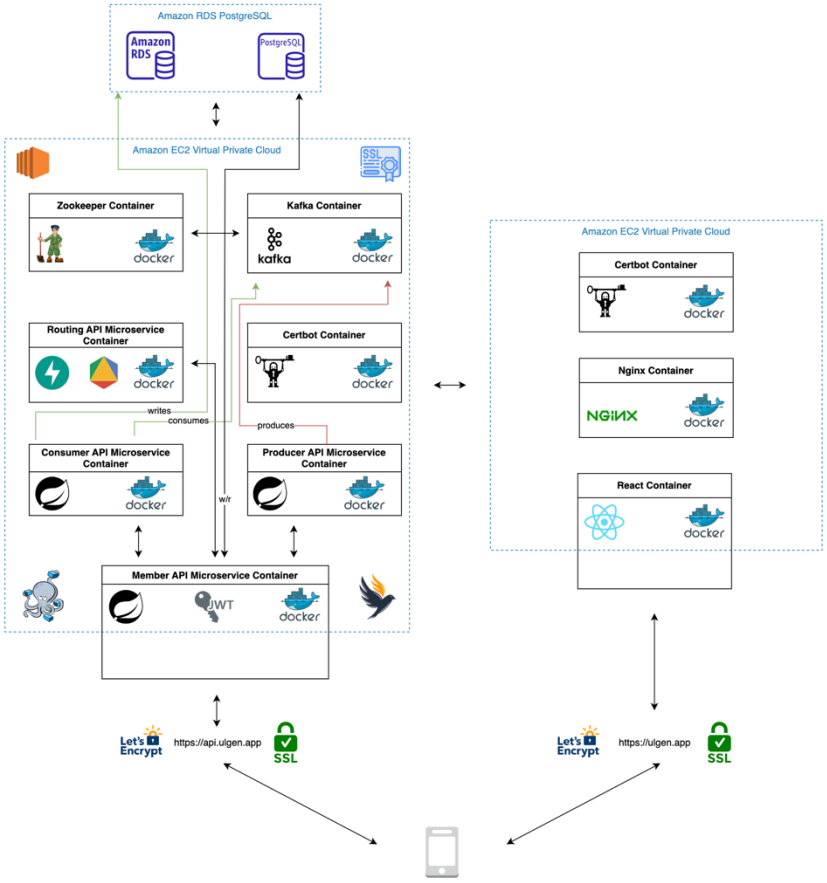

Figure 4. System Architecture.

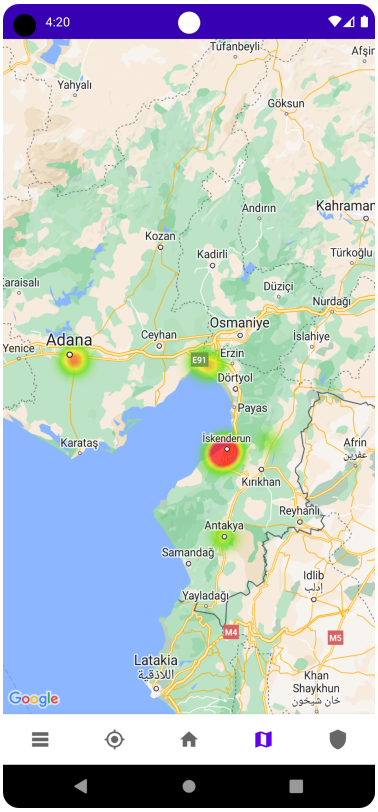

Figure 5. Heatmap Display on App.

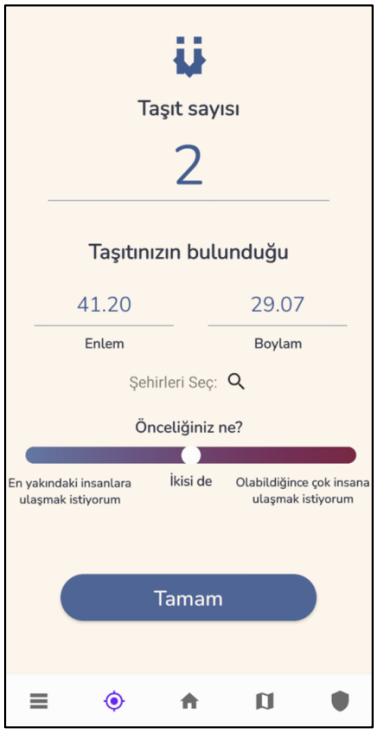

Figure 6. Vehicle Information Screen.

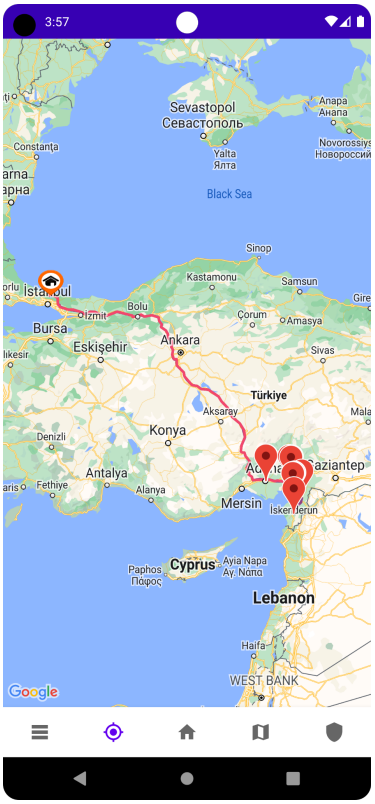

Figure 7. Routing Guidance on App.

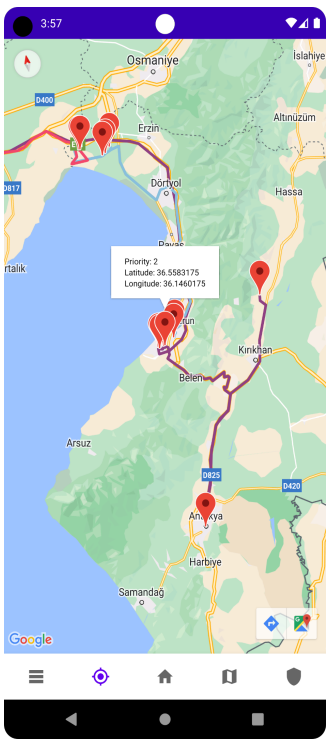

Figure 8. Closer Display of Routing Guidance.

Figure 9. Recent Earthquakes Display.

Figure 10. I am Safe Button.

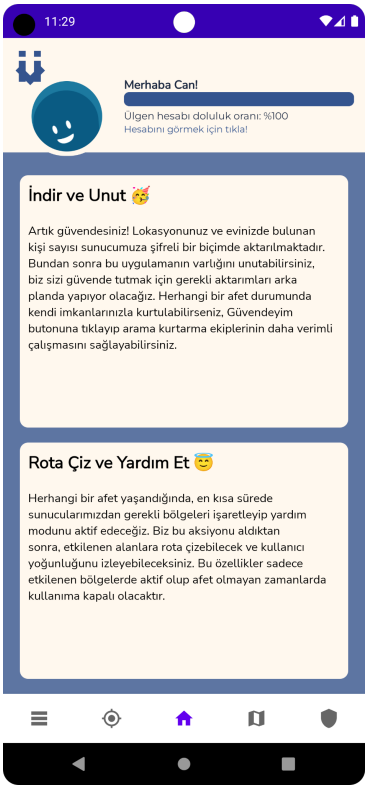

Figure 11. Main Menu Screen.

Figure 12. User Profile Screen.

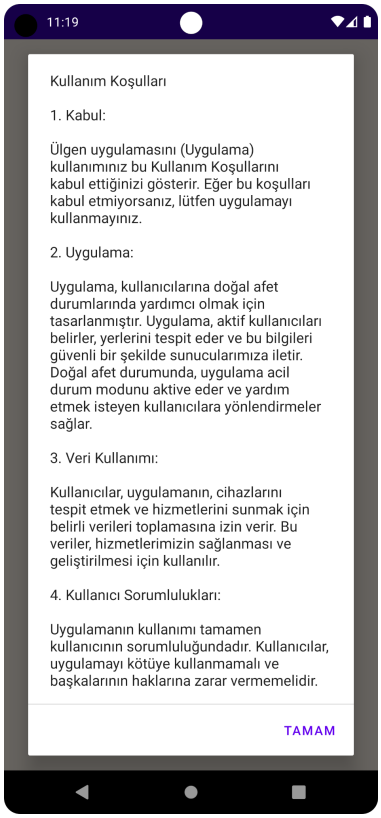

Figure 13. Terms of Use.

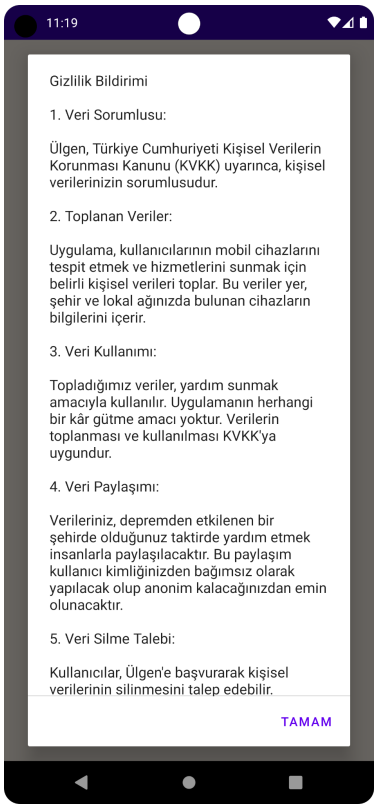

Figure 14. Privacy Notice.

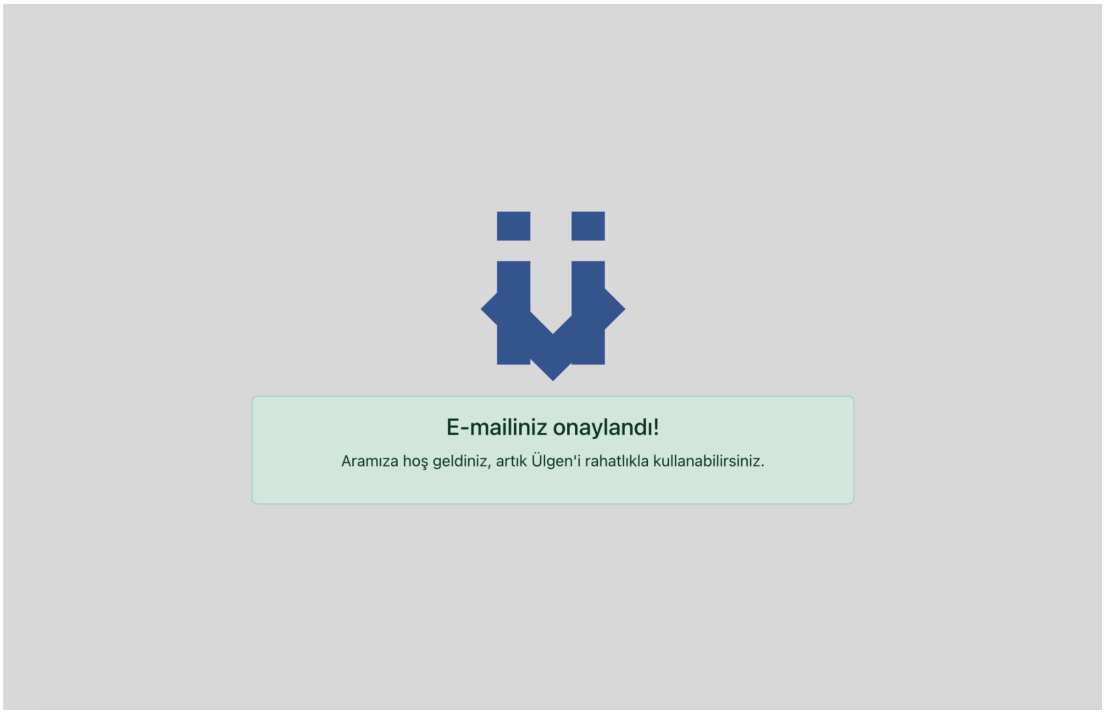

Figure 15. Email Verification Page.

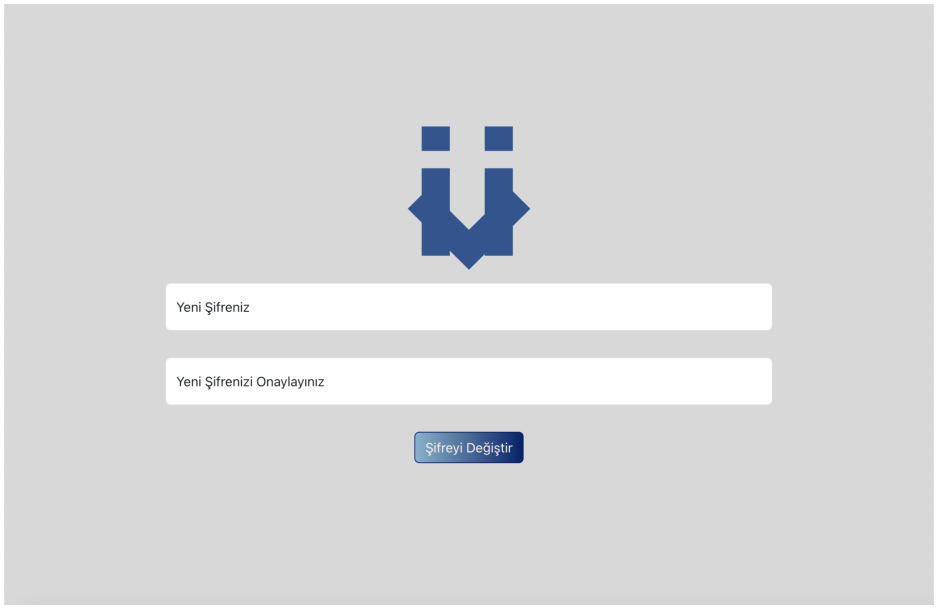

Figure 16. Reset Password Page.

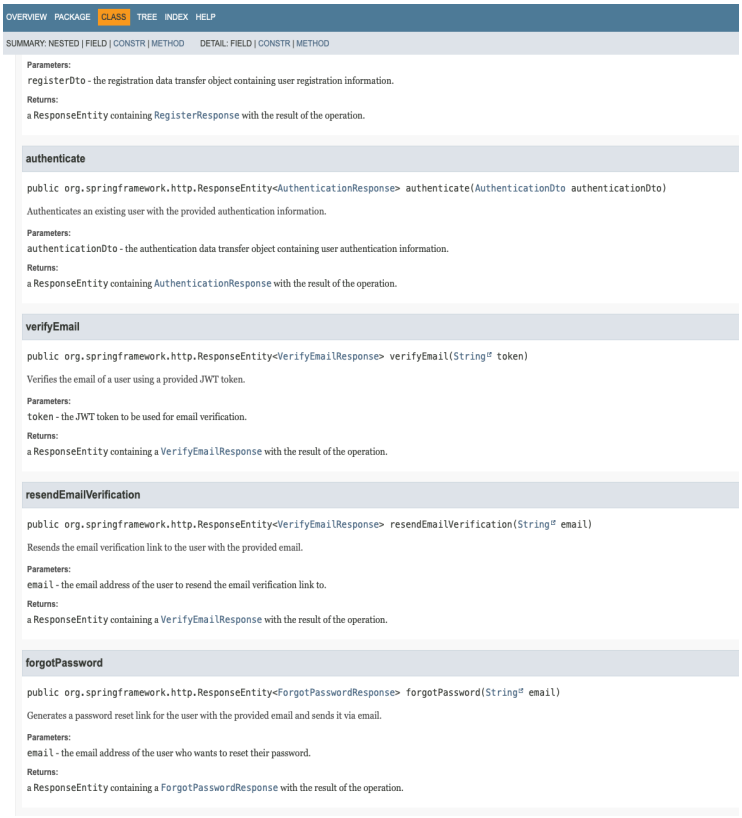

Figure 17. Member API Javadoc.