SOFTWARE ARCHITECTURE DESCRIPTION

afetbilgi.com

GROUP 62

Denizcan Yılmaz	2444172
Ozan Cinci	2448223

Table of Contents

List of Figures

List of Tables

1. Introduction

This document is the Software Architecture Description (SAD) document for the project “afetbilgi.com”, which is developed by ODTU students. The document includes architectural views of the afetbilgi.com, including information, deployment, context, and functional views of the system.

1.1. Purpose and objectives of afetbilgi.com

"afetbilgi.com" is a website created by ODTÜ students and graduates with the goal of delivering accurate and verified information to aid disaster victims and those who want to assist in dealing with the Pazarcık Earthquake that occurred on February 6, 2023. It operates by utilizing a database that is entirely manually created by volunteers, and its aim is to provide prompt and useful information to people. Creators urge anyone who contacts them to provide only verified information to prevent the spread of misinformation and ensure that only accurate information is shared.

1.2. Scope

- System will provide its users a user-friendly interface which is easy to navigate through different categories of what they are searching for
- System will provide its users verified and up-to-date information which is contributed by volunteers
- System will let volunteers to contribute verified information for data set
- System will provide its users to search functionality to quickly find information
- System will contain contact forms for emergency requests or general inquiries
- System will be mobile responsive to provide access from mobile devices
- System will have accessibility features for people with disabilities, which they can use

1.3. Stakeholders and their concerns

There are four main stakeholders of the system: the End users, Admins, and Data Collectors & Validators.

End Users are the people who want to access websites to get help and information about where and how to find help. Since the website is served in Turkish, Kurdi, and English, basic understanding of any of these three languages and ability to interact with the website by clicking buttons to redirect is enough for End Users. Their concern is usability of the system in which they can easily access the validated data even if they have no technological literacy.

Admins are the one who regulates the website. They are in charge of maintaining the website, adding new features, and inspecting errors that may be caused. Basically solving technical issues are the main responsibility of the admins. Having high computer skills, knowing the internal mechanism of services used by afetbilgi.com, and producing the appropriate solutions to improve the system are essential for Admins. Their concern is regulation and maintaining the system with support of validated data.

Data Collectors & Validators are the ones who want to improve the website by somehow collecting the verified data and validating them. Since the website is served in 3 languages, basic understanding of these three languages, and basic knowledge of excel are essential for this role since data is collected in the form of excel. Their concern is providing correct and up-to-date data to the system.

2. References

This document is written with respect to IEEE 42010 standard, using the source below:
[1] 42010-2011 - ISO/IEC/IEEE International Standard - Systems and software engineering – Architecture Description.

3. Glossary

Term	**Definition**
UI	User Interface
API	Application programming interface
Database	S3 object storage system
HTTP/S	Hypertext transfer protocol
VPC	Virtual Private Cloud
JSON	Javascript object notation
AWS	Amazon web services

Table 1: Glossary

4. Architectural Views

4.1. Context View

4.1.1. Stakeholders’ uses of this view

There are four main stakeholders of the system: the End users, Admins, and Data Collectors & Validators.

The End users use the system to get validated data when needed. Data collectors & validators use the system to contribute to the system in order to help end users. Admins use the system to regulate the spread of valid data to end users, which are help seekers.

4.1.2. Context Diagram

afetbilgi.com is not a part of a large system, but it utilizes some services of Amazon Web Services such as S3 bucket, CloudWatch, Athena, VPC. It does not have a database; rather it uses S3 which is an object storage system as a data warehouse. Volunteers shall contribute to it via sharing verified information about categories like general need, important resources, health services etc. Users shall view the website in four languages such as Turkish, English, Kurdish and Arabic.

The interface that users benefit from surfing the website is powered by ReactJS. All users shall download the related data with filtering as pdf into their devices. The system uses static websites to reduce the latency as much as possible in case of emergency and unstable internet connection access.

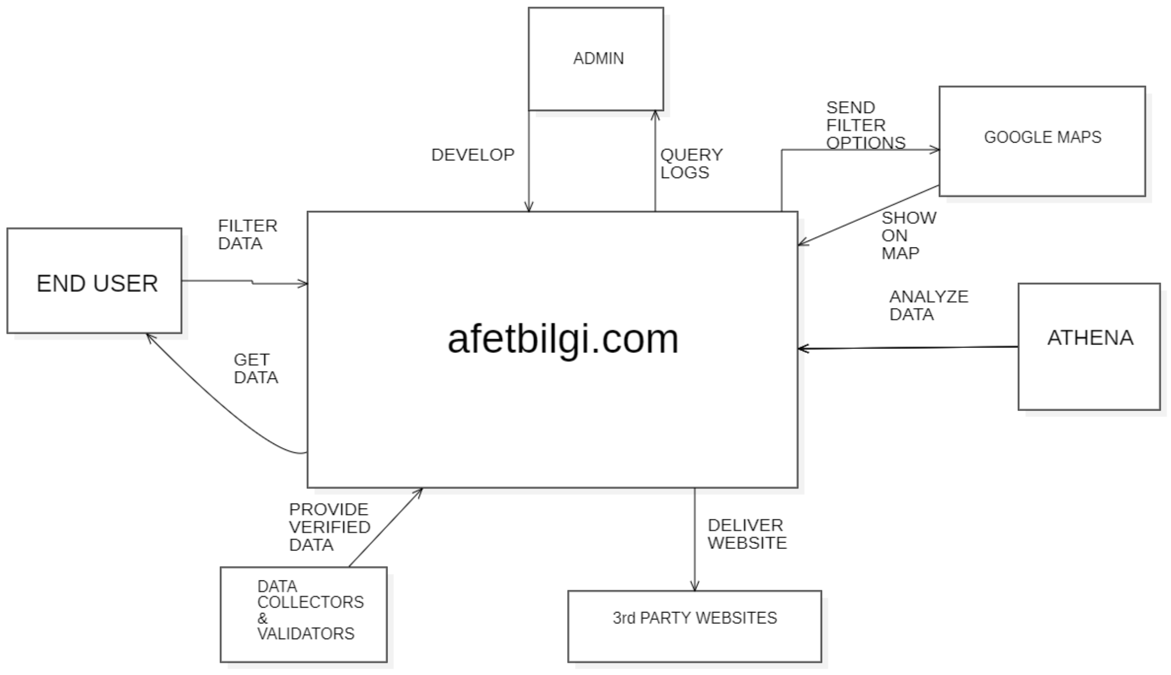

Figure 1: Context Diagram

4.1.3. External Interface

In this section, the external interfaces of afetbilgi.com are depicted with its operations, relations, and attributes.

It can be observed from the class diagram, afetbilgi.com has multiple external interfaces. Data Collectors & Validators, and Admin can be generalized as End Users. Moreover; there are Github Server, Athena, Google Maps.

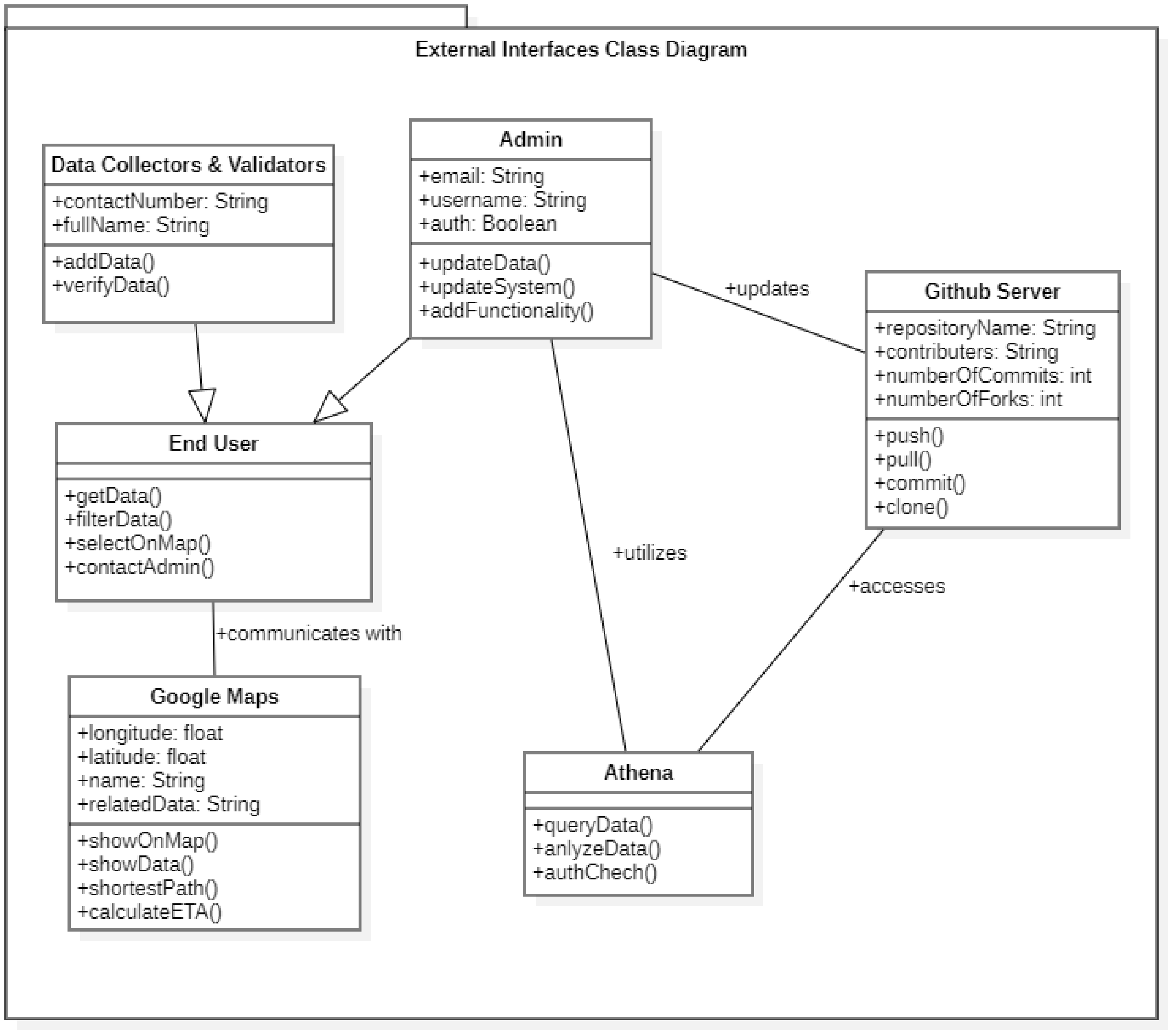

Figure 2: External Interface Diagram

Operation	**Description**
addData	Add new data into data warehouse
verifyData	Verify data as valid
updateData	Update the database using new data
updateSystem	Update the system
addFunctionality	Add new features into the system
getData	Get desired data
filterData	Filter desired data using parameters
selectOnMap	Select a location on map
contactAdmin	Contact admin
push	Update system code with changes
pull	Get updated system code
commit	Save newly added code
clone	Download system code into local computer
queryData	Run queries on data
analyzeData	Analyze data using parameters
authCheck	Check if authentication is provided
showOnMap	Show a desired location on map
showData	Show details of a location
shortestPath	Calculate how to arrive using shortest path
calculateETA	Calculate estimated arrival time

Table 2: External Interface Operations

4.1.4. Interaction scenarios

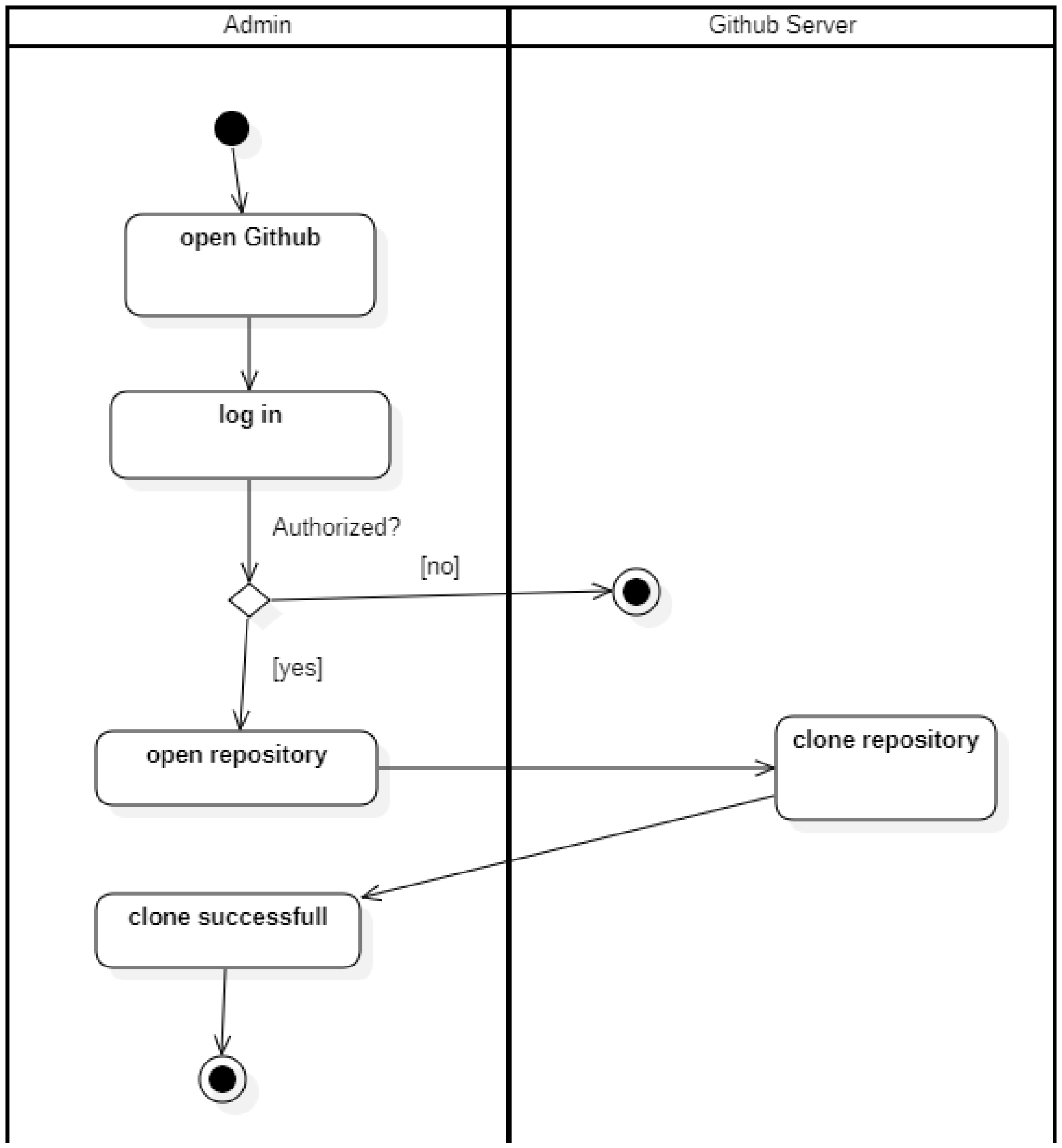

Figure 3: Interaction Diagram for github clone

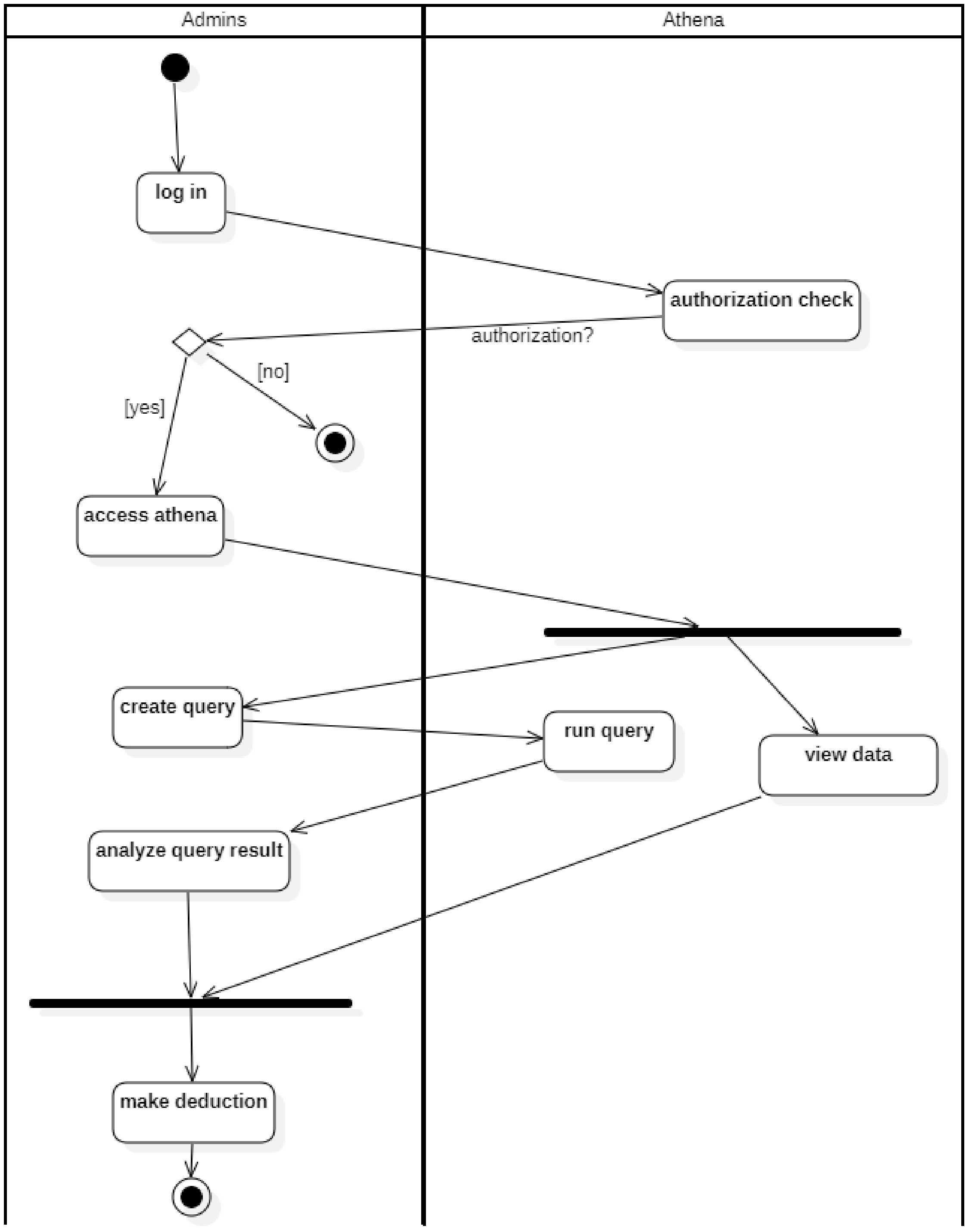

Figure 4: Interaction Diagram for analyzing data

4.2. Functional View

4.2.1. Stakeholders’ uses of this view

In this viewpoint, diverse components of the system, internal interfaces of the system and their various interactions and responsibilities are shown. Functional view is vital for stakeholders of afetbilgi.com considering that it elaborates on the idea of the functionalities and properties of the system. Moreover; it correlates with other viewpoints.

4.2.2. Component Diagram

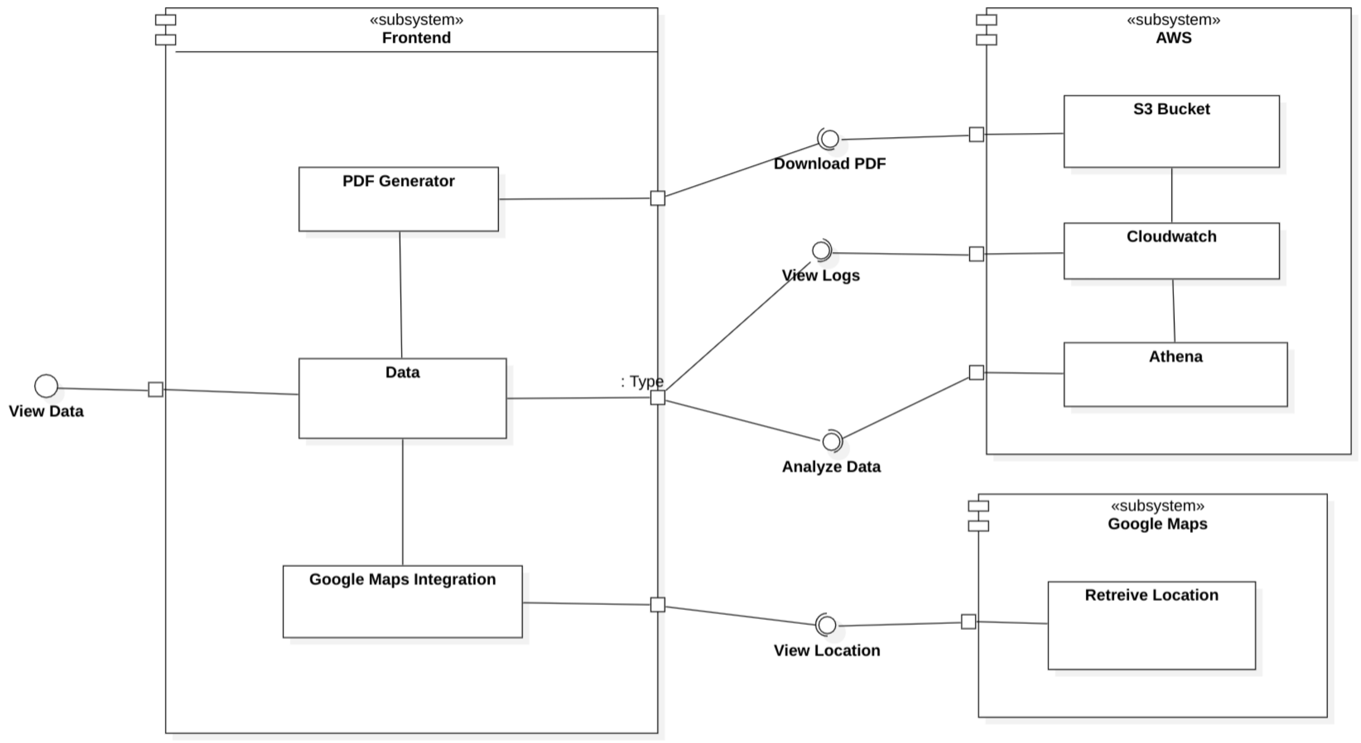

Figure 5: Component Diagram

The system is divided into 3 main subsystems: Frontend, AWS and Google Maps.

- Google Maps is an external component, which displays the location sent in latitude and longitudes on Google Maps. The integration is provided via the frontend and the Google Maps API. When a request is made, the given location is opened on Google Maps in a new tab.

- The Frontend subsystem is divided into 3 components: PDF Generator, Data and Google Maps Integration.
    - PDF Generator works as a script on the frontend and sends a request of PDF generation with the location data as its payload. PDF Generator interacts with the S3 Bucket. It uses the data kept in S3 Bucket while generating the PDF file. The generated file then served to the frontend to be downloaded by the end-user.
    - Data is the main part of the system, which contains all the valuable information on the website. It retrieves and sends data from/to the S3 Bucket in the AWS Subsystem.
    - Google Maps integration also works as a small script which sends the location information to Google Maps, to be displayed on the map.
- AWS Subsystem contains all the AWS services used on the website, which are S3 Bucket, Cloudwatch and Athena:
    - S3 Bucket is responsible for containing all the data related to the website and serves as a de-facto database for the system.
    - Cloudwatch is responsible for viewing logs and metrics related to the system.
    - Athena is responsible for generating reports related to the website, utilizing the data kept in S3.

4.2.3. Internal Interfaces

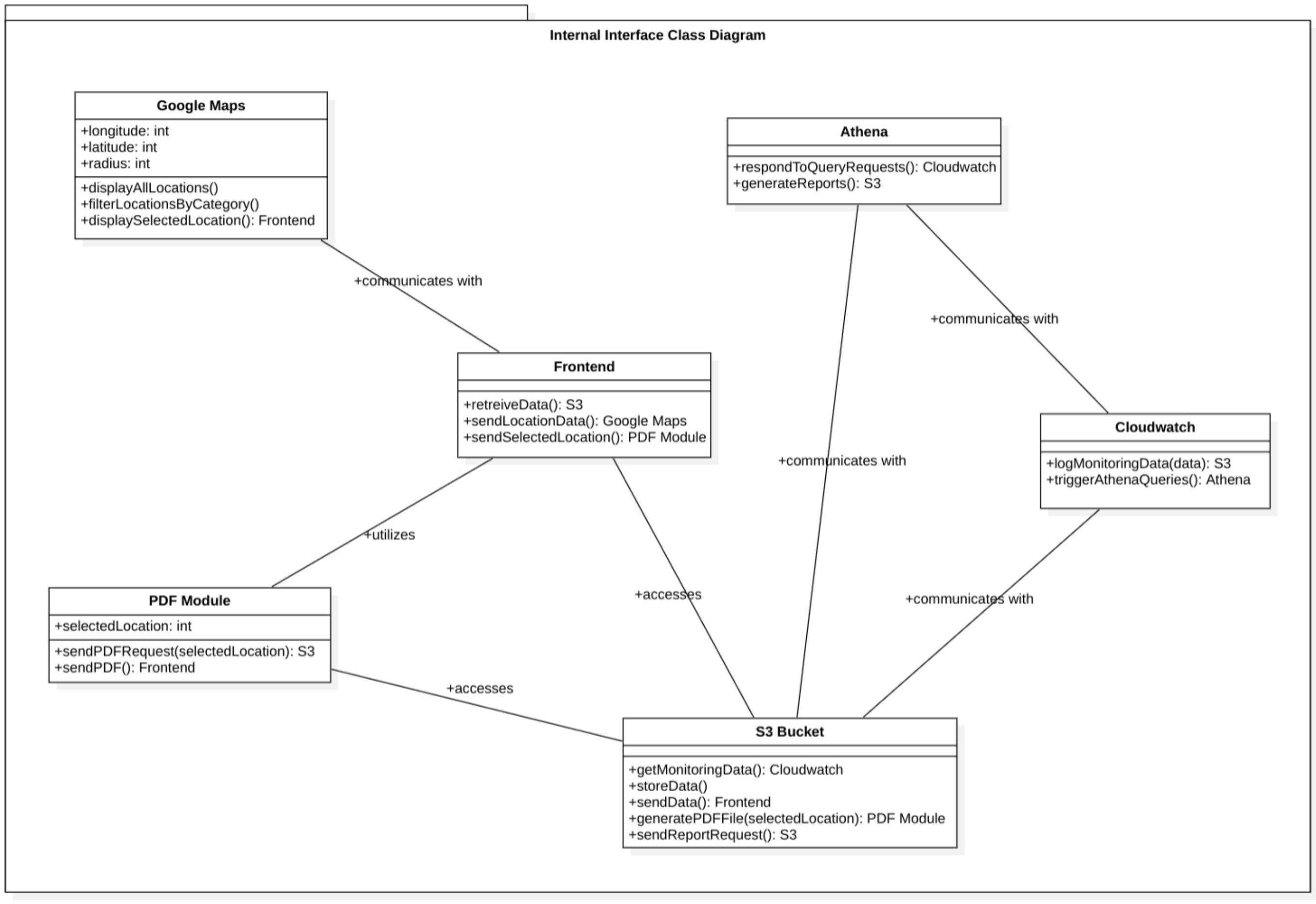
Internal Interface Class Diagram
Google Maps
+longitude: int
+latitude: int
+radius: int
+displayAllLocations()
+filterLocationsByCategory()
+displaySelectedLocation(): Frontend
Athena
+respondToQueryRequests(): Cloudwatch
+generateReports(): S3
Frontend
+retreiveData(): S3
+sendLocationData(): Google Maps
+sendSelectedLocation(): PDF Module
Cloudwatch
+logMonitoringData(data): S3
+triggerAthenaQueries(): Athena
PDF Module
+selectedLocation: int
+sendPDFRequest(selectedLocation): S3
+sendPDF(): Frontend
S3 Bucket
+getMonitoringData(): Cloudwatch
+storeData()
+sendData(): Frontend
+generatePDFFile(selectedLocation): PDF Module
+sendReportRequest(): S3
+communicates with
+communicates with
+communicates with
+communicates with
+utilizes
+accesses
+accesses

Figure 6: Internal Interface Diagram

As seen from the diagram, afetbilgi.com has several internal interfaces such as Frontend, Google Maps, Athena, Cloudwatch, S3 Bucket and PDF Module.

displayAllLocations	Display all locations registered on the given session.
filterLocationsByCategory	Filter locations based on their category: Accommodation, Food, Hospital, Pharmacy etc.
displaySelectedLocation	Display the selected location on the map.
respondToQueryRequests	Respond to query requests generated by Cloudwatch.

generateReports	Generate reports for data analysis.
retrieveData	Retrieve data from the S3 bucket.
sendLocationData	Send location data to Google Maps.
sendSelectedLocation	Send selected locations to Google Maps.
sendPDFRequest	Send PDF request to the PDF Module.
sendPDF	Send the generated PDF to Frontend.
logMonitoringData	Send monitoring logs to S3 bucket.
triggerAthenaQueries	Trigger a query from Athena.
getMonitoringData	Get the monitoring data from S3 bucket.
storeData	Store data in the S3 bucket.
sendData	Send data to frontend.
generatePDFFile	Generate PDF file for the given specifications and send back to frontend.
sendReportRequest	Send a report request to Athena for analysis.

Table 3: Internal Interface Operations

4.2.4. Interaction Patterns

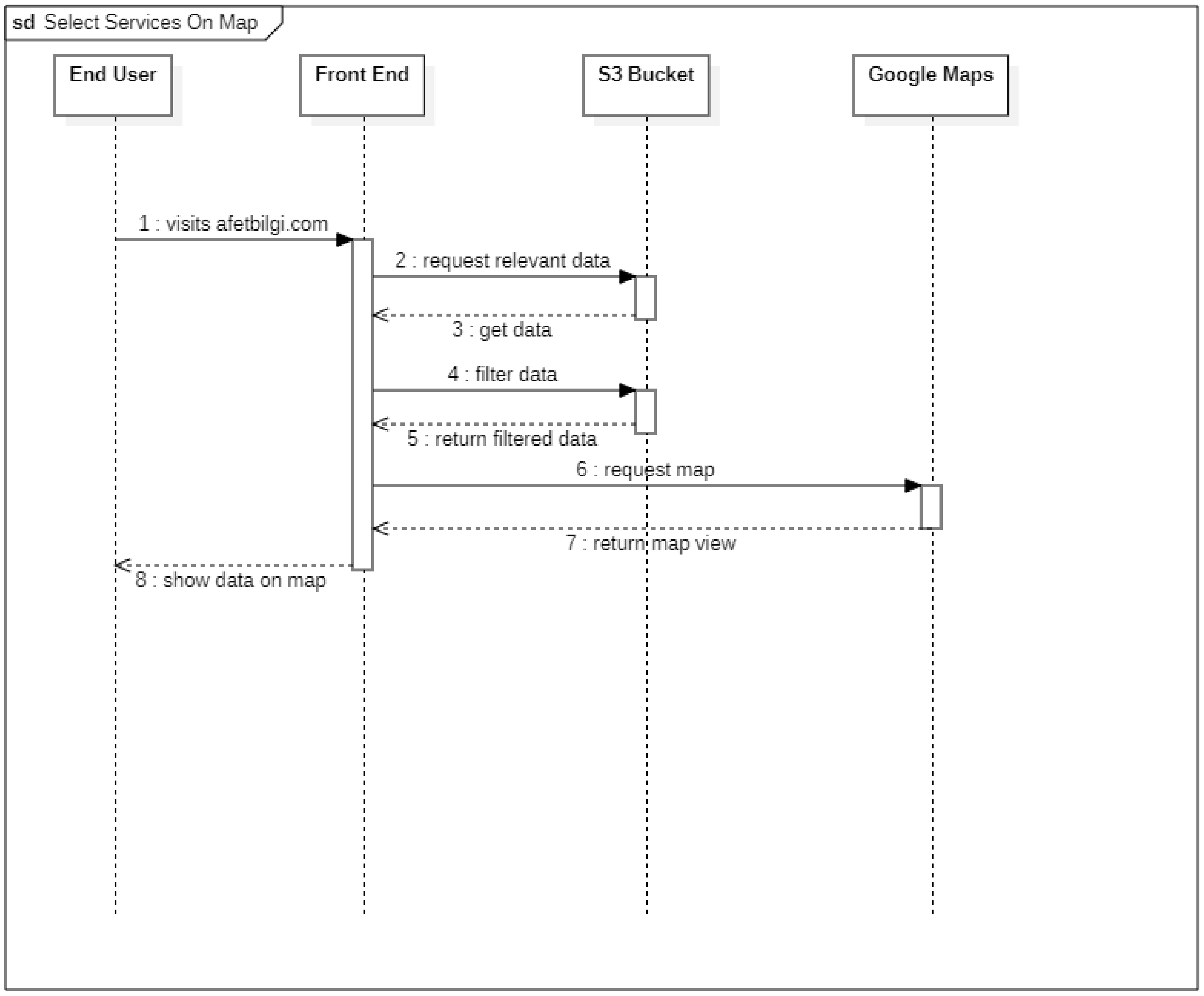

Figure 7: Sequence Diagram for selecting services on the map

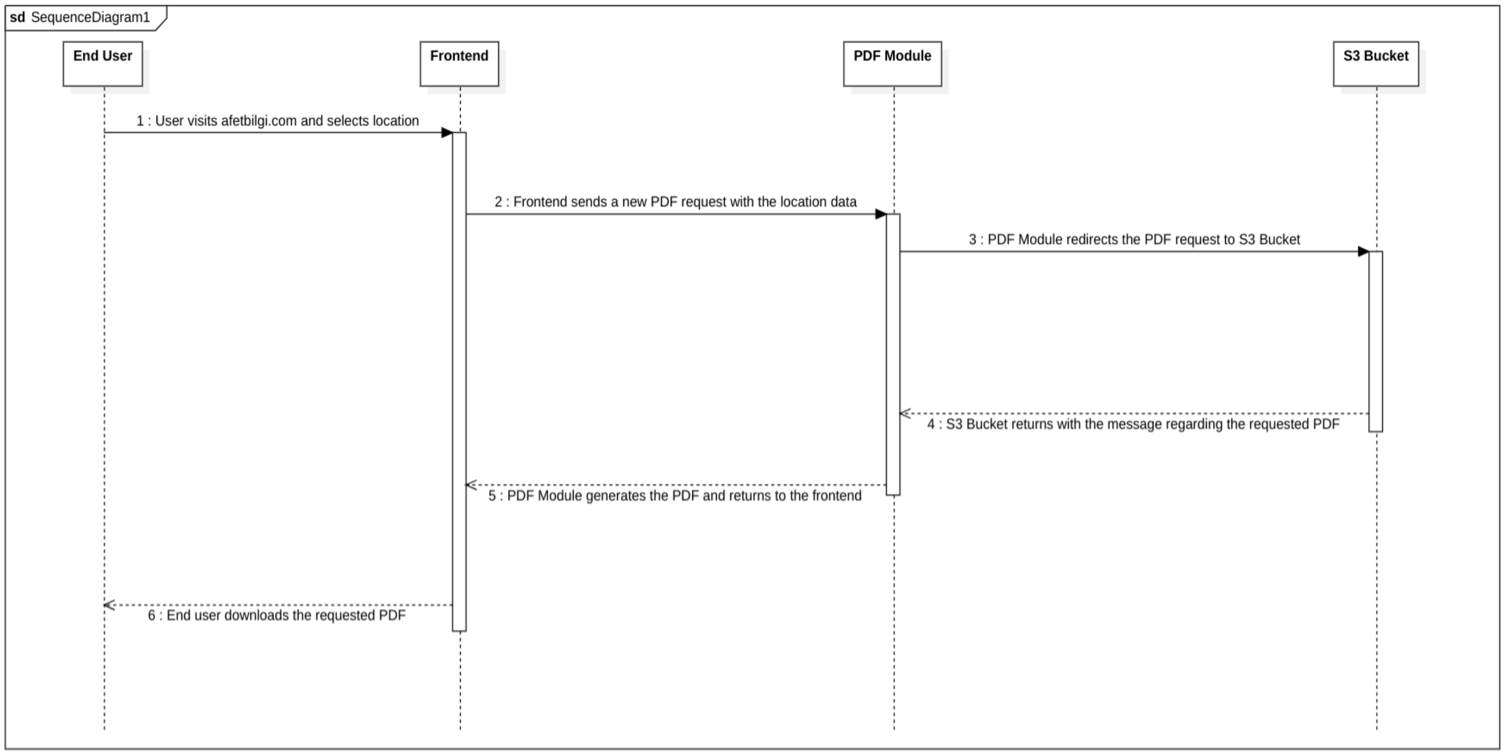
sd SequenceDiagram1
End User
Frontend
PDF Module
S3 Bucket
1 : User visits afetbilgi.com and selects location
2 : Frontend sends a new PDF request with the location data
3 : PDF Module redirects the PDF request to S3 Bucket
4 : S3 Bucket returns with the message regarding the requested PDF
5 : PDF Module generates the PDF and returns to the frontend
6 : End user downloads the requested PDF

Figure 8: Sequence Diagram for downloading PDFs

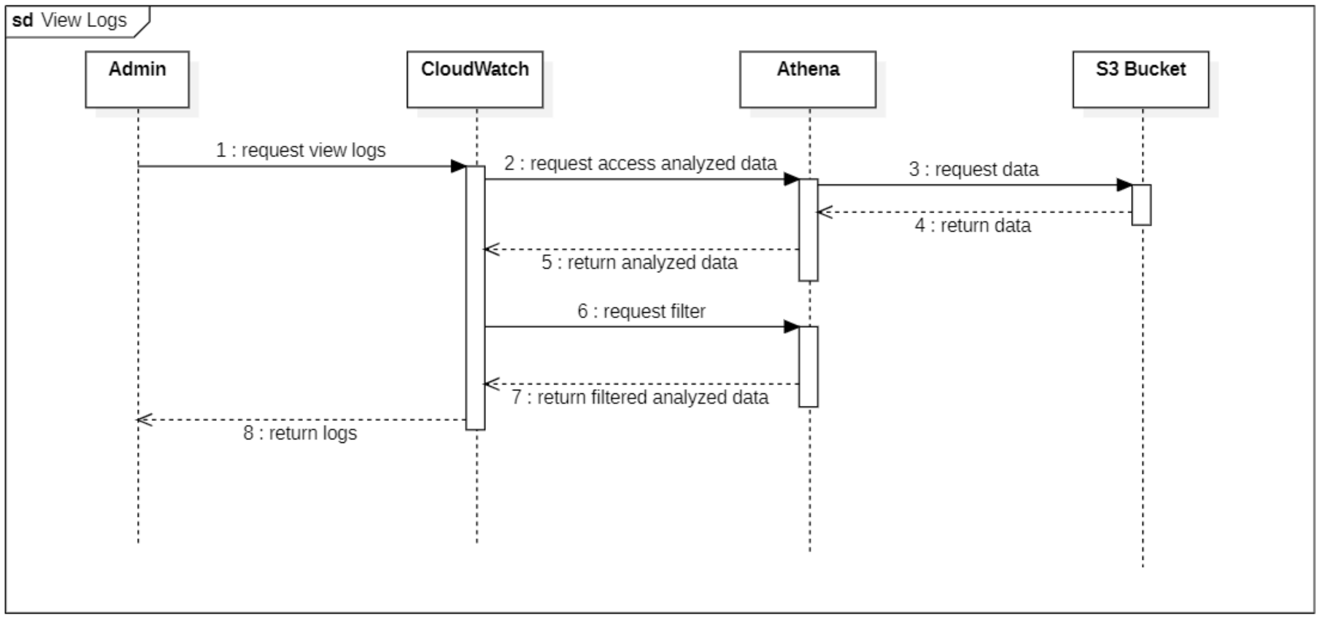

Figure 9: Sequence Diagram for reporting help request

4.3. Information View

4.3.1. Stakeholders’ uses of this view

This perspective offers an overview of essential elements in the database and main memory, along with their connections. It provides a broad understanding of the data flows, memory objects used, and data operations performed within the system. Consequently, users gain familiarity with the storage, management, modification, and distribution of data, which facilitates their utilization of different data objects and manipulation techniques.

4.3.2. Database Class Diagram

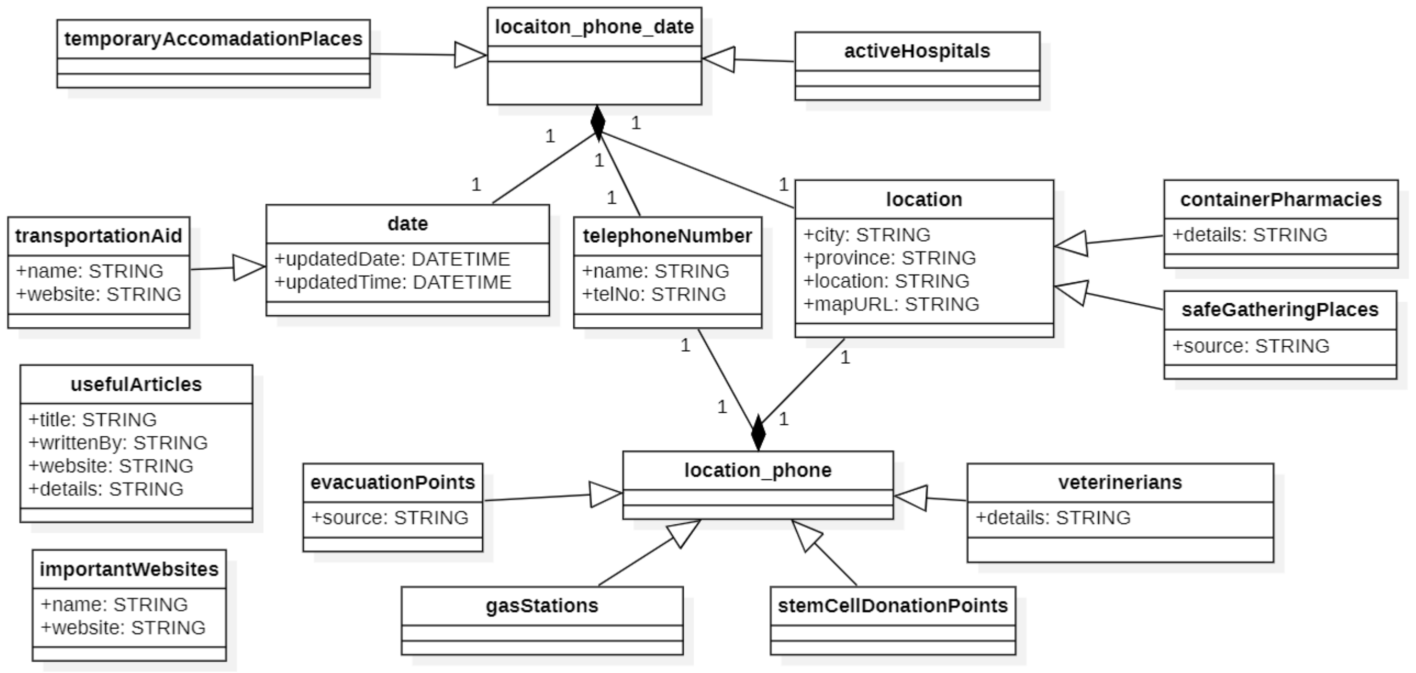

Figure 10: Database Class Diagram

There are 3 main classes which are “date”, “telephoneNumber”, and “location”. There are also their composite combinations “location_phone_date” and “location_phone”. As their names suggest, they serve as templates for various classes. Various classes inherit from them utilizing commonly used attributes while adding their new attributes. Other than that, there are few classes that do not inherit and exist on their own. Since all the attribute names are clear, it is unnecessary to explain them one by one.

4.3.3. Operations on Data

createContainerPharmacies	Create: containerPharmacies Read: location Update: - Delete: -
createVeterinerians	Create: veterinarians Read: location_phone Update: - Delete: -
createActiveHospitals	Create: activeHospitals Read: location_phone_date Update: - Delete: -
createUsefulArticles	Create: useFulArticles Read: - Update: - Delete: -
createTransportationAid	Create: transportationAid Read: date Update: - Delete: -
updateContainerPharmacies	Create: containerPharmacies Read: location Update: - Delete: -
updateVeterinerians	Create: veterinarians Read: location_phone Update: - Delete: -
updateActiveHospitals	Create: activeHospitals Read: location_phone_date Update: - Delete: -
updateUsefulArticles	Create: - Read: - Update: usefulArticles Delete: -
updateTransportationAid	Create: - Read: - Update: transporationAid Delete: -

getContainerPharmacies	Create: - Read: - Update: containerPharmacies Delete: -
getVeterinerians	Create: - Read: veterinarians Update: - Delete: -
getActiveHospitals	Create: - Read: activeHospitals Update: - Delete: -
getUsefulArticles	Create: - Read: useFulArticles Update: - Delete: -
getTransportationAid	Create: - Read: transportationAid Update: - Delete: -
deleteContainerPharmacies	Create: - Read: - Update: - Delete: containerPharmacies
deleteVeterinerians	Create: - Read: - Update: - Delete: veterinarians
deleteActiveHospitals	Create: - Read: - Update: - Delete: activeHospitals
deleteUsefulArticles	Create: - Read: - Update: - Delete: usefulArticles
deleteTransportationAid	Create: - Read: - Update: - Delete:transportationAid

Table 4: Operations on Data

4.4. Deployment View

4.4.1. Stakeholders’ uses of this view

Even though this viewpoint may not be very useful for end-users, and data collectors & validators it is particularly beneficial when admins are the point of view. This view describes the environment in which the afetbilgi is deployed, also its dependencies, which gives a better understanding and general idea about run-time, third-party software packages, hardware and network necessities. Those can be classified as crucial aspects of software development, maintainability, validity, and testability.

4.4.2. Deployment Diagram

In the deployment diagram, deployment environment and dependencies are shown.

There are computer and mobile phone devices which constitute Clients, and there is a third party server called Google Maps. This is an external server that the system depends on to use utilities of visual location based functionalities. Moreover, there is the main Web Server. It lives in one of the remote machines of AWS. It is deployed on Linux AMI Virtual Machine which operates on Linux kernel. It deploys “index.html” as the main resource. This is a html file that represents all of the website. It utilizes “all.json” as a data resource for different languages versions of the website. Also there is “robots.txt” which is basically used for initializing the website. Finally, there is a Database Server in the SERVER node. It represents the database. It deploys various files as “blood.json”, “vpn.json”, and “donation.json” for data resources and also “external_combine.py”, and “integrity_validator.py” as script files to create data from resource files.

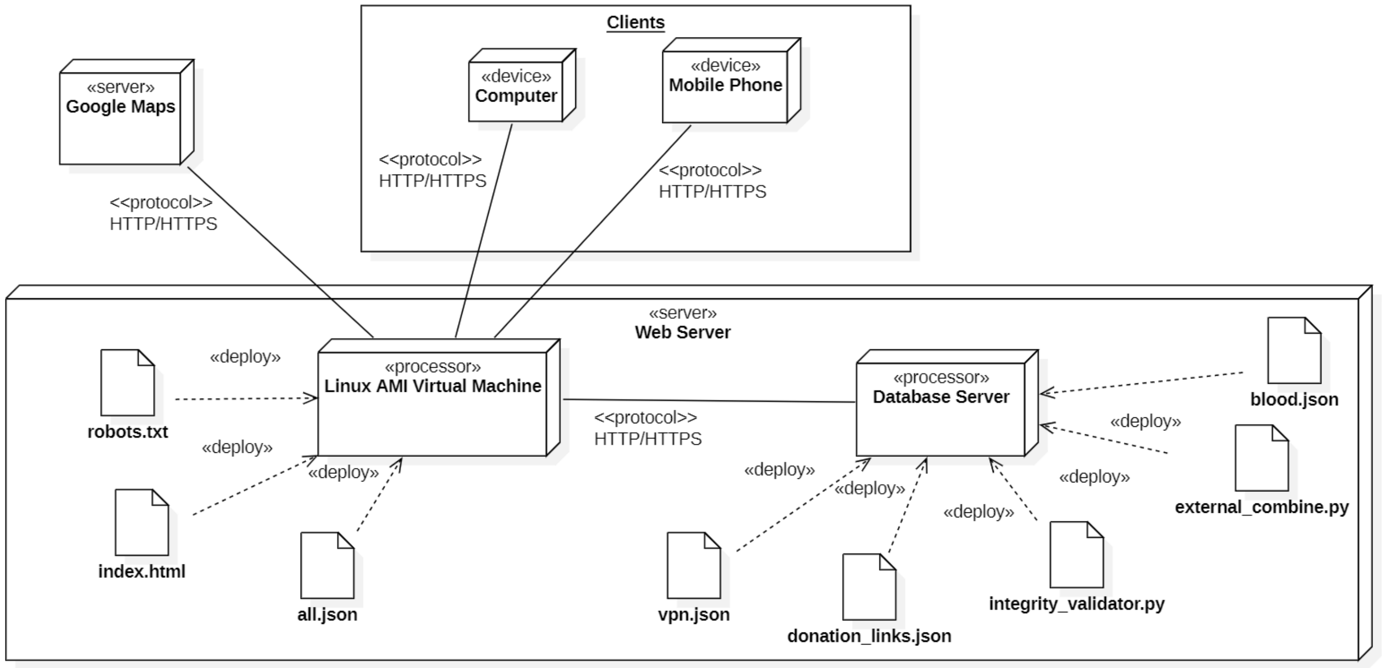

Figure 11: Deployment Diagram

4.5. Design Rationale

Context View:

In the context view, we used a context diagram and a class diagram for external interfaces. The main purpose was to describe the relationships and interactions between the system and its environment. The system context diagram depicted four stakeholders and their interactions with the "afetbilgi.com" system. The context diagram made the interactions clearer for stakeholders to understand. The external interfaces diagram showcased attributes and methods related to each interface, providing more details about the interactions. We included Github as an external interface for file sharing. Activity diagrams clarified complex scenarios that were difficult to grasp with just the class diagram. The use of context, class, and activity diagrams provided a broader and more detailed view of the system's context.

Functional View:

The main design decision in the functional view was to incorporate component and class diagrams for representing internal interfaces. These diagrams effectively showcased the runtime functional components of the system and their respective roles. The component diagram specifically helped visualize the involved elements, while the class diagrams provided a more detailed description of the interactions and responsibilities, similar to the approach used in the context view. Additionally, three sequence diagrams were developed to reinforce the objectives of the functional view and enhance comprehension for users and stakeholders. These diagrams played a vital role in clarifying the purpose of the functional view in a user-friendly manner.

Information View:

This perspective's database class diagram gave details on numerous database and main memory items. The class objects shown in the diagram mostly represented main memory objects, while the internal database of the system acted as the data storage and manipulation hub. The use of UML class diagram parts helped to easily demonstrate the links between various objects. The diagram also featured a list of possible procedures, which provided more insight into the use and manipulation of the data.

Deployment View:

The deployment view's rationale is to keep the entire software and hardware architecture simple and modular. Each component has a separate file, which must be imported before use by all other modules. Both the host computer and the operating system are virtual machines running AWS Linux AMI2. This makes the system more portable while still maintaining the operating system's ability to run any Python, Javascript, and Docker code. In addition to Python and Javascript, the system also requires to run React, which is a Javascript framework.

5. Architectural Views for Suggestions to Improve the Existing System

5.1. Context View

5.1.1. Stakeholders' uses of this view

There are four main stakeholders of the system: the End users, Admins, and Data Collectors & Validators.

The End users use the system to get validated data when needed. Data collectors & validators use the system to contribute to the system in order to help end users. Admins use the system to regulate the spread of valid data to end users, which are help seekers.

5.1.2. Context Diagram

We suggest that:

- afetbilgi.com should also be a mobile app.
- afetbilgi.com should have an account system.
- End users can register into the account system.
- There would be a concept such as Help Request.
- Users would create Help Requests.
- Admin would manage Help Requests.
- There would be an admin panel so that admin would easily contact authorities to answer help requests.

End users can download the mobile app of afetbilgi.com so that they can easily use the original version of the website instead of reading through the downloaded PDF. Thanks to mobile apps, there would be an account registration system that can be integrated with mobile apps. Users would easily register into this system by providing some information such as their phone number, name, location, blood type, relatives' name, and emergency contact number. In case of emergency, registered users can easily access a roll call thanks to this system. After an emergency, the end user checks his/her status as alive so that no resource will be wasted. Also registered users would create Help requests for nutrients, paramedical help, or crane needs to lift rubble, etc. . When a help request is created, the admin can regulate this help request based on its location and would let authorities to react to an emergency so that only an internet connection would be enough to collect proper data about needed help.

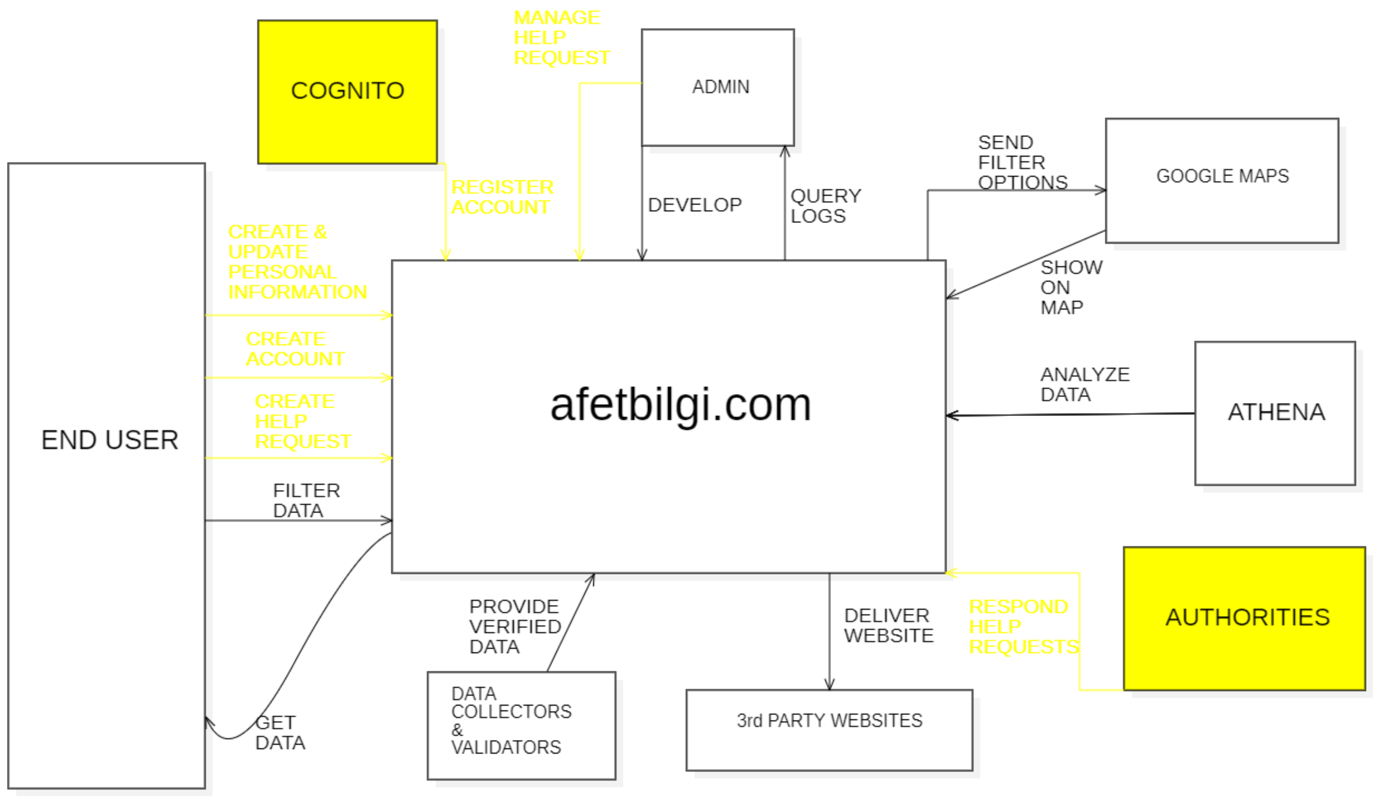

Figure 12: Context Diagram

5.1.3. External Interfaces

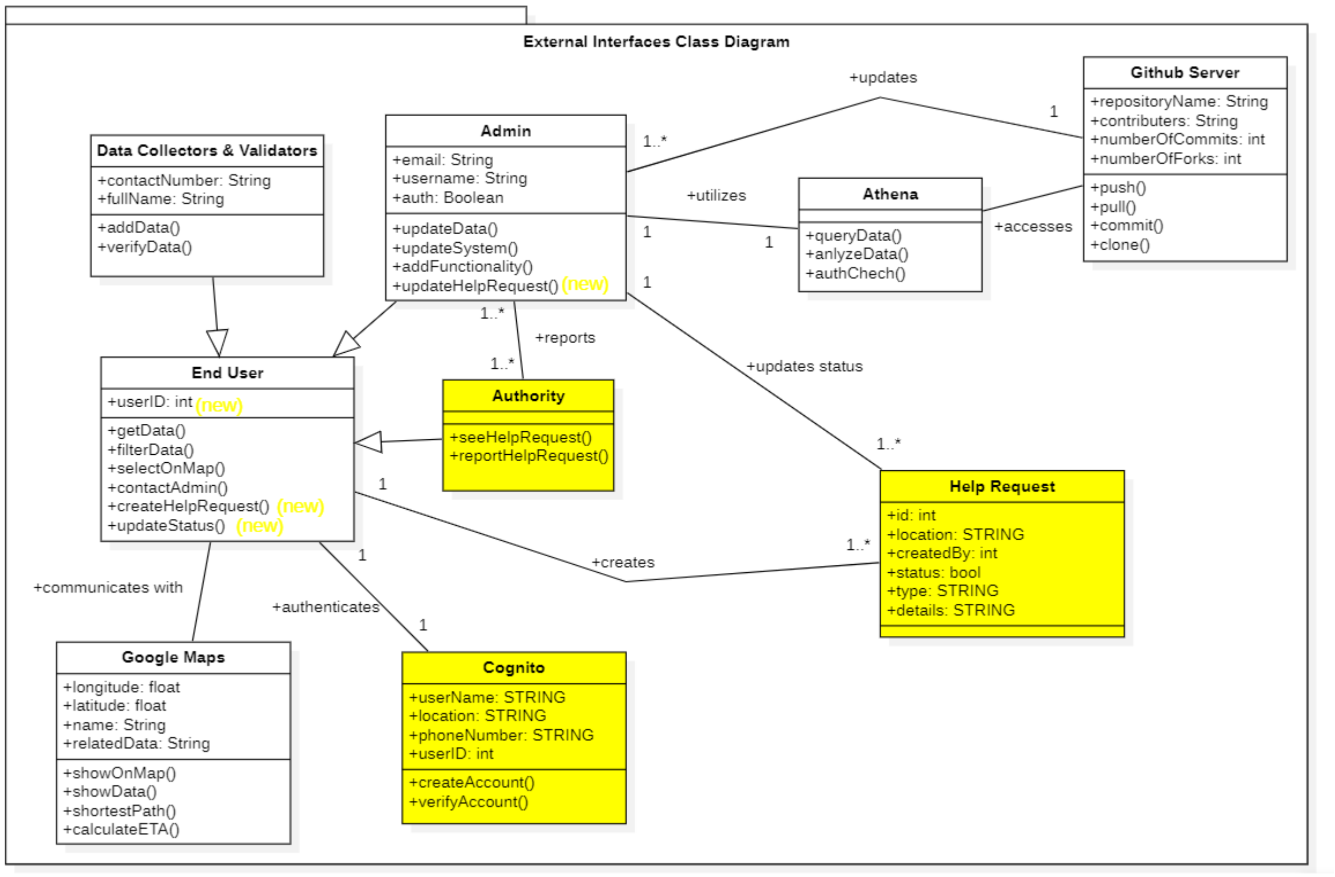

Figure 13: External Interfaces Class Diagram

Authority, Help Request, and Cognito are new external interfaces. Also, some previous external interfaces have new attributes and operations. “userID” attribute is for the 1 to 1 matching with Cognito. Operations are listed below:

Operation	**Description**
updateHelpRequest	After receiving feedback from an authority, admin can close the help request or let an authority know about the help request using this operation.
createHelpRequest	End users can create help request
updateStatus	End users can update their status as alive, trapped, or anything else after a disaster.

seeHelpRequest	See the details of the created help request.
reportHelpRequest	Report a help request to the admin as done.
createAccount	Create an account.
Verify account	Verify an account as admin or authority.

Table 5: External Interface Operations

5.1.4. Interaction scenarios

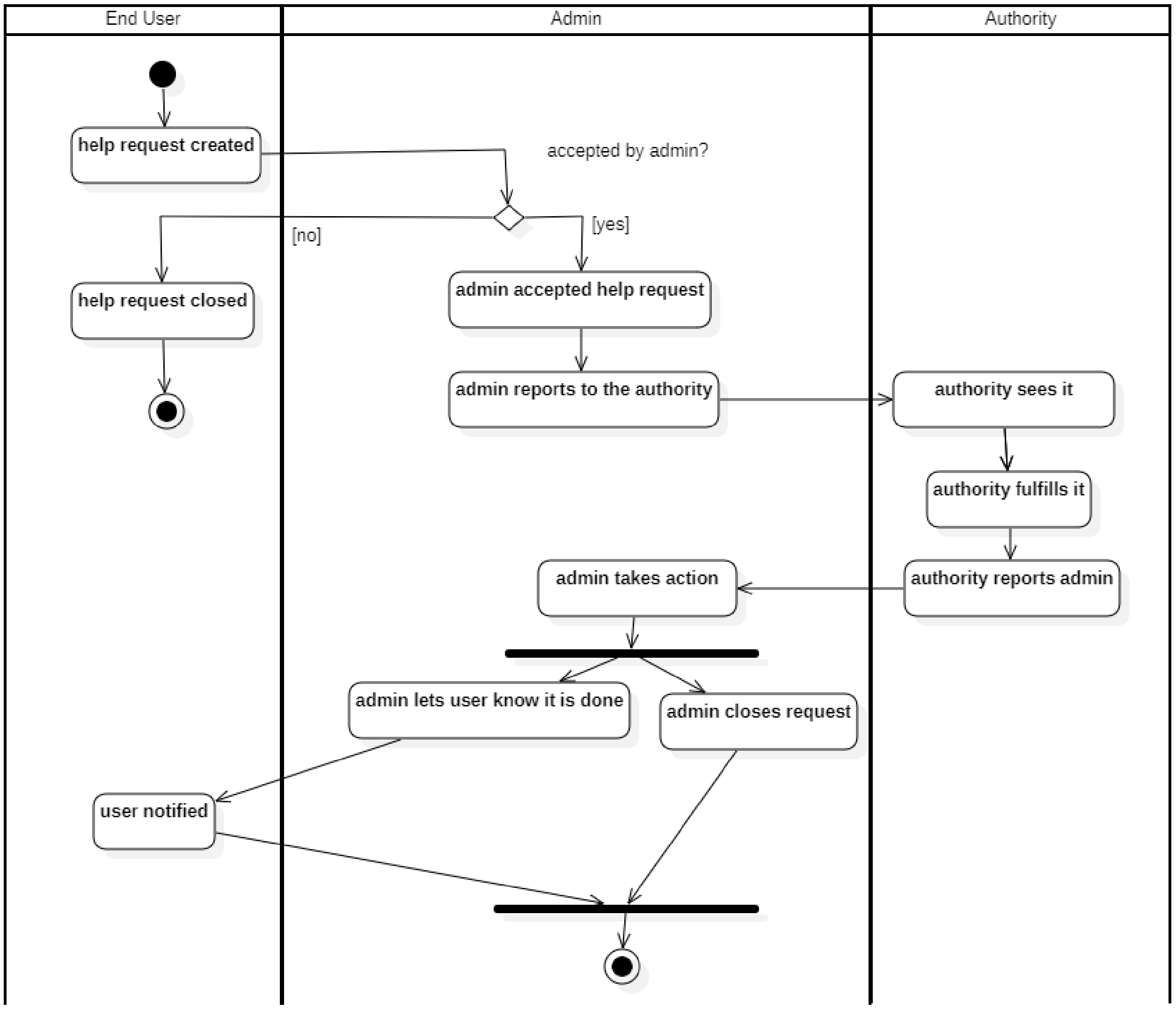

Figure 14: Interaction Diagram for updating help request

5.2. Functional View

5.2.1. Stakeholders' uses of this view

The inclusion of component and class diagrams for depicting internal interfaces is the primary design choice for the functional view. The system's runtime functional components and their separate tasks are clearly illustrated in these diagrams. The component diagrams provide the relationships between different components of the system. The class diagrams give a more in-depth description of the interactions and responsibilities, similar to the method used in the context view, while the component diagrams specifically assist in visualizing the associated pieces. One sequence diagram is created to further the functional view's goals and improve users' and stakeholders' comprehension of the interaction patterns. These illustrations significantly contribute to the user-friendly clarification of the functional view's goal.

Different uses of this view according to different stakeholders are as below:

- **Help Seeker:** People who need assistance can utilize the functional perspective to comprehend the capabilities of the system and how it can aid them in achieving their objectives. In order to better satisfy their needs, the system's design can be improved by helping users identify potential problems.
- **Help Provider:** Help providers can utilize the functional view to comprehend the capabilities of the system and how they might make use of it to aid help seekers. In order to better serve the requirements of both help seekers and help providers, this can assist providers detect possible problems and make suggestions for changes to the system's architecture.
- **Developer:** The functional perspective can be used by developers to comprehend the functional needs of the system and how they are implemented in the architecture of the system. This can assist developers in identifying future problems and making design suggestions for the system.
- **Data Collector:** Data collectors can use the functional view to comprehend the system's requirements for data collection as well as the processes involved in gathering and storing data. This can assist data gatherers in making sure that the information they are gathering is accurate and complies with system standards.
- **Data Validator:** Data validators can use the functional view to understand the data validation requirements of the system and how data is validated. This can help data validators ensure that the data being used by the system is accurate and valid.

5.2.2. Component Diagram

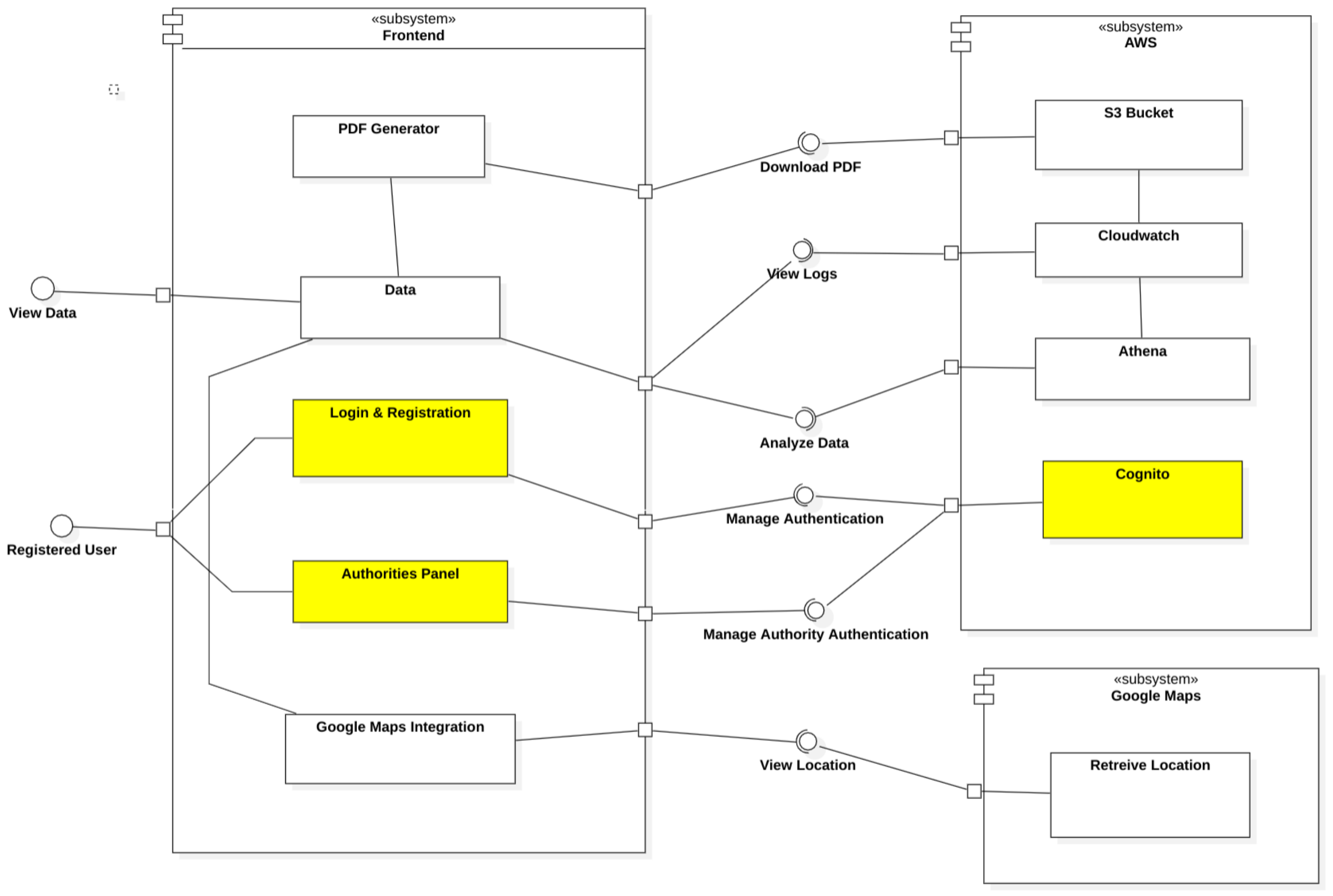

Figure 15: Component Diagram

5.2.3. Internal Interfaces

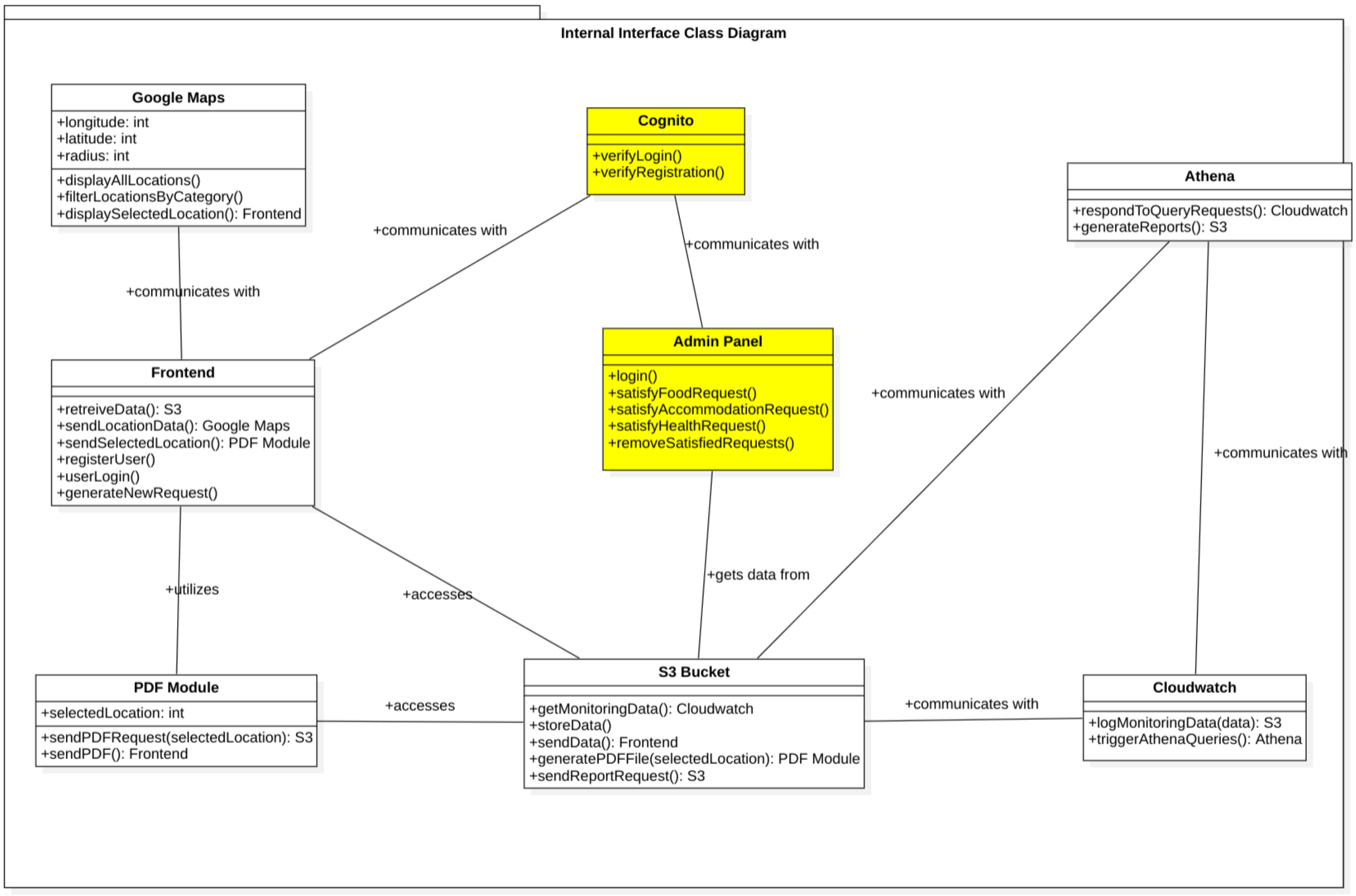

Figure 16: Internal Interface Diagram

verifyLogin	Cognito verifies the user login
verifyRegistration	Cognito verifies the user registration
login	User login for the Admin Panel
satisfyFoodRequest	Satisfy existing food request on the Admin Panel
satisfyAccommodationRequest	Satisfy existing accommodation request on the Admin Panel
satisfyHealthRequest	Satisfy existing health request on the Admin Panel

removeSatisfiedRequests	Remove satisfied requests on the Admin Panel
registerUser	Register new user on the frontend
userLogin	User login on the frontend
generateNewRequest	Generate new request for the logged in user

Table 6: Internal Interface Operations

5.2.4. Interaction Patterns

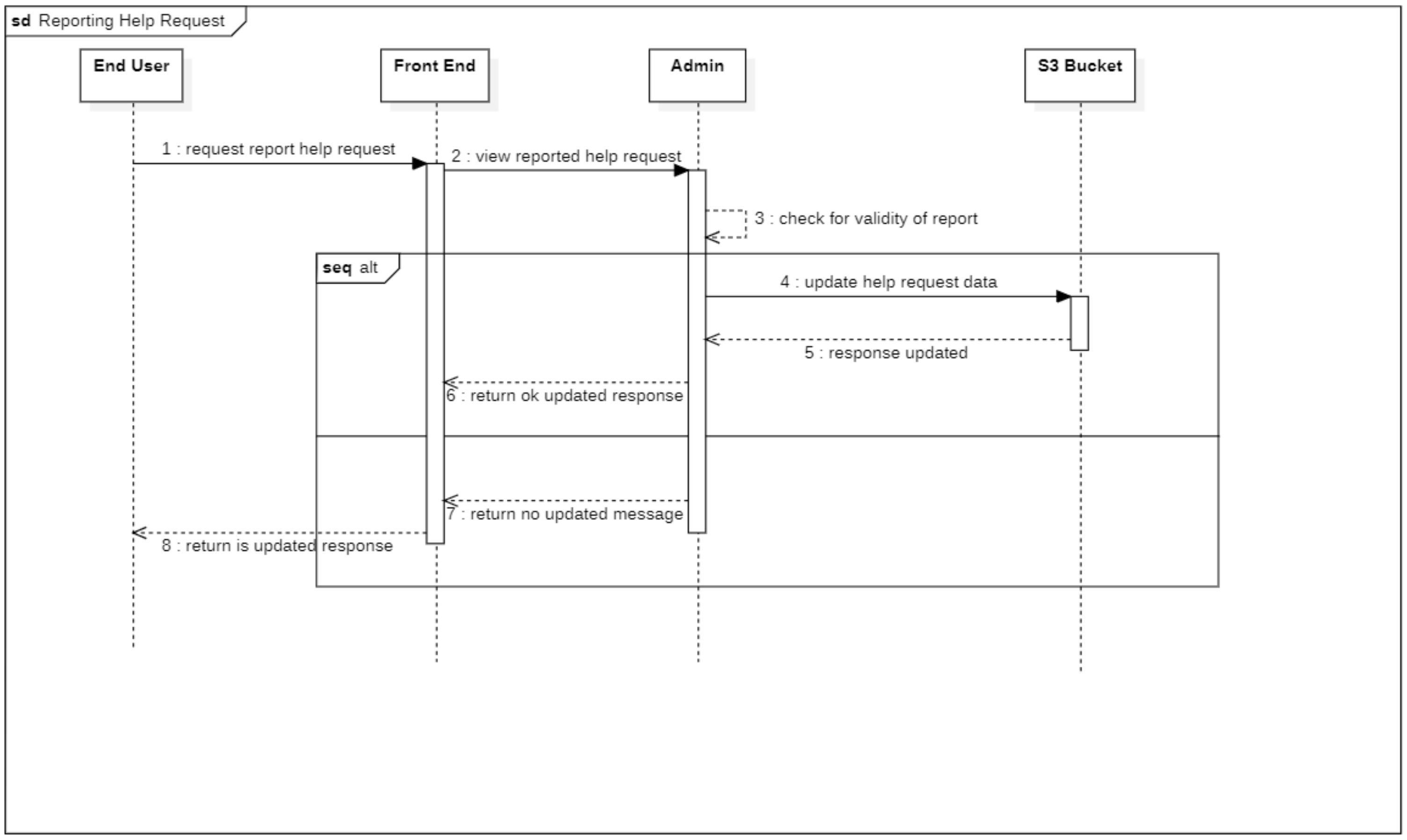
sd Reporting Help Request
End User
Front End
Admin
S3 Bucket
1 : request report help request
2 : view reported help request
3 : check for validity of report
seq alt
4 : update help request data
5 : response updated
6 : return ok updated response
7 : return no updated message
8 : return is updated response

Figure 17: Interaction Diagram for updating help request

5.3. Information View

5.3.1. Stakeholders’ uses of this view

This perspective offers an overview of essential elements in the database and main memory, along with their connections. It provides a broad understanding of the data flows, memory objects used, and data operations performed within the system. Consequently, users gain familiarity with the storage, management, modification, and distribution of data, which facilitates their utilization of different data objects and manipulation techniques.

5.3.2. Database Class Diagram

To make the document clear, the previous part is not illustrated. Only the new part is illustrated below.

There will be 3 new classes. Users, contact numbers, and requests. Requests are created by users and there will be one to many relationships between them.A user can create as many requests as s/he wants. A request consists of data that is related to creation time and type of request. User class type has data related to personal information to identify the user and contact number to contact him/her. Account type may be “admin”, “authority”, or “end user”. Request types may be “nutrition”, “heatlhEmergncy”, etc. No further details provided since the names are descriptive.

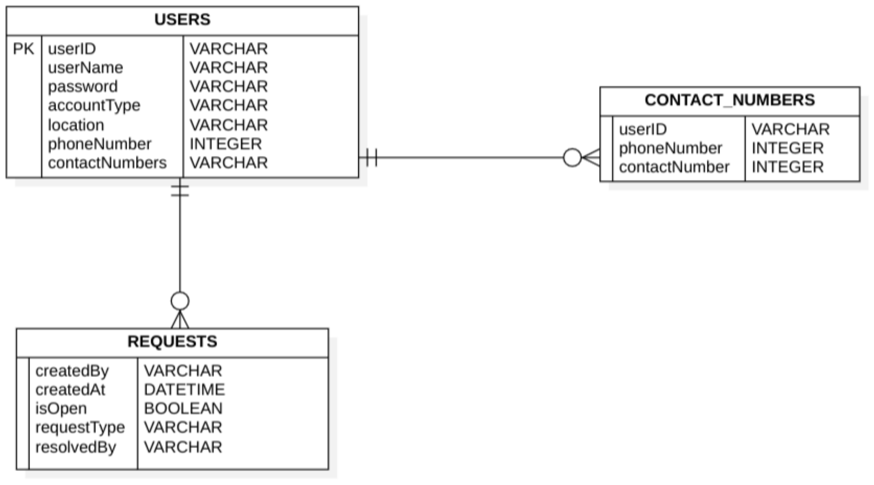

Figure 18: Database Class Diagram

5.3.3. Operations on Data

createUsers	Create: Users Read: -

	Update: - Delete: -
createRequests	Create: Requests Read: - Update: - Delete: -
createContactNumbers	Create: Contact_Numbers Read: - Update: - Delete: -
updateUsers	Create: - Read: - Update: Users Delete: -
updateRequests	Create: - Read: - Update: Requests Delete: -
updateContactNumbers	Create: - Read: - Update: Contact_Numbers Delete: -
getUsers	Create: - Read: Users Update: - Delete: -
getRequests	Create: - Read: Requests Update: - Delete: -
getContactNumbers	Create: - Read: Contact_Numbers Update: - Delete: -
deleteUsers	Create: - Read: - Update: - Delete: Users
deleteRequests	Create: - Read: - Update: -

	Delete: Requests
deleteContactNumbers	Create: - Read: - Update: - Delete: Contact_Numbers

Table 7: Operations on Data

5.4. Deployment View

5.4.1. Stakeholders' uses of this view

The deployment view's rationale is to keep the entire software and hardware architecture simple and modular. Each component has a separate file, which must be imported before use by all other modules. Both the host computer and the operating system are virtual machines running AWS Linux AMI2. This makes the system more portable while still maintaining the operating system's ability to run any Python, Javascript, and Docker code. In addition to Python and Javascript, the system also requires to run React, which is a Javascript framework.

5.4.2. Deployment Diagram

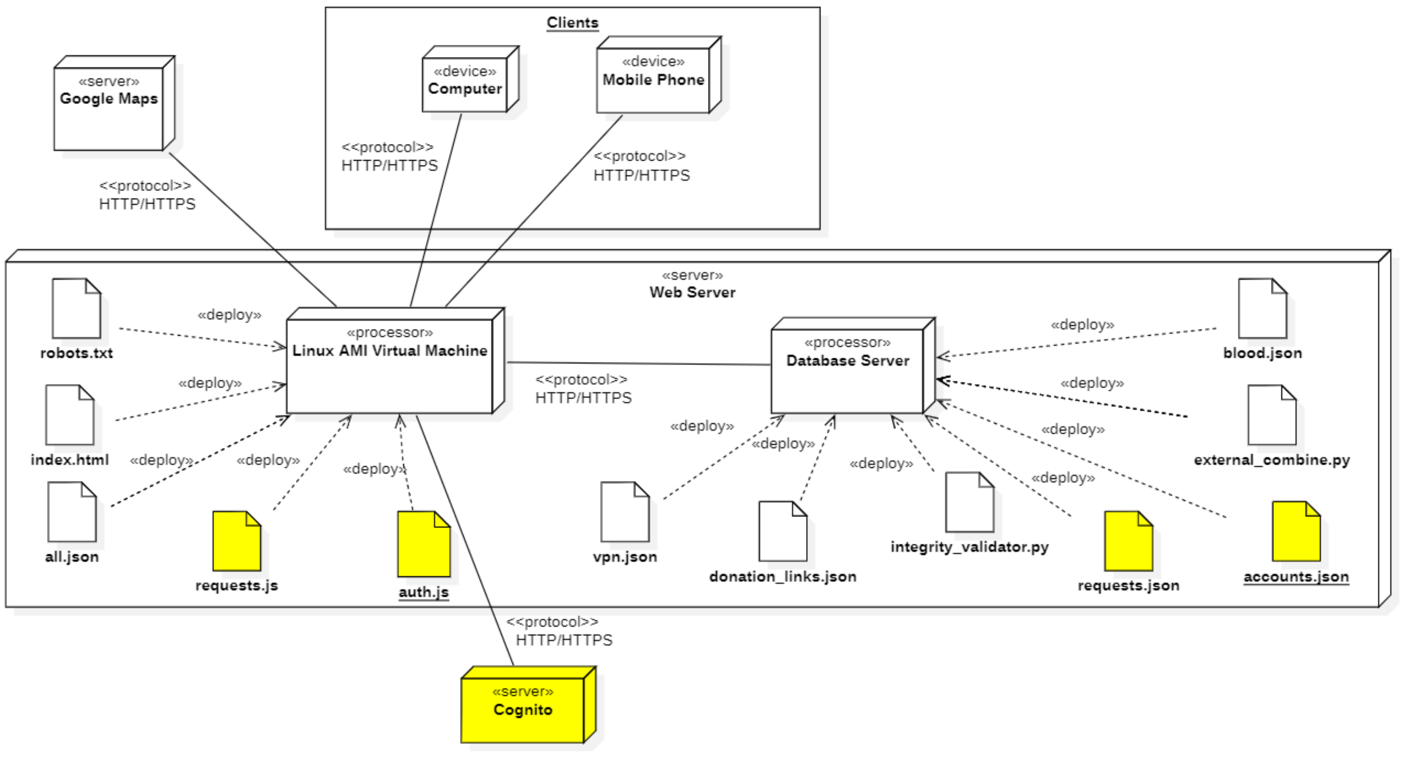

Figure 19: Deployment Diagram

The yellowed part is the suggestion part. According to our suggestion, an account system and help request system should be implemented.

To implement an account system, we utilize an external server called Cognito. It communicates with the main web server using HTTP and HTTPS protocols. To communicate with the Cognito server, our main server should also deploy a Javascript file called “auth.js”. It is a simple NodeJS file that connects Cognito, Main Web Server, and Database Server. In Database Server, we also deploy a file called “accounts.json.” This file is to prevent time waste since The Web Server mainly uses it as cache since the main account database is inside Cognito server. By utilizing “accounts.json” as cache, we also reduce the latency for lately used accounts. Moreover; admin and authority accounts will always be in the cache so that they will never experience any latency or error due to the Cognito system.

To implement a help request system, we deploy a Javascript file called “requests.js”. It is to save the created or updated request into the database. It will cooperate with Cognito to authorize admins, authorities, and end users. Also, a file called “requests.json” is deployed into the database to create, read, update, and delete requests systematically.

5.5. Design Rationale

- **Context View:**

    In the context view, we used a context diagram and a class diagram for external interfaces. The main purpose was to describe the relationships and interactions between the system and its environment. The system context diagram depicted four stakeholders and their interactions with the "afetbilgi.com" system. The context diagram made the interactions clearer for stakeholders to understand. The external interfaces diagram showcased attributes and methods related to each interface, providing more details about the interactions. We included Cognito, Help Request and Authority as external interfaces for our suggestions. Activity diagrams clarified complex scenarios that were difficult to grasp with just the class diagram such as report help request scenario. Thanks to adding new external interfaces, their operations to interact with system such as “seeHelpRequest”, and “reportHelpRequest” from Authority or “createAccount”, and “verifyAccount” from Cognito are also added to the system. The use of context, class, and activity diagrams provided a broader and more detailed view of the system's context.

- **Functional View:**

    The main design decision in the functional view was to incorporate component and class diagrams for representing internal interfaces. These diagrams effectively showcased the runtime functional components of the system and their respective roles. The component diagram specifically helped visualize the involved elements, while the class diagrams provided a more detailed description of the interactions and responsibilities, similar to the approach used in the context view. Additionally, one new sequence diagram was developed to reinforce the objectives of the functional view and enhance comprehension for users and stakeholders. These diagrams played a vital role in clarifying the purpose of the functional view in a user-friendly manner. It can be seen that there are 3 new components in the two separate subsystems. They are “Login & Registration”, “Authorities Panel”, and “Cognito”. They are meant to introduce account management and request management systems into afetbilgi.com. Registered users shall interact with Authorities Panel and Login & Registration depending on their user type which is admin, end user, or authority. Also, the Cognito component is there for account verification and management systems.

- **Information View:**

    This perspective's database class diagram gave details on various database and main memory items such as Users, Contact_Numbers, and Requests. Users class is added to provide account functionality for the system Contact_Numbers is added to store contact data for individual users. There is one to many connection between them since a user might have many contact information whereas a contact information should belong to only one user. Also, we added a request class to represent help requests created by users. There is one to many relationship as well since an user may create more than one request whereas a request should only belong to one user. The class objects shown in the diagram mostly represented main memory objects, while the internal database of the system acted as the data storage and manipulation hub. The operations on data featured a list of possible procedures, which provided more insight into the use and manipulation of the data. It is listed as a table and uses CRUD operations properly.

- **Deployment View:**

The deployment view's rationale is to keep the entire software and hardware architecture simple and modular. Each component has a separate file, which must be imported before use by all other modules. Both the host computer and the operating system are virtual machines running AWS Linux AMI2. This makes the system more portable while still maintaining the operating system's ability to run any Python, Javascript, and Docker code. In addition to Python and Javascript, the system also requires to run React, which is a Javascript framework. In the suggestion part we added a new external server called Cognito to introduce account functionality. It communicates with the main Web Server using internet protocols HTTP and HTTPS. Moreover, Linux AMI Virtual Machine should deploy two new files called “request.js” and “auth.js”. They are both Javascript files utilizing NodeJS to provide managing authentication and request systems. Finally, there are two newly introduced files called “request.json”, and “accounts.json” in the database server to help caching and database management purposes.