SOFTWARE ARCHITECTURE DESIGN

Afetbilgi Website

Group No:	40
Hashem Qaryouti	e2652816
Lala Salimli	e2653160

Table Of Contents

List of Figures

List of Tables

1 Introduction

This document is a Software Architecture Design of afetbilgi.com. The website' URL is afetbilig.com.

1.1 Purpose and Objectives of afetbilgi.com

Aftebilgi.com is a website that is prepared by a group of Middle East Technical University students and graduate to verify important information in the fight against the 6 February earthquake disaster and to deliver it to both victims and those people and organizations who want to help.

1.2 Scope

The scope of the system can be listed as follows:

• The website provides details about the active Hospitals in the districts those affected by the earthquake such as the location of the hospital, telephone number, last time to be updated and the services that are provided.

• The website shows the available pharmacies in those areas affected by the earthquake such as, in which city and district the pharmacy is located and a map for it.

• The System enables the user to look for a veterinary clink and provides detailed information about it such as name and phone number.

• The system provides details about safe gathering places in case of earthquakes and evacuation points.

• The system enables the user for looking to temporary accommodation places with information such as location, address and link for it.

• The system provides information about institutions announced to provide transportation aid for those people affected by the earthquake such as validation date of the service.

• The system offers information about food distribution centers.

• The system provides information about blood donation places

1.3 Stakeholders' and their Concerns

The stakeholders of the system can be listed as follows:

- **Earthquake Victims:** The primary concern of earthquake victims is to receive timely and accurate information about the disaster, access essential resources such as food and medical aid. They may be also interested in finding opportunities for connection or volunteering with other affected individuals to share experiences and support one another.
- **Website Administration and Developers:** The individuals responsible for maintaining and managing the website have concerns regarding the website's functionality, security and reliability. The need to ensure that the website can handle traffic. Provide accurate information, facilitate smooth communication and protect sensitive user data.
- **Researchers:** The website can provide a platform for researchers to collect data related to the earthquake and its impact on the affected population. They can gather information about needs of the victims. Researchers can leverage the information, resources and connections available on the website to conduct meaningful studies, collaborate with other researchers, contribute to the knowledge base in disaster management.

2 References

- ISO-IEC-IEEE-4210-2022 International Standard -Software, systems, and enterprise Architecture description

3 Glossary

URL	Uniform Resource Locator
ISO	International Organization for Standardization
IEC	International Electrotechnical Commission
CMS	Content Management System
PDF	Portable Document Format
AWS	Amazon Web Service
HTTP	Hypertext Transfer Protocol
YAML	Yet Another Markup Language
VPN	Virtual Private Network
CI	Continuous Integration
CD	Continuous Deployment
S3	Simple Storage Service

Table1: Glossary

4 Architectural Views

4.1 Context View

4.1.1 Stakeholders' uses of view

In this viewpoint, context of the system with all actors are defined in general and detailed viewpoints. In the context diagram, actors and their interaction with the system will be explained in general terms.

4.1.2 Context Diagram

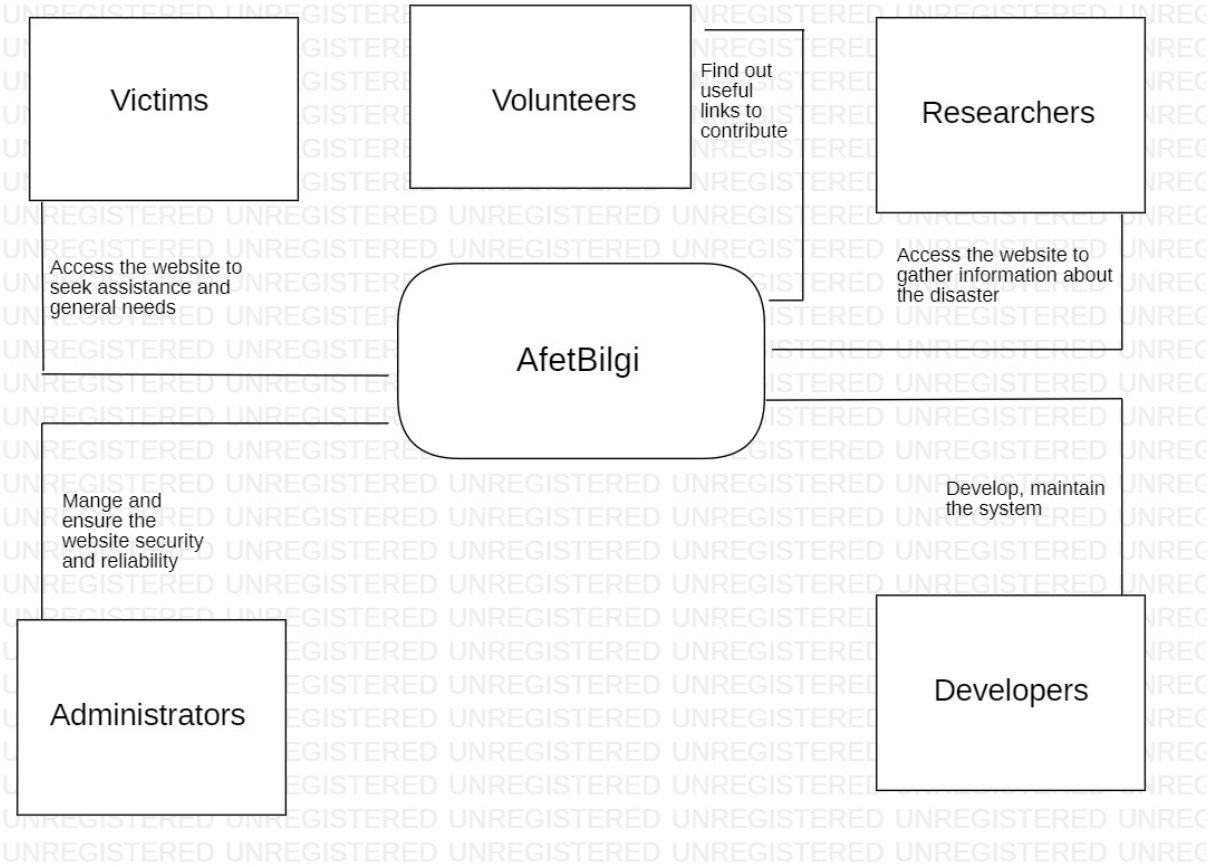

Figure1: Context diagram

Victims access the website to find information and resources to meet their immediate needs. They may access the website through various devices, including computers, smartphones, or tablets, using an internet connection.

Volunteers may access the website to assist earthquake victims. Although the website does not offer direct help for victims, volunteers can access the website to find opportunities to contribute, such as a bunch of links to other resources.

Developers access the system to perform tasks such as designing and implementing user interfaces, developing the website's functionality and ensuring security and data protection.

Administrators may be responsible for managing and overseeing the operations and content. They have the capability to manage and update the website's content. This includes editing, and removing information, resources, articles, and announcements to ensure the website provides up-to-date information to earthquake victims and other users.

4.1.3 External Interfaces

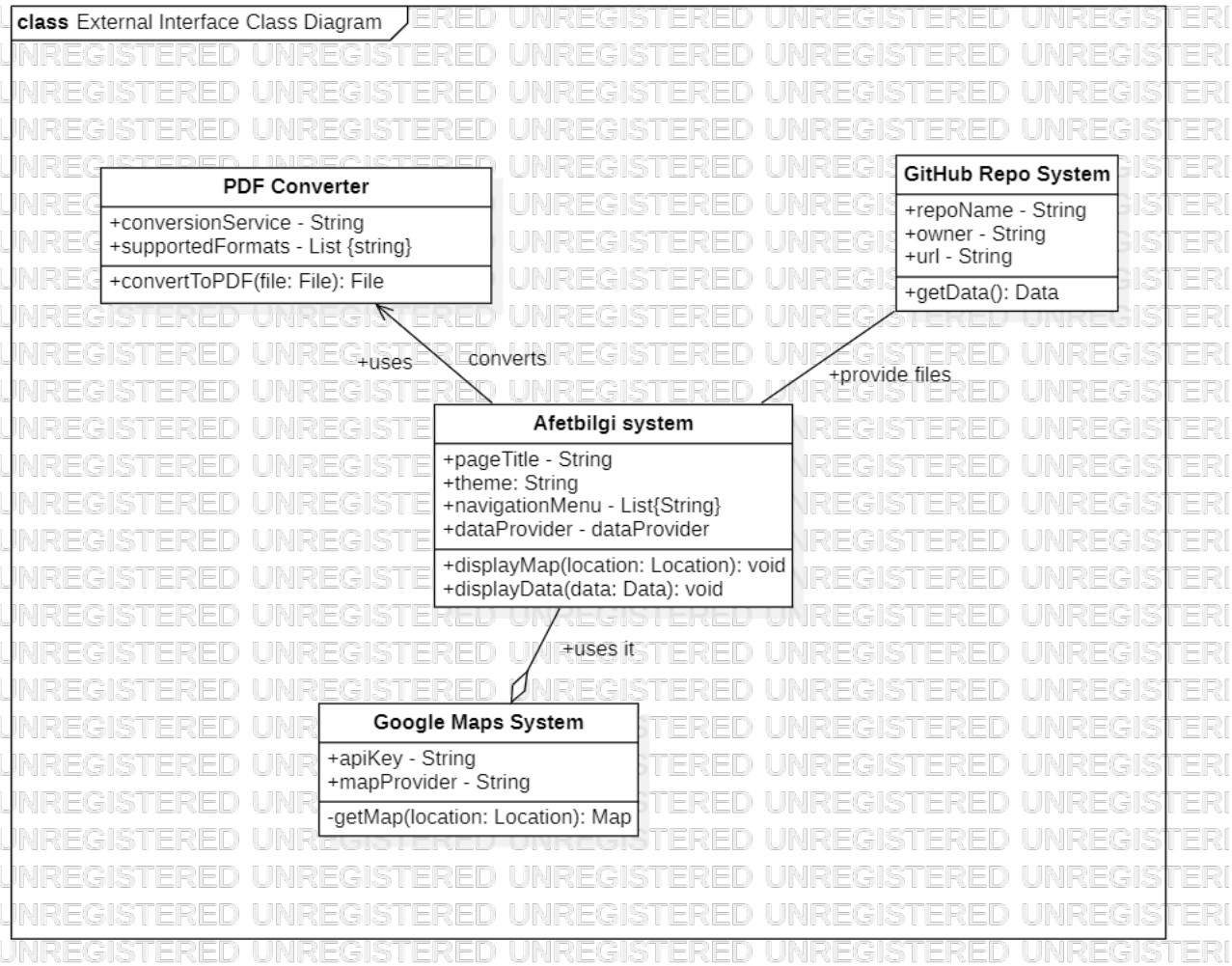

Figure2: External Interfaces Class Diagram

Interface between Google maps and Afetbilgi system

In the interaction of these systems, google maps is an external interface. It provides the website with mapping and location features. When a user wants to see the facilities in the map uses the map feature. This interface in this case enables the completion of the process.

Interface between GitHub Repository and Afetbilgi system

In this interface the GitHub Repository is an external platform. GitHub Repo offers features for code management, also hosts repositories for version control. This interface is usually used by the developers. When they want to fetch code updates, deploy changes etc. they direct to this interface.

Interface between PDF converter system and Afetbilgi

PDF converter is an external interface in this interface. When the users use the pdf feature this interface makes turns the data to the pdf format for users. This interface allows website to receive converted pdfs from the PDF converter and display/download them for users

4.1.4 Interaction Scenarios

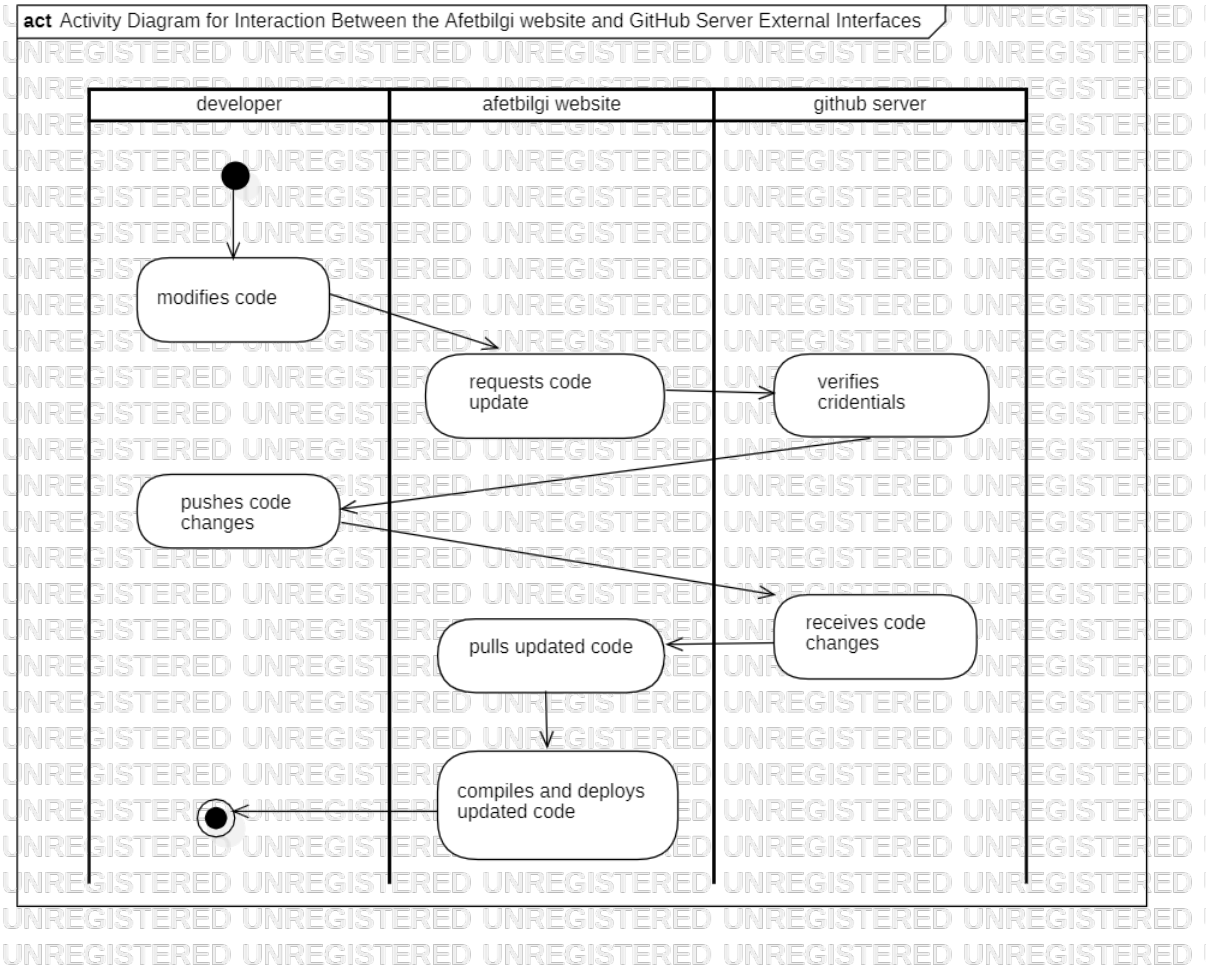

Figure3: Activity Diagram for Interaction Between the Afetbilgi website and GitHub server External Interface

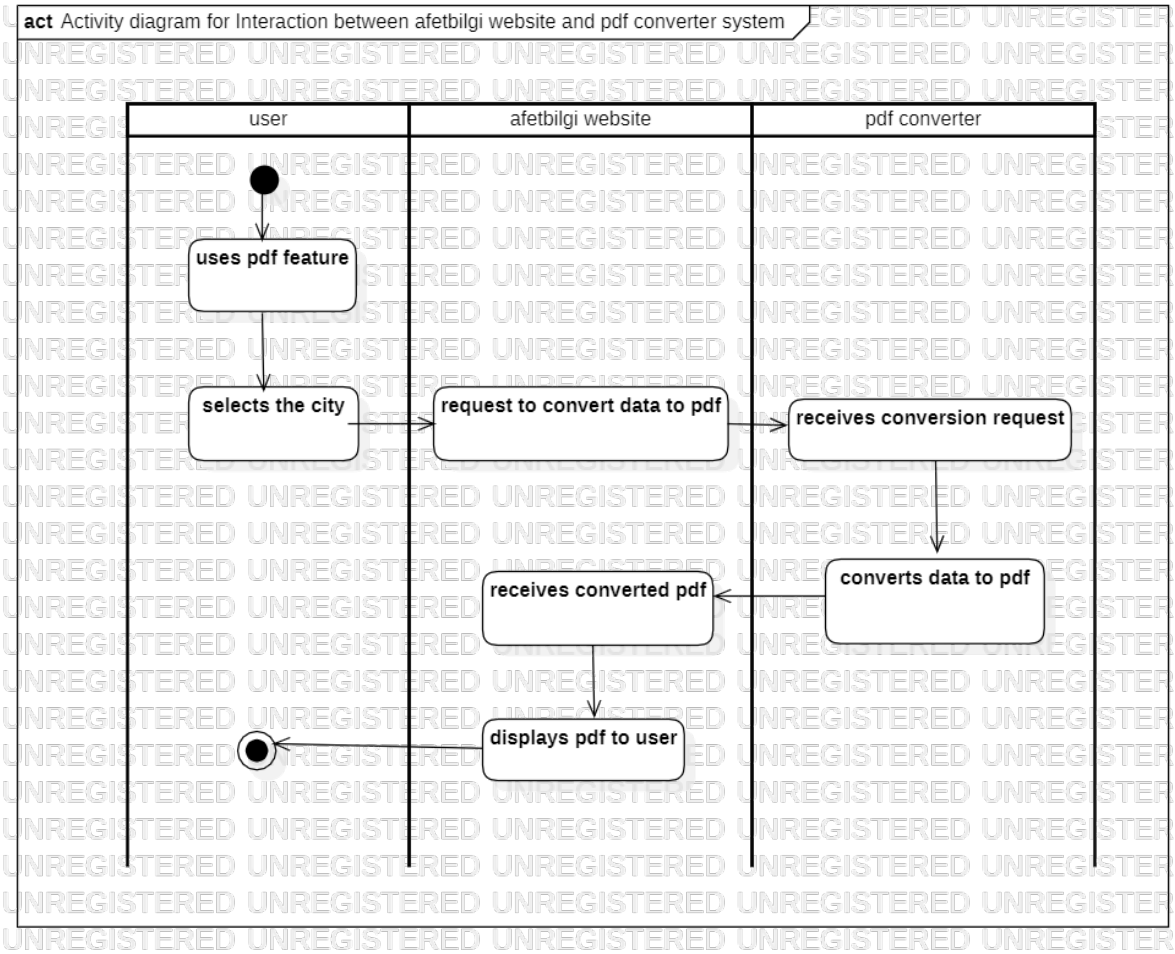

Figure4: Activity Diagram for the interaction between afetbilgi website and pdf converter system

4.2 Functional View

4.2.1 Stakeholder's uses of view

There are several key stakeholders of the Afetbilgi website: users, developers, and researchers. In this viewpoint, different diagrams are going to be presented about the internal interfaces, interactions and so on. The functional view is highly relevant to the stakeholders of the Afetbilgi website.

4.2.2 Component Diagram

This UML diagram represents the component diagram of afetbilgi system. It consists of 6 components and 2 of them are external systems. Components and Interfaces are explained in this section.

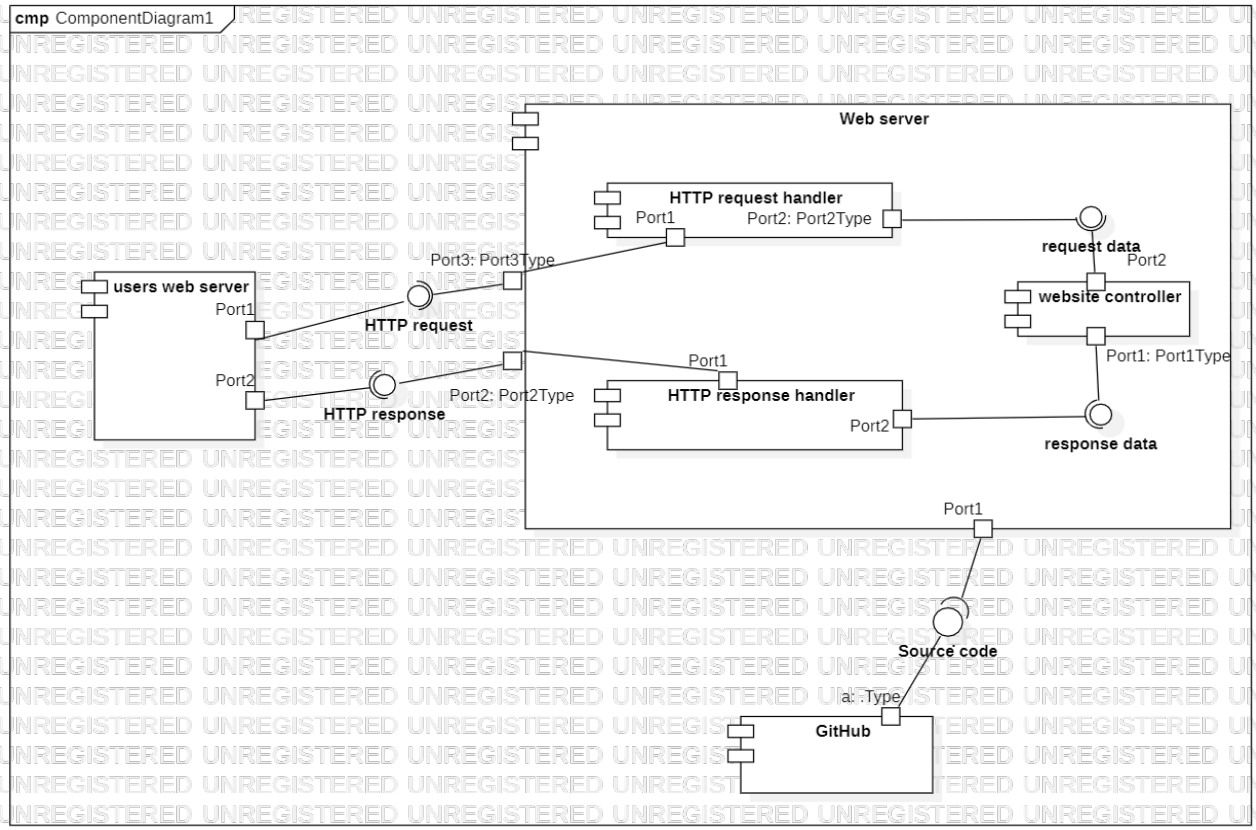

Figure5: Component Diagram

- Web server is the main subsystem, it is responsible for handling and responding to users' actions. Web server holds the HTTP

request handler, HTTP response handler and website controller.

- HTTP request handler component is responsible for handling the request coming from user and sending them to website controller.
- HTTP response handler component on the other hand, is responsible for sending HTTP responses to the user. It gets these responses from website controller.
- Website controller is responsible for getting HTTP requests from HTTP request handler and sending response to HTTP response handler.
- Users web server is an external subsystem in component diagram. It represents user's actions
- GitHub component is also an external subsystem in this diagram.

4.2.3 Internal Interfaces

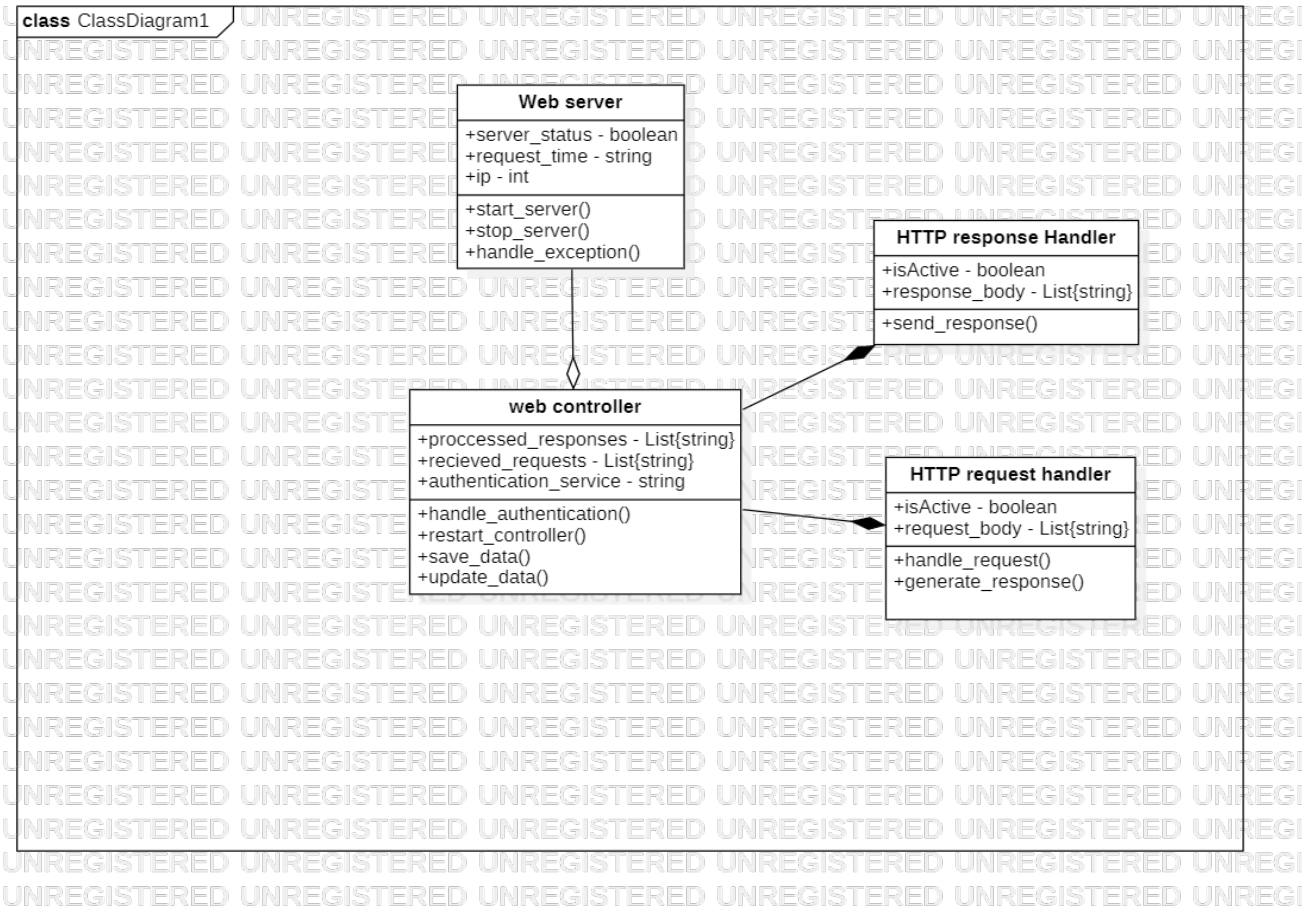

Figure6: Internal Interfaces Class Diagram

Operations	Descriptions
start_server	starts the web server and it begins to accept incoming HTTP requests, defines the required configurations
stop_server	stops the web server, closes the connection, and terminates the processes.
handle_exception	handles exceptions and errors, unexpected situations.
send_response	sets the content or body of the HTTP response
handle_request	receives incoming HTTP requests and initiates the corresponding processes
generate_response	generates an HTTP response to send back to the client
restart_controller	shuts down current controller and makes new fresh start
save_data	stores data to database
update_data	gives opportunity to user to update their data

Table 2: description of operations on internal interfaces class diagram

4.2.4 Interaction Patterns

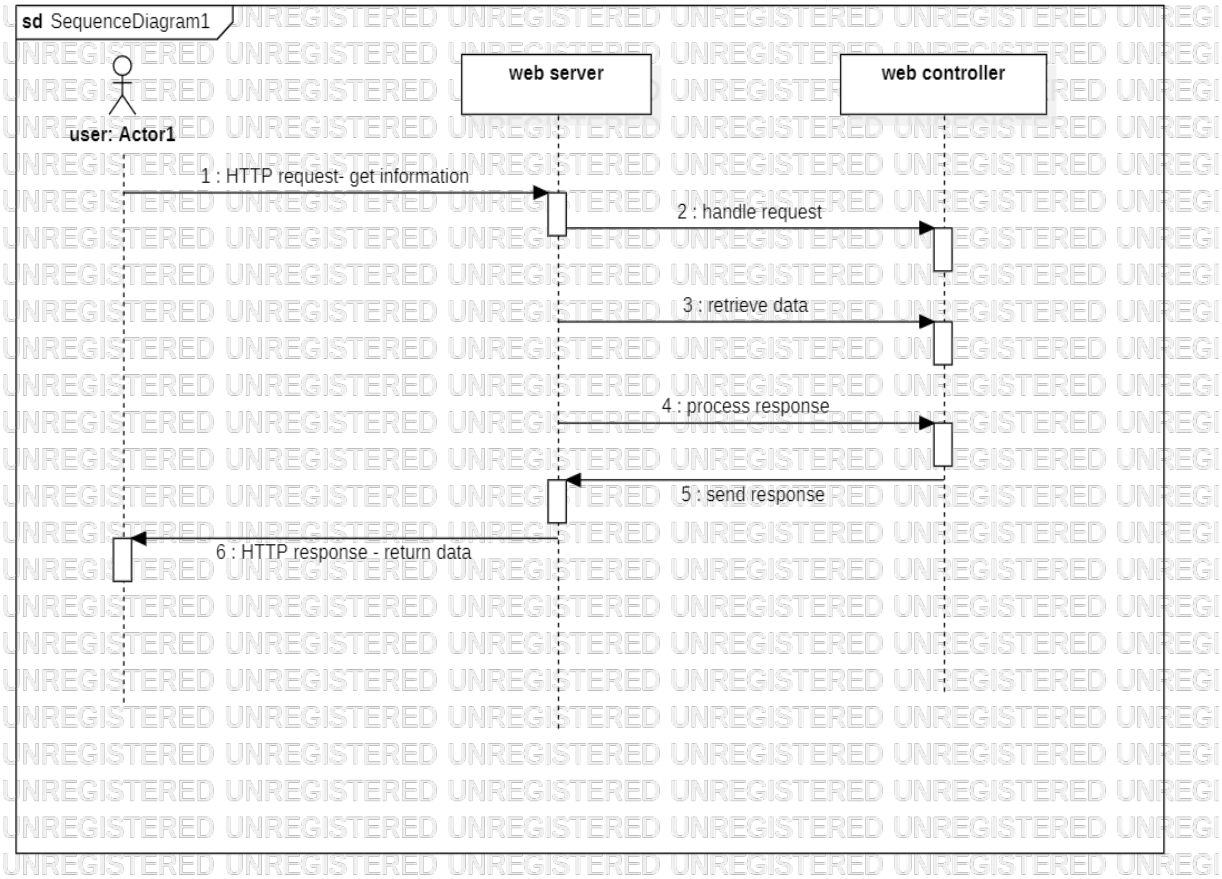

Figure7: Sequence Diagram

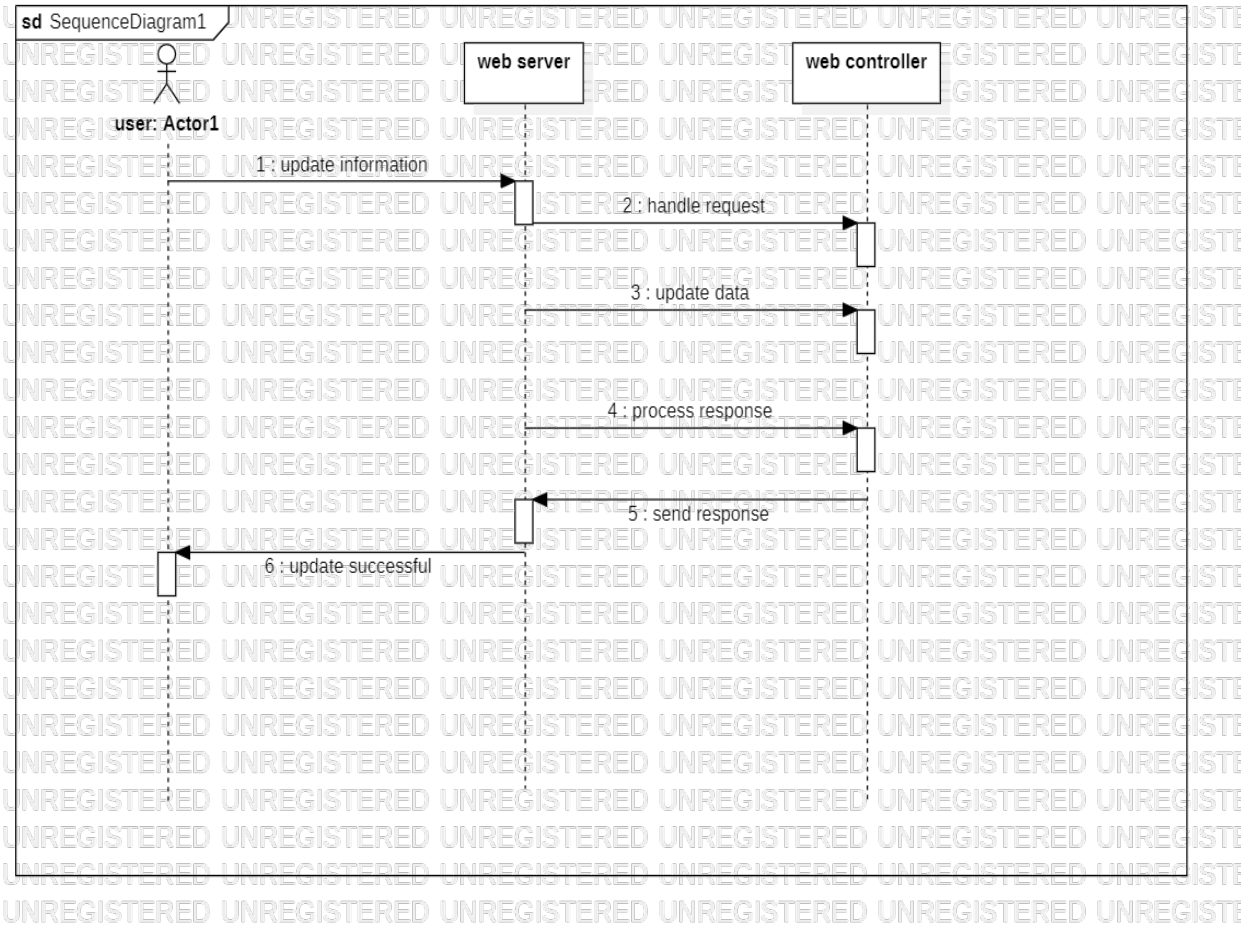

Figure8: Sequence Diagram

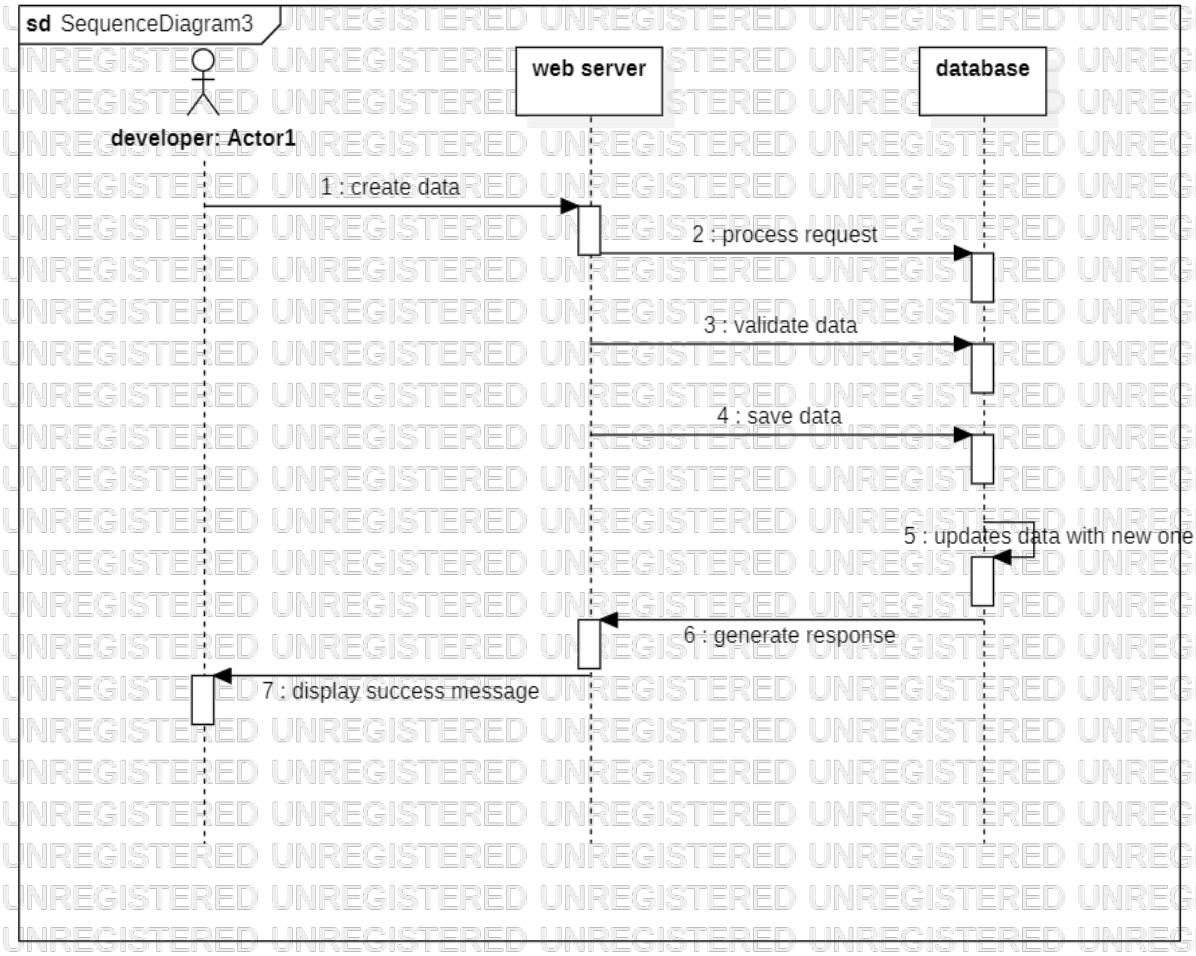

Figure9: Sequence diagram

4.3 Information Interfaces

4.3.1 Stakeholders' uses of view

This section focuses on how data is organized and related within the Afetbilgi website. It also covers operations such as creating, using, modifying, and deleting this data. The aim is to clarify the database structure and connections between different classes/entities.

4.3.2 Database Class Diagram

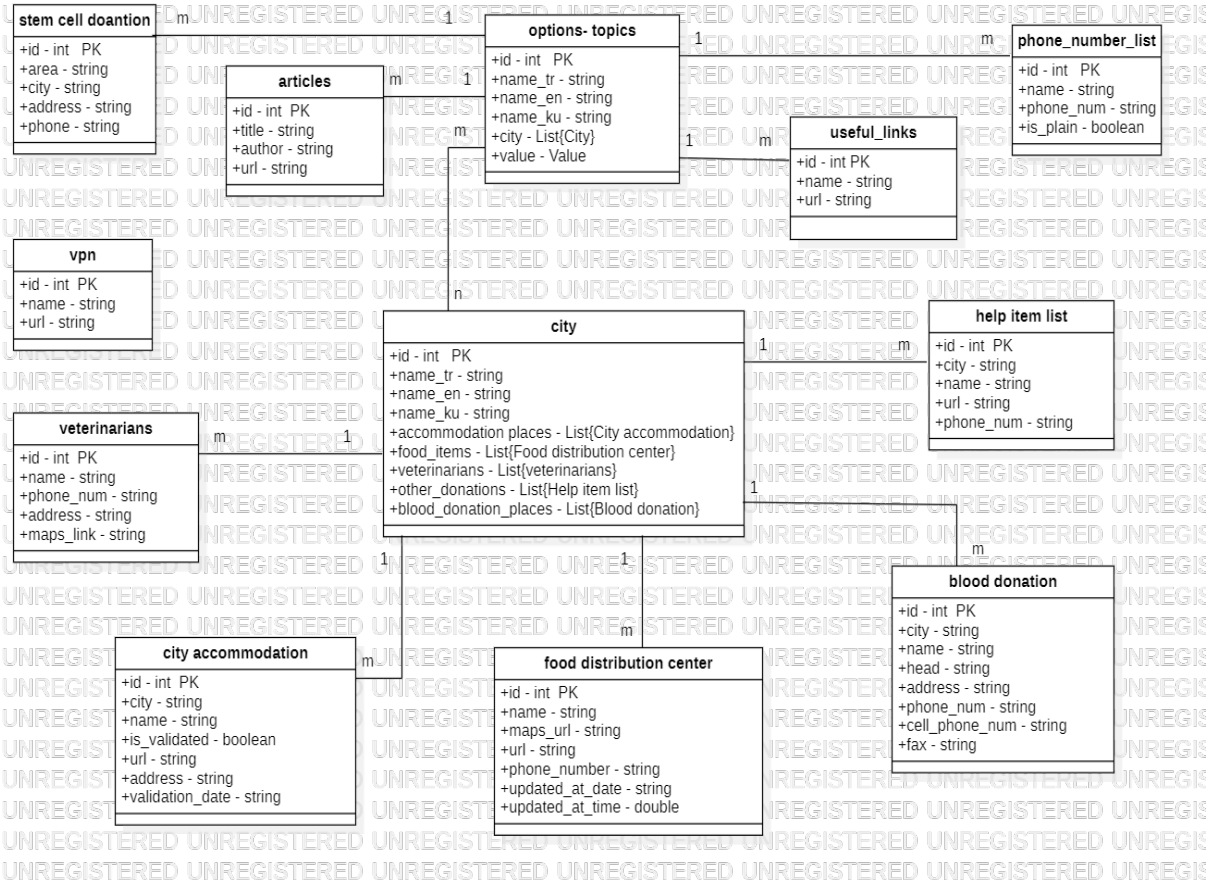

Figure10: Database Class Diagram

Options-topics	
id	0001
name_tr	"Geçici Barınma Alanları"
name_en	"Temporary Accommodation Places"
name_ku	"Bicîhbûna Demkî"

Options-topics	
id	0002
name_tr	"Faydalı Yazılar"
name_en	"Useful Articles"
name_ku	"Agahiyên Kêrhatî"

4.3.3 Operations on Data

Operation	CRUD (create/read/update/delete)
createDonationLink	Create: Donation Link Read: - Update: - Delete: -
isActive	Create: - Read: Active developers Update: - Delete: -
updateTemporaryAccommodation	Create: - Read: - Update: Temporary Accommodation Delete: -
getStemCellDonationCity	Create: - Read: City of Stem cell donation Update: - Delete: -
newCityTranslated	Create: City translation Read: - Update: - Delete: -
updateVPN	Create: - Read: - Update: vpn name, vpn url Delete: -
translateCityName	Create: - Read: city name Update: city in tr/en/ku/ar Delete: -
retrieveDonationHistory	Create: - Read: Stem Cell Donation Update: - Delete: -
checkAvailability	Create: - Read: Active Temporary Accommodation Update: - Delete: -
updateDonationLink	Create: - Read: - Update: Donation Link Delete: -
getValidationdate	Create: - Read: validation date of Temporary Accommodation Update: - Delete: -

Table3: Operations on data tab

4.3 Deployment View

4.3.4 Stakeholders' uses of view

Deployment Diagram is used to illustrate the physical deployment of software components and the relationships between them in a system. Stakeholders may use it know how software components are deployed on hardware architecture. Deployment diagram provides a visual representation how software components are interconnected and deployed across different nodes or machines.

4.3.5 Deployment Diagram

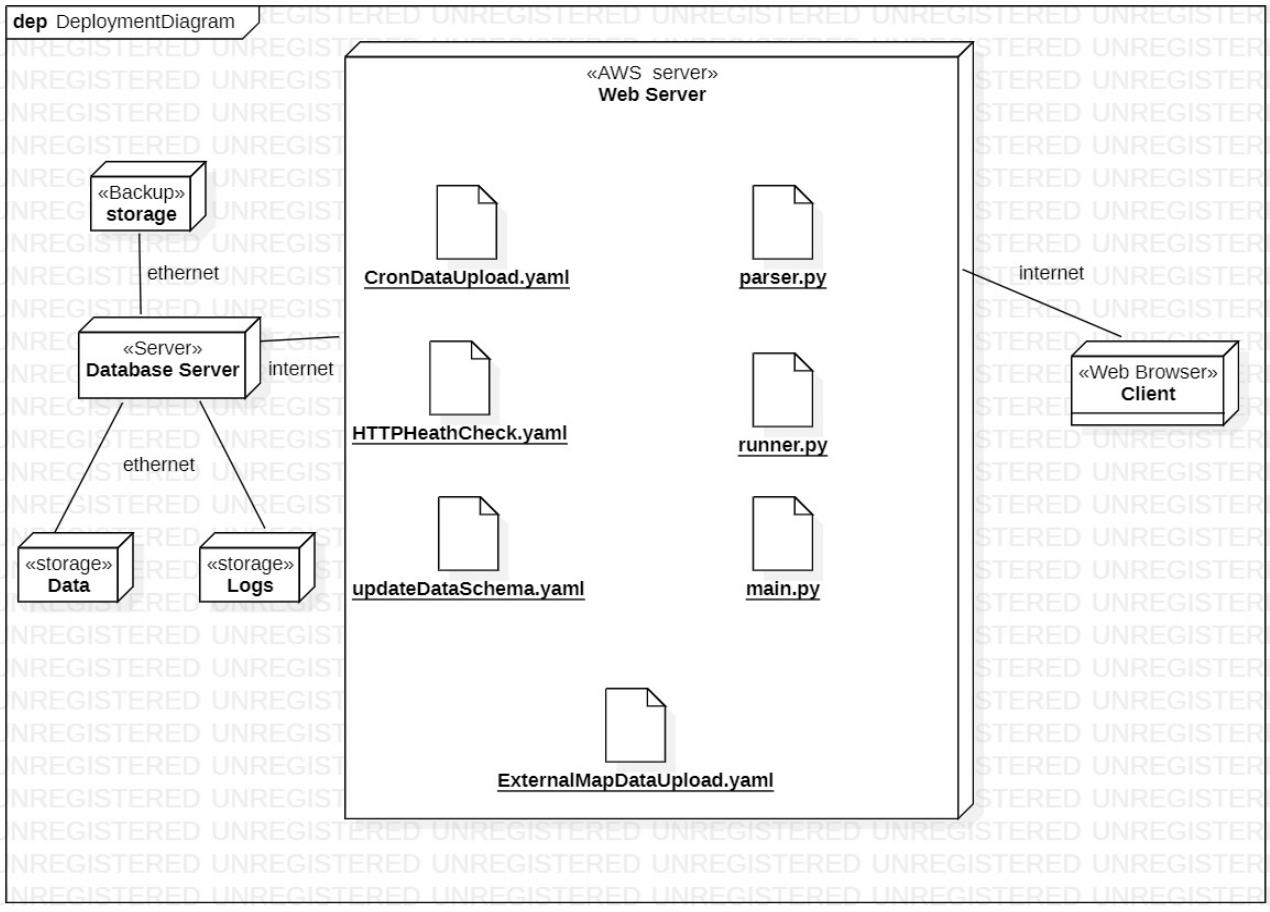

Figure11: Deployment Diagram

- There are three different database storages:
    - Data storage is for storing website's content.
    - Logs storage is for storing system logs
    - Backup storage is for storing the backup of data storage.

- The web application is deployed on AWS. The node labeled "AWS Cloud" indicates the used cloud service provider.
- The user accessibility on the web relies a simple architecture where a single server directly handled all incoming user requests.
- Kubernetes cluster is responsible for managing the deployment and operation of the web application.
- **CronDataUpload.yaml** is configuration file which is responsible for deploying and update map data.
- **Parser.py** is a python script that defines several functions for parsing and processing data from different resources.
- **Runner.py** is a python script that merges multiple PDF files within a directory and its subdirectories into a single PDF file.
- **ExternalMapDataUpload.yaml** is a configuration file that fetches JSON data from external sources, combines it and then uploads the resulting files to an S3 bucket for further use or distribution.
- **HTTPHeathCheck.yaml** is a configuration file which performs HTTP requests to the specified URLS and checks if the received HEEP status code match the expected codes. This allows monitoring the health and availability of the listed URLs.

4.4 Design Rationale

- **Context View:** The design rationale for this view ensure smooth integration, address security concerns, handle errors, and facilitate effective collaboration with external stakeholders. The design rationale of this view ensures that the website can interact with external entities.

- **Functional View:** The design rationale for this view focuses on creating a clear understanding of the system's structure, behavior, and functionality. It aims to ensure that the website's components are well-organized, modular and able to work together to achieve the desired functionalities. It considers reusability of components, allowing for future enhancements. I emphasize the management of dependencies, ensuring that components are deployed and integrated in the appropriate sequence to avoid any issues.

- **Information View:** Design rationale concerns to develop a backup and recovery strategy. Data backup is stored on the AWS cloud using simple storage service S3 in which primary data, backup, and logs are stored. Amazon CloudWatch is also used to monitor and observe capabilities to track the performance and health of various AWS resources and services. It can collect and analyze log, and events related to the website's infrastructure, services and applications.

- **Deployment View:** The design rationale for this view addresses the need for scalability and availability to handle potential spikes in website traffic during emergency situation. It considers cloud-based infrastructure to ensure that the website can handle increased demand without performance degradation or downtime. The website is deployed on the cloud using Kubernetes technology which provides features such as replication controllers, self-healing and automatic failover, which enhance fault tolerance and high availability.

5 Architectural Views for Suggestions to Improve the Existing System

5.1 Context View

5.1.1 Stakeholder's uses of this view

In this viewpoint, context of the system with all actors are defined in general and detailed viewpoints. In the context diagram, actors and their interaction with the system will be explained in general terms. This context diagram shows a couple of changes and suggestions regarding the actors and their interaction within the system.

5.1.2 Context Diagram

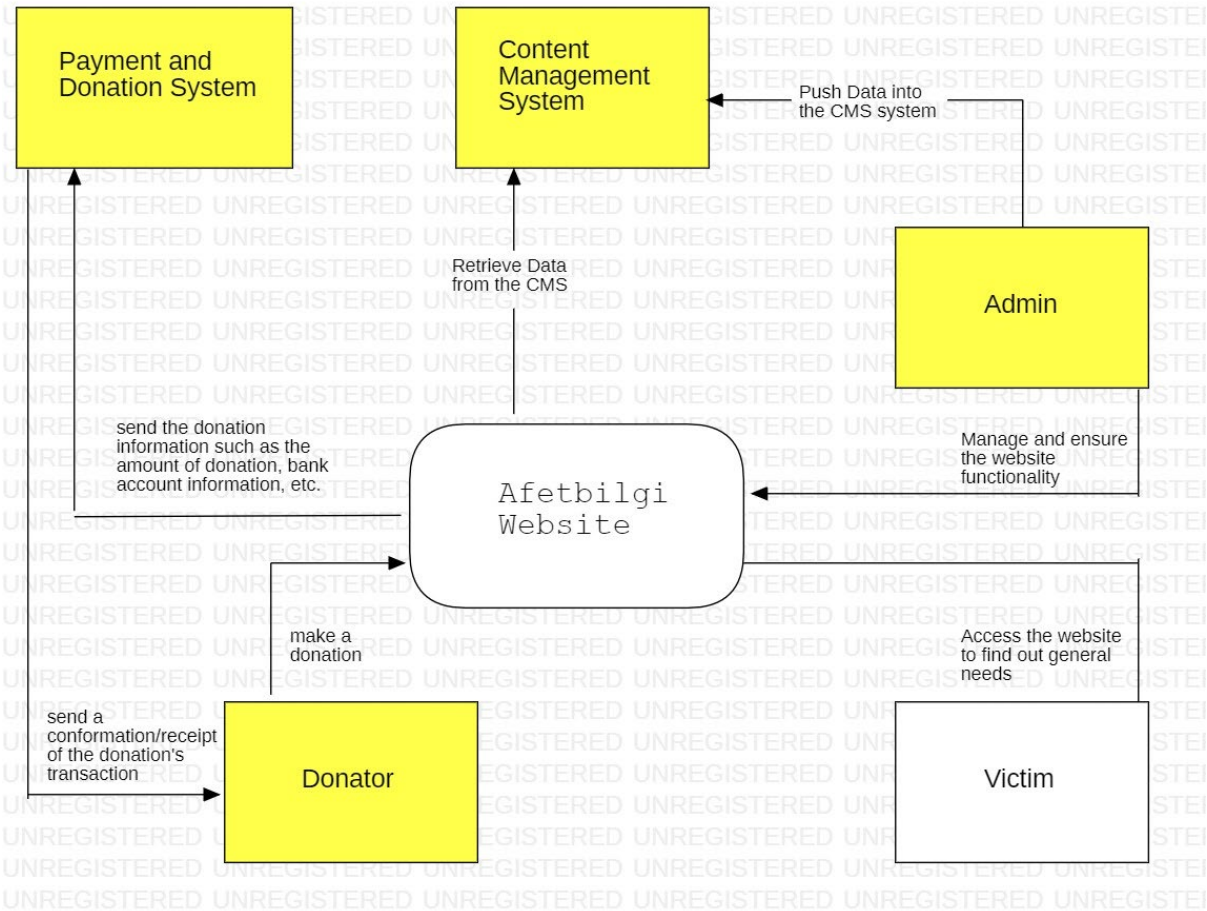

Figure12: Context Diagram

Content Management System: it is a system which enables the website administrators to create, manage, and organize the content of the website. It allows them to update information, add resources and maintain the overall structure of the website.

Payment and Donation System: This system is responsible to process the payment transaction securely using the payment/donation information provided by the main system. It confirms the successful completion of the payment transaction and notifies the main system about it. It is also going to send the confirmation/receipt for the donator.

Donator: They may access the website to make some donations and to offer help for the victim affected by the earthquake.

Admin: may be responsible for managing and overseeing the operations and content. They are responsible to manage the website's functionality and reliability. They have the right to make changes to the website's contents by using the content management system (CMS). This includes (IT stuff, manager stuff, etc.,).

5.1.3 External Interfaces

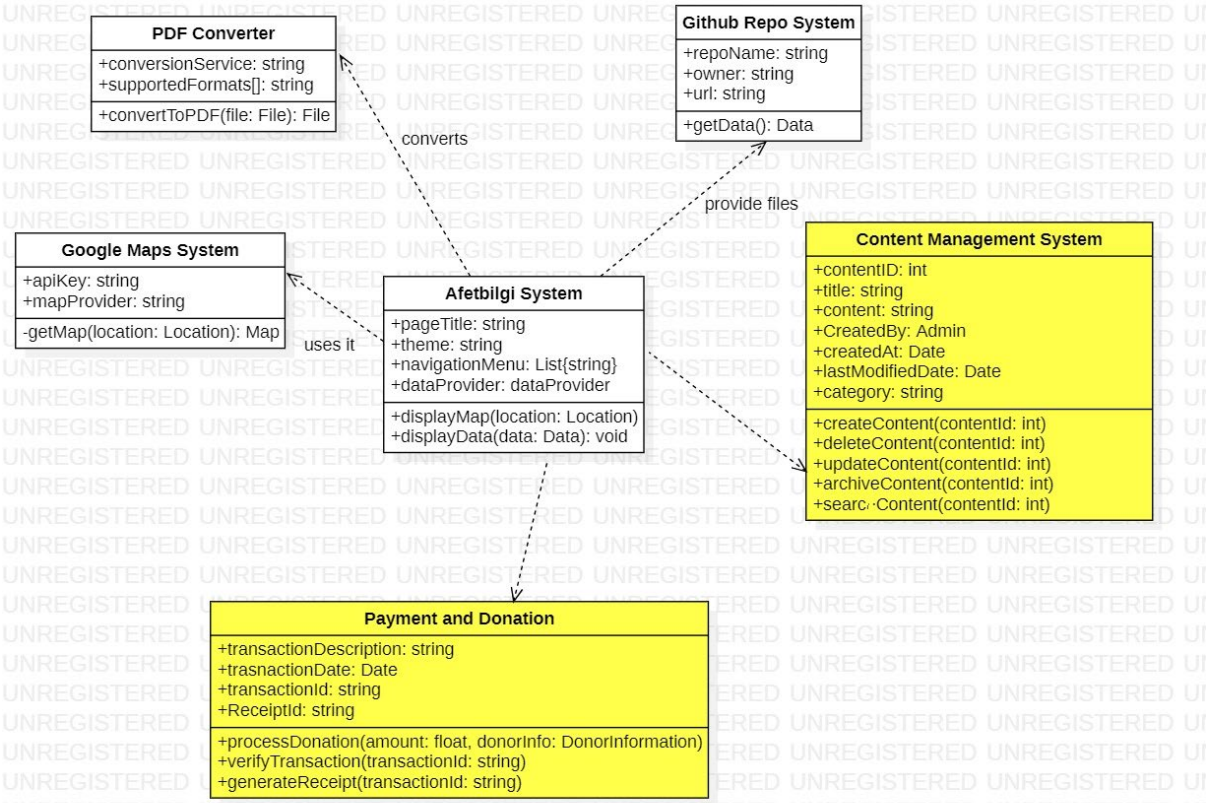

Figure13: External Interfaces Class Diagram

Operation	Description
createContent	Create new content for the main website
updateContent	Update website's contents
deleteContent	Delete website's contents
processDonation	This method is responsible for processing a donation transaction, including handling donor information and donation amount.
verifyTransaction	This method is responsible for verifying the status of the transaction, to ensure its competition and integrity. It may involve checking transaction details, status code.
generateReceipt	This method generates a receipt for a specific transaction.

Table4: External Interface Operations descriptions

5.1.4 Interactions Scenarios

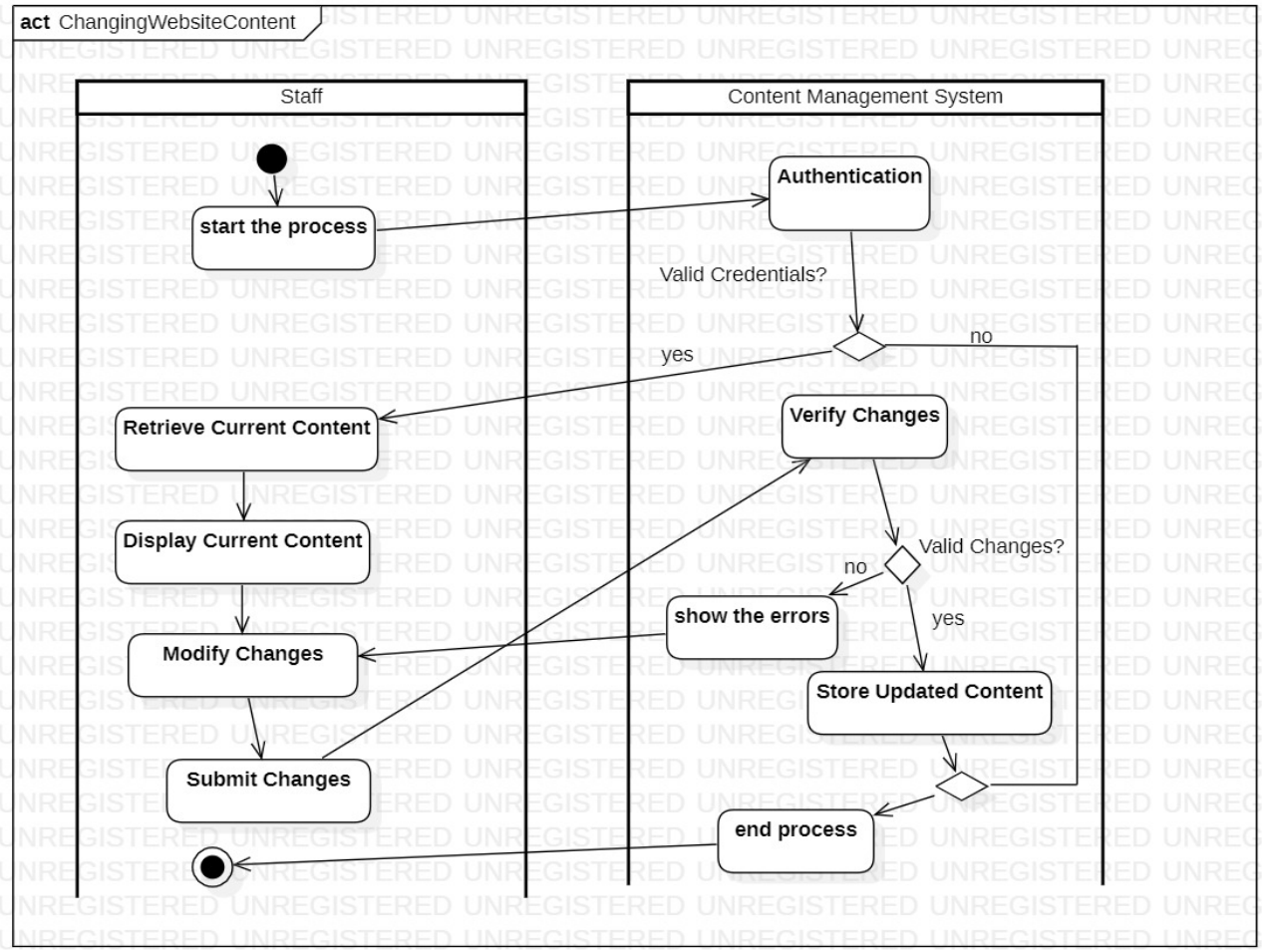

Figure14: Interaction Scenarios Activity Diagram

5.2 Functional View

5.2.1 Stakeholder's uses of view

In this viewpoint, updated and improved version of components, interfaces, and interactions are displayed. Provided with diagrams and regarding detailed explanations.

5.2.2 Component Diagram

This UML diagram below represents the component diagram of afetbilgi website with suggestions and improvements. It consists of 12 components, 5 of them are external systems, others are subsystems. The detailed explanation is given in this section about each component.

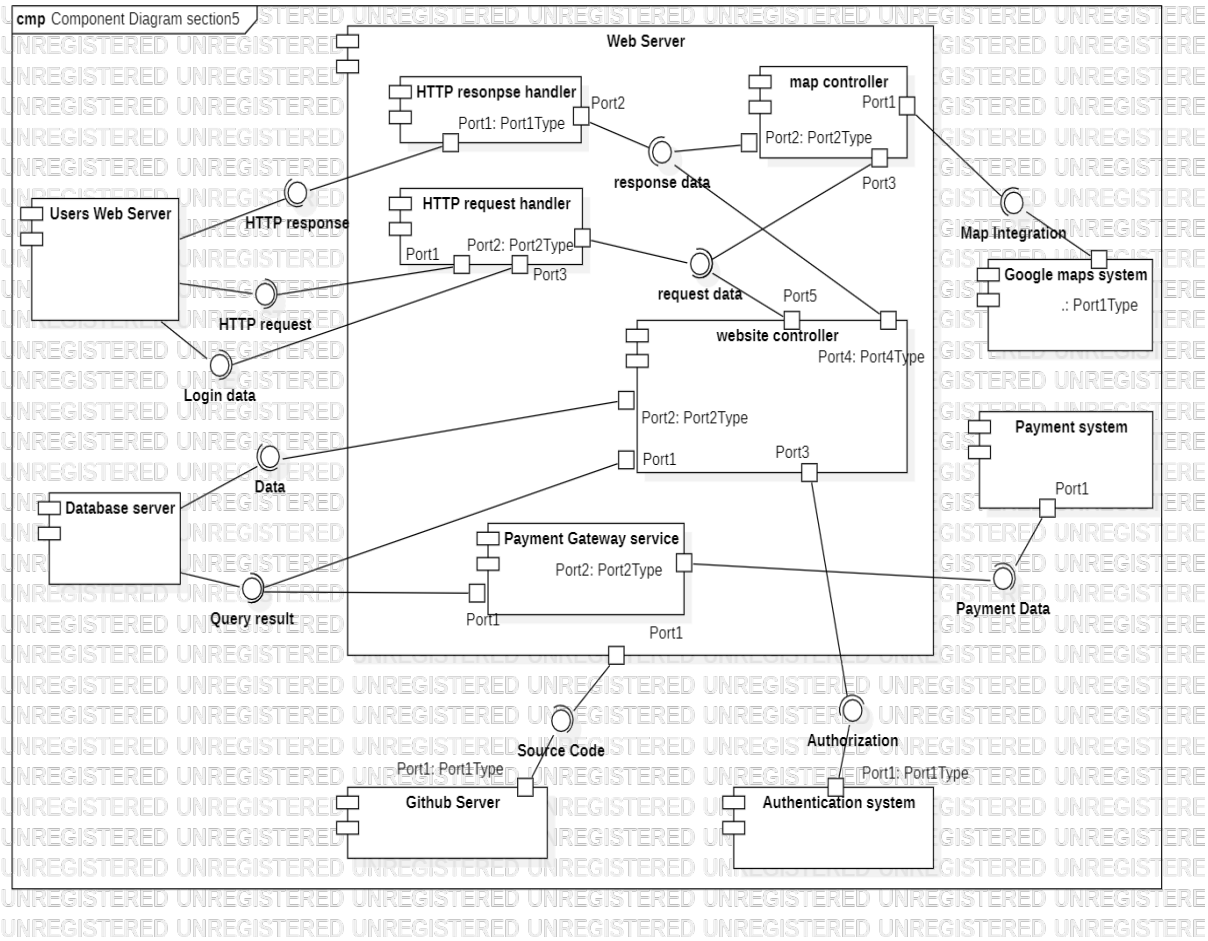

Figure 15: Component Diagram

- Web server is one of the main subsystems, it is responsible for handling and responding to users' actions, getting and sending data to database server, handling with external systems.
- Web server consists of 5 components: HTTP request handler, HTTP response handler, website controller, Map controller and Payment Gateway service.
- HTTP request handler component is responsible for handling the request coming from user and sending them to website controller and map controller.
- HTTP response handler component on the other hand, is responsible for sending HTTP responses to the user. It gets these responses from website controller and map controller.
- Website controller has few responsibilities. It is getting HTTP requests from HTTP request handler and sending response to HTTP response handler. It also gets query results and sends data to database server. It also interacts with authentication system.
- Map controller is responsible for getting and sending HTTP request/responses related to maps. It also interacts with Google maps system for providing access to its services and features. It sends request to Google Maps API and gets responses.
- Payment Gateway service is responsible for dealing with payment processes. It sends request (payment data) to Payment system (bank system).
- Database server is another subsystem. It stores the data coming from web controller and send query results.
- Users web server is an external subsystem in component diagram. It represents user's actions.
- GitHub server is an external subsystem in this diagram, it provides source code for web server.
- Authentication system guarantees the security by regulating and controlling access to the resources within the system. It interacts with the web controller to authenticate and authorize user requests.
- Google Maps System also an external subsystem, it displays map and provides some functionality by interacting with map controller.

- Payment system is another external subsystem it processes and handles payments.

5.2.3 Internal Interfaces

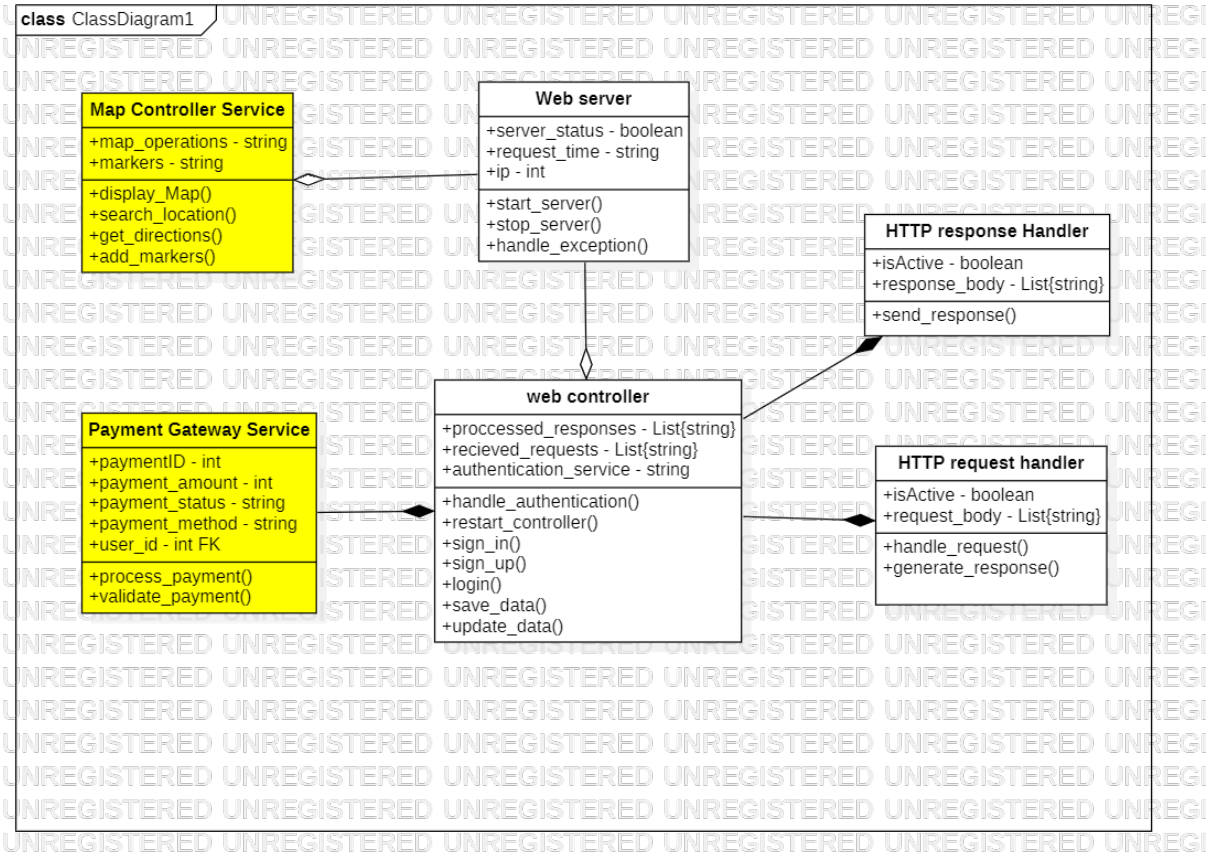

Figure16: Internal Interfaces Class Diagram

5.2.4 Interaction Patterns

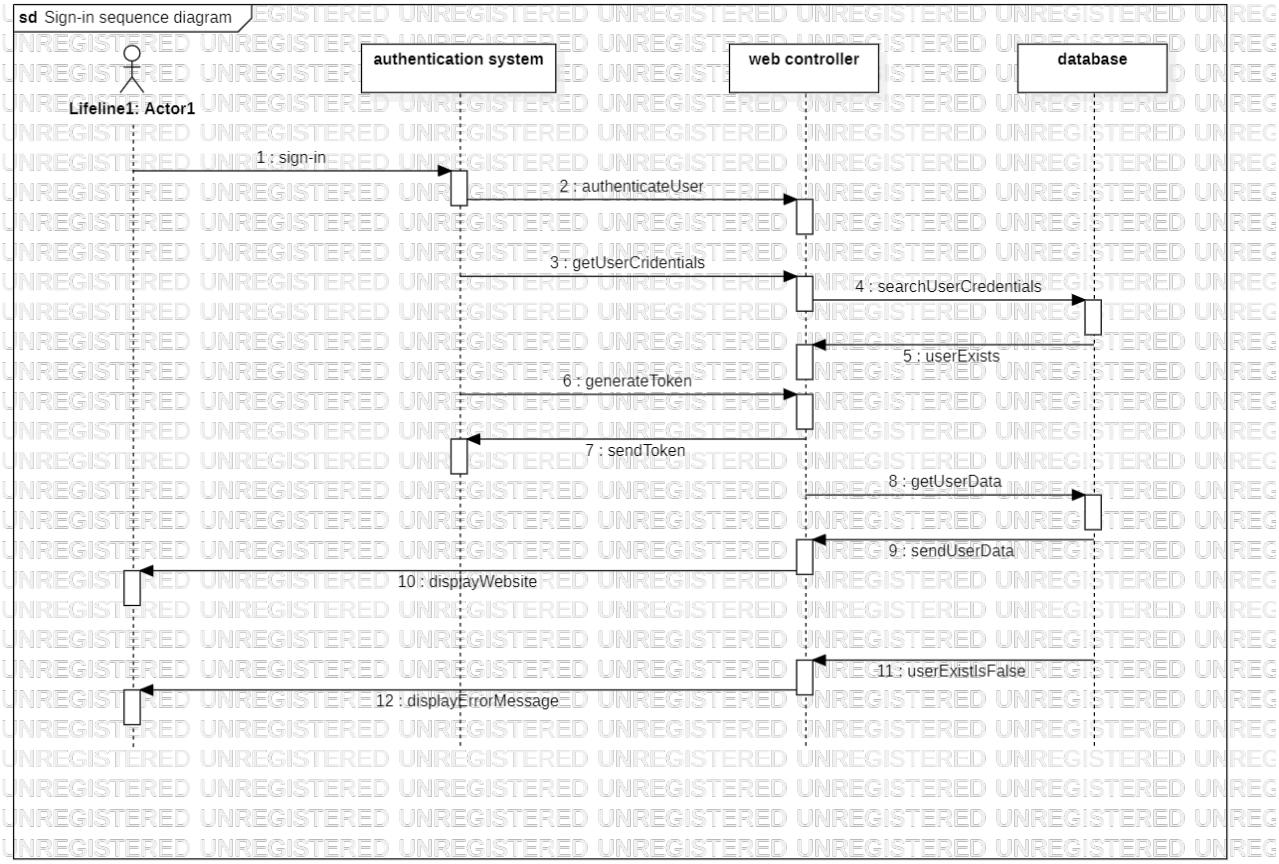

Figure17: Sequence Diagram of Sign-in feature

5.3 Information View

5.3.1 Stakeholder's uses of this view

In this view, the database class diagram of the new updated version of the afetbilgi website represented. Additions are made to database class diagram in section 4 according to our suggestions. Detailed explanations can be found in this section.

5.3.2 Database Class Diagram

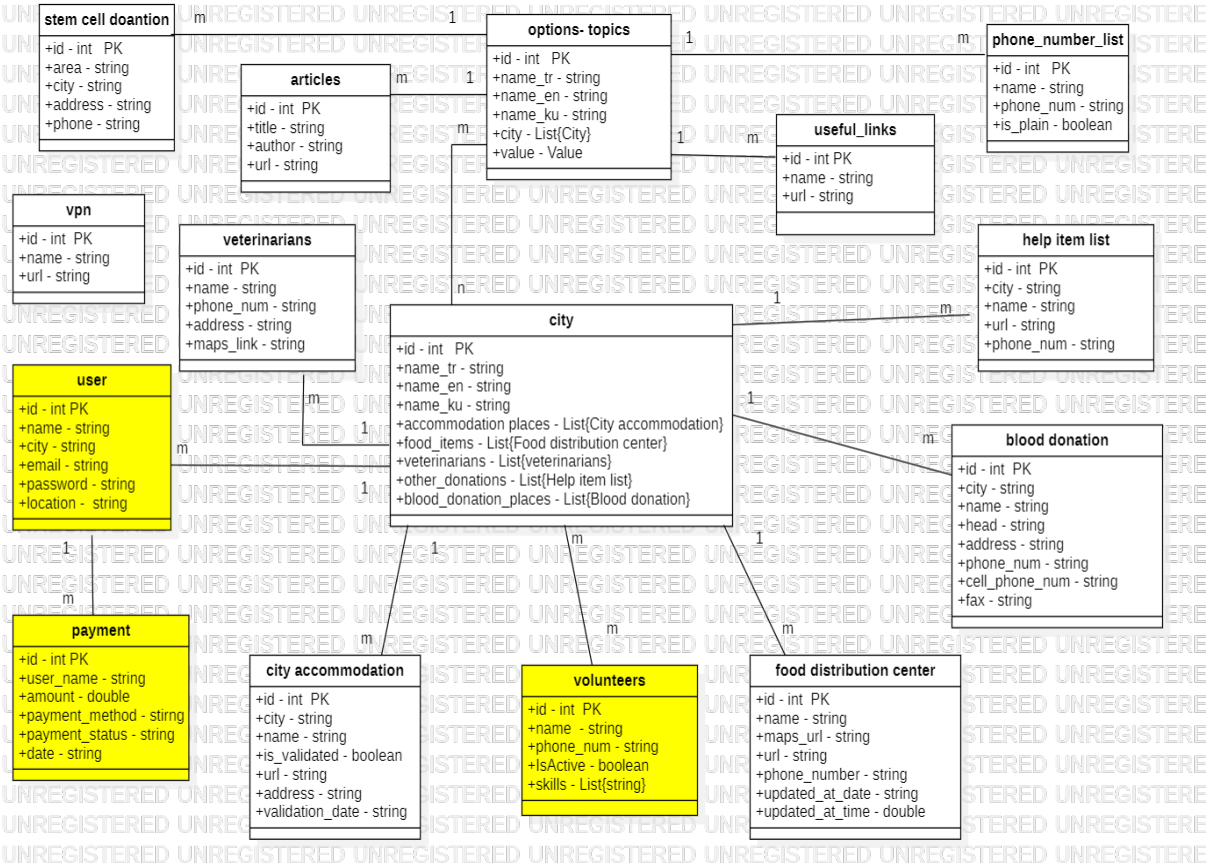

Figure 18: Database Class Diagram

5.3.3 Operations On Data

operation	description
registerVolunteer()	Create: Creates a new volunteer Read: - Update: - Delete: -
updateSkills()	Create: - Read: - Update: updates the skills of volunteer Delete: -
getContactDetails()	Create: - Read: contact details of volunteer Update: - Delete: -
newUser()	Create: creates new User Read: - Update: - Delete: -
loginUser(email, password)	Create: - Read: - Update: - Delete: -
updateLocation()	Create: - Read: - Update: users can update their location Delete: -
makePayment()	Create: adds new payment Read: - Update: - Delete: -
getStatus()	Create: - Read: gets the status of payment Update: - Delete: -
updatePassword()	Create: - Read: - Update: users can update their password Delete: -
isActive()	Create: - Read: check if volunteer is active Update: - Delete: -

Table5: Operations on data

5.4 Deployment View

5.4.1 Stakeholder's uses of this view

In the view, stakeholders may use deployment diagram for further system's enhancement. It will provide them to gain visibility into the system's infrastructure as well as it shows a visual representation of how system's components are deployed.

5.4.2 Deployment Diagram

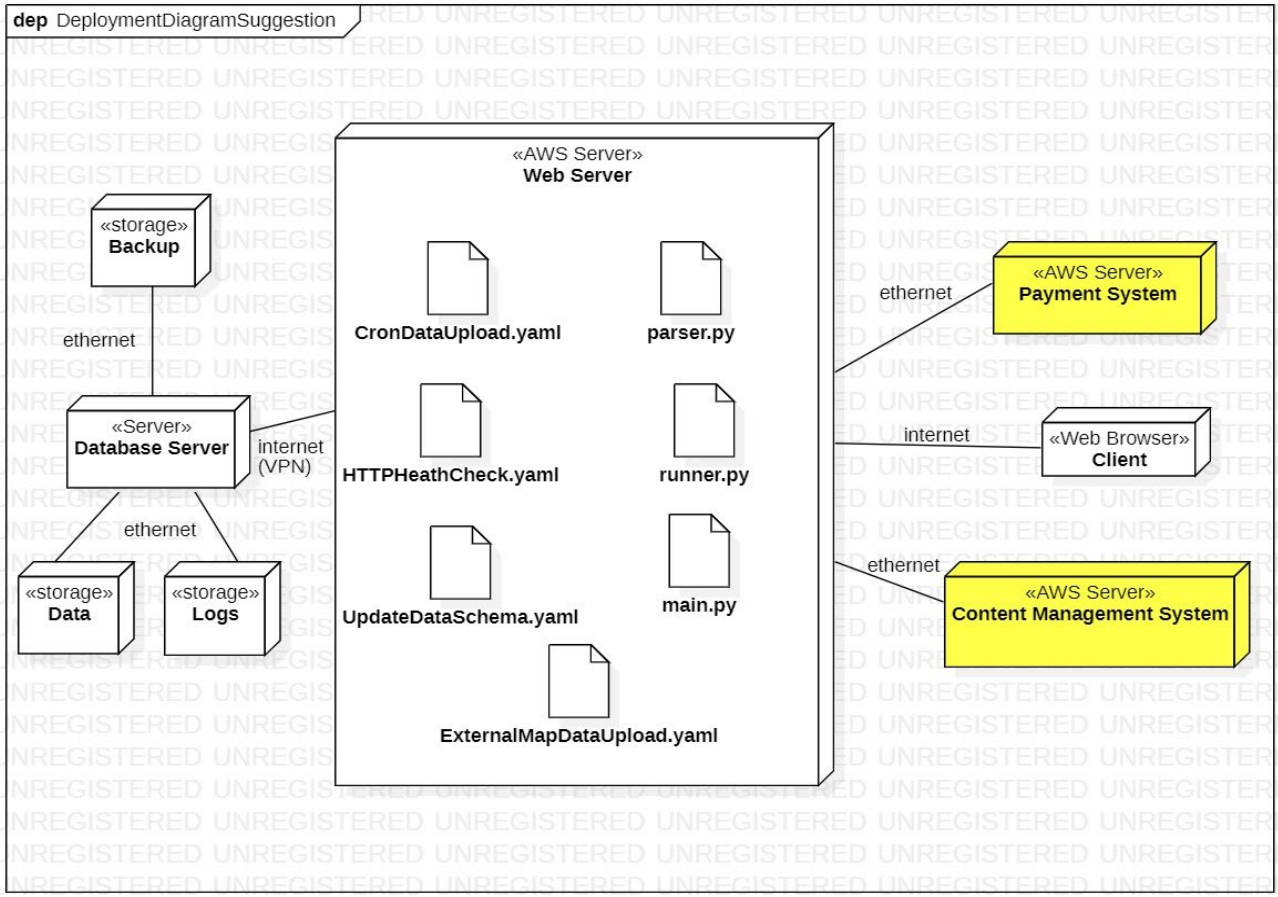

Figure 19: Deployment Diagram

- To expose the web application to external users, a Kubernetes Service may be used. It provides a table network endpoint that allows clients to access the web application. It acts as load balancer, distributing incoming traffic across the pods running the web application and ensuring high availability.

- Payment System may be deployed on the AWS cloud and it may involve integrating with a third-party payment gateway. Ethernet connection represents the physical network used for communication between these two servers.
- Content Management System may be also deployed on the cloud on a separate server. This creates a clear isolation between its and the main system.

5.5 Design Rationale

- **Context View:** The design may develop and execute integration testing strategies to validate the interactions between the main system and external entities. The design may implement appropriate security measures for integrating with external systems.
- **Functional View:** The design rationale ensures that the boundaries between components are well-defined and aligned with the responsibilities and functionalities of each component. It avoids components that are too larges or too small, and strive for a balance that promotes modularity and maintainability. It uses descriptive names for interfaces to enhance understanding and communication among team members.
- **Information View:** The design rationale of this view may extend the backup and recovery strategy to include disaster recovery and high availability considerations. It may use indexes to optimize query performance. The design may use encryption mechanisms to protect sensitive data (e.g., user data as well as payment information).
- **Deployment Diagram:** The design rationale should consider implementing CI/CD practices to automate the build, test, and deployment process. This helps ensure the continuous delivery of updates, bug fixes, and enhancements to the website while maintaining its stability and reliability.