MIDDLE EAST TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DESIGN

CENG 491-492

SENIOR PROJECT

COMBINED REQUIREMENTS, DESIGN, AND TEST DOCUMENT

"AUTOMATED MONITORING SOLUTIONS"

by GROUP RAAD

MEMBERS:

- **DUYGU DOĞAN**
- **OĞUZHAN BURAK ISSI**
- **METEHAN ÖZTÜRK**
- **YİĞİT YÜKSELEN**

ASSISTANT : Hazal MOĞULTAY

SUPERVISOR: Asst. Prof. Ebru Aydın GÖL

TABLE OF CONTENTS

TABLE OF FIGURES

1. SYSTEM REQUIREMENTS

1.1. REVISED SYSTEM REQUIREMENTS (with Sequence Diagrams)

These sequence diagrams shows the interactions between the components of the system (i.e. administrator, user, administrator interface, user interface, and data system). The messages between each lifetime named with their corresponding function names in our class diagram. The interaction names of each sequence diagram are related with the use case names in our use case diagram.

The explanations of each use case diagram and class diagram (and its functionalities) can be found in their corresponding parts.

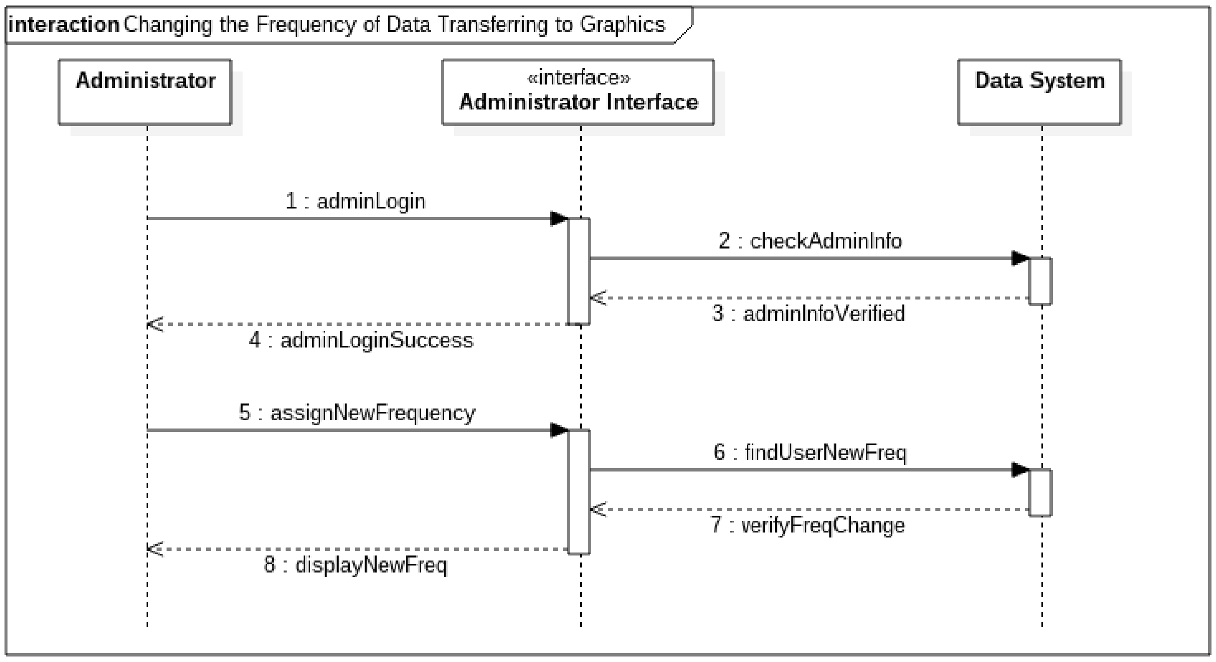


Figure 1 : Sequence Diagram of “Changing the Frequency of Data Transferring to Graphics”

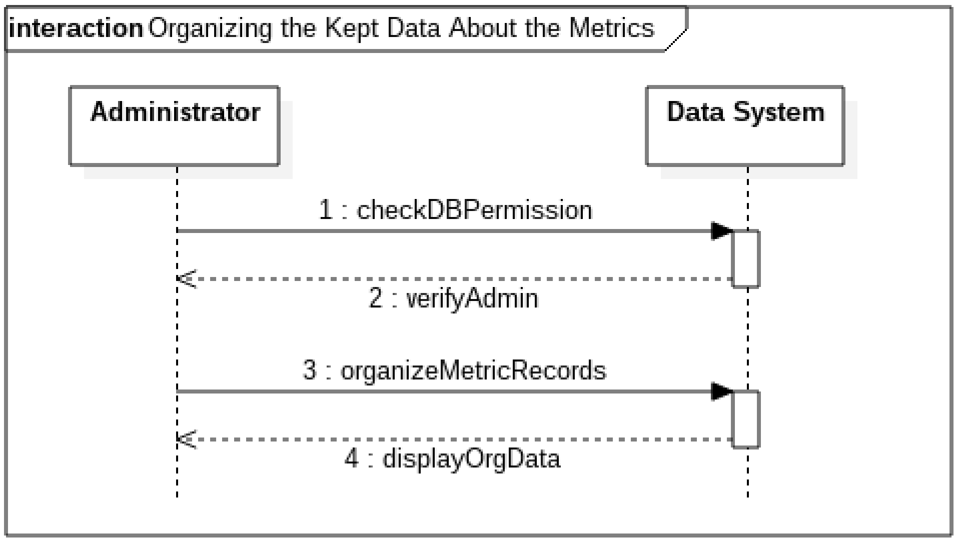

interaction Organizing the Kept Data About the Metrics
Administrator
Data System
1 : checkDBPermission
2 : verifyAdmin
3 : organizeMetricRecords
4 : displayOrgData

Figure 2 : Sequence Diagram of “Organizing the Kept Data About the Metrics”

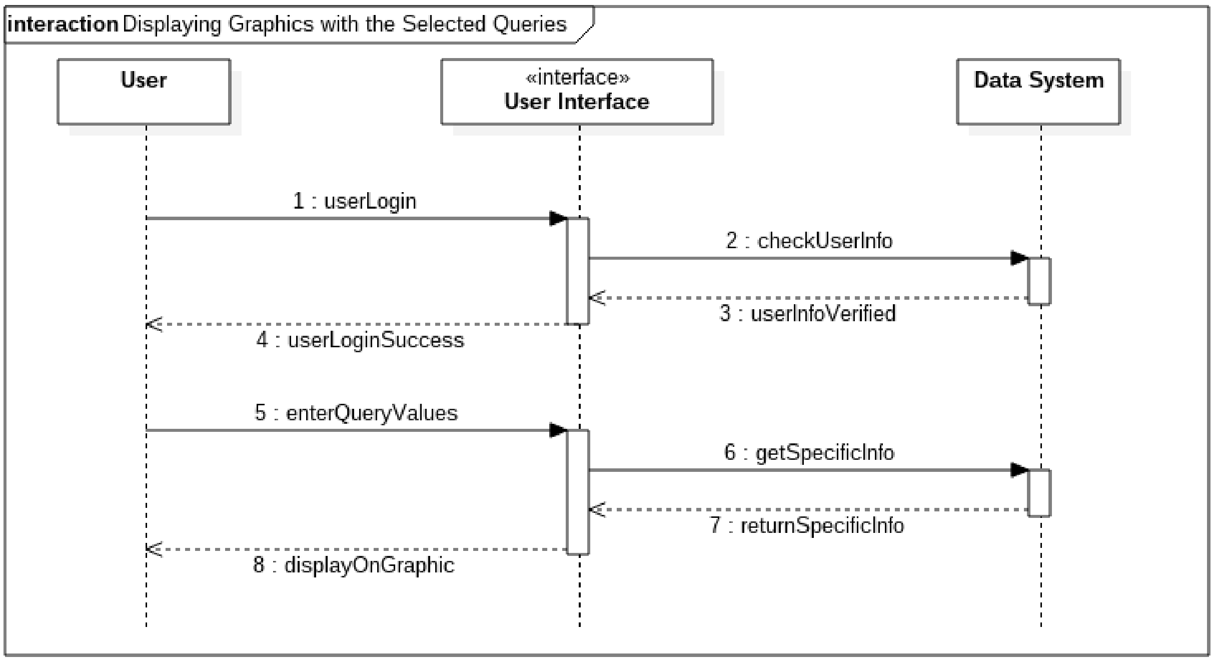


Figure 3 : Sequence Diagram of “Displaying Graphics with the Selected Queries”

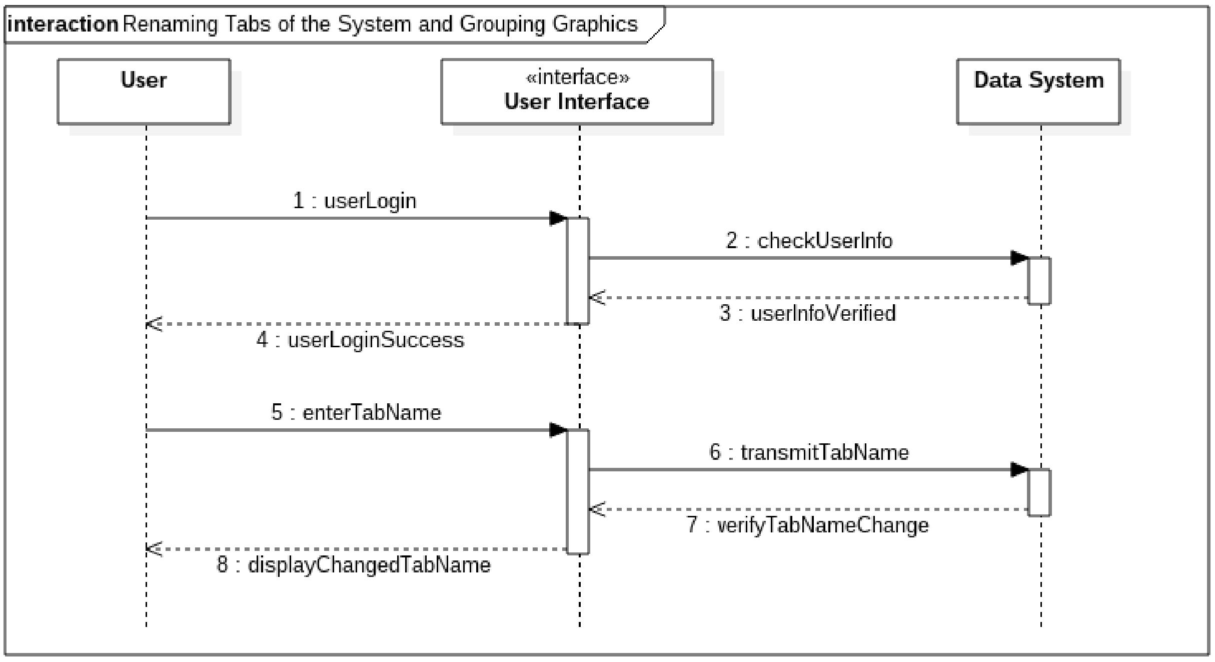


Figure 4 : Sequence Diagram of “Renaming Tabs of the System and Grouping Graphics”

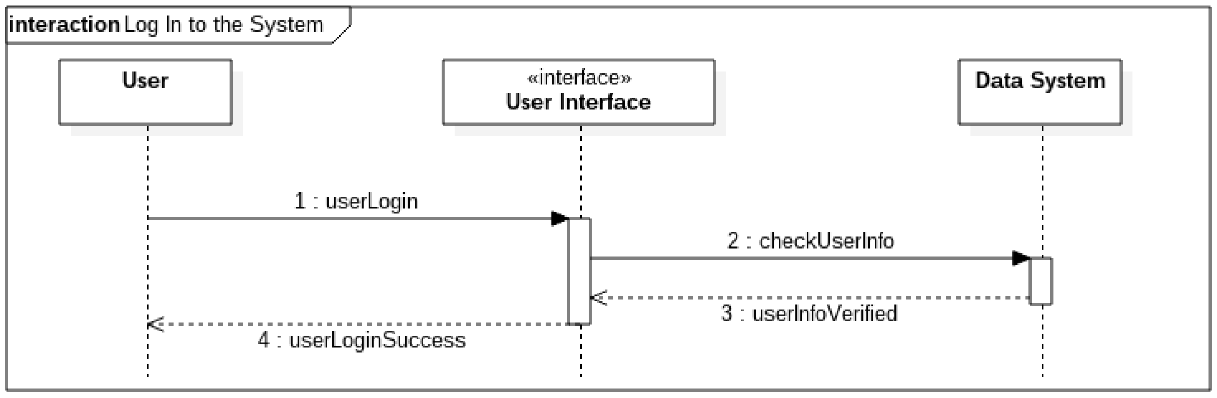


Figure 5 : Sequence Diagram of “Log In to the System”

1.2. REVISED PROJECT PLAN

This part includes our work package plans about the project from the beginning of the year to end of the year. Each work package has its own sub-packages which completed during the process.

(1) **WP1 – Core Functions and Logger Class Core Implementation**
This work package includes developing test servers. Log files were created manually during the dummy server test phase.
The second sub-package is defining basic metrics, log standardization and implementing Logger Class. Since we worked on standard log files first, we defined the basic metrics which are needed mostly to be monitored. After this phase, the logs were standardized according to metrics that we defined. On the behalf of flexibility, a core logger class was implemented. This logger class was able to extract a log file, in our standard, including different metrics from different given log files, thanks to the its modular system and added loggers.
The third sub-package is exporting basic metrics. The metrics which were chosen to be stored in our data storage were exported from the logs we standardized and prepared to be sent to the data storage.

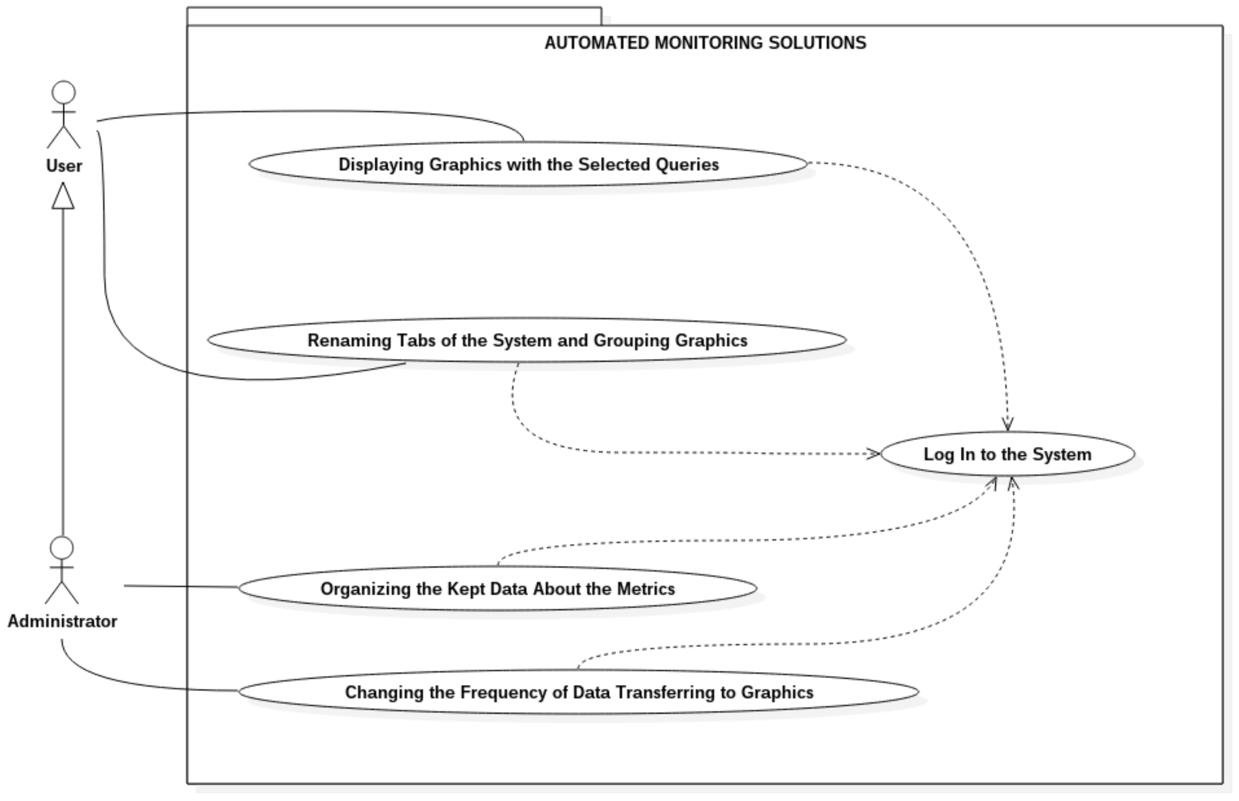
(2) **WP2 – Data Storage**
The first sub-package setting up a data storage system. MongoDB is used as data storage solution.
The second sub-package is data gathering and organizing. The prepared data were queued to the storage and the template of data storage methods were prepared. The storage system is in communication with the back-end server.

(3) **WP3 – Back-end Server Development and Component Testing**
The sub-package is starting develop both front-end and back-end servers. The data in our storage system were retrieved and processed using a back end server that was implemented in JavaScript (NodeJS). To verify that the data were properly processed by the back end server, and a simple front-end server was developed.

(4) **WP4 – Front-end Server Development and Visualization**
The first sub-package is monitoring and showing dashboards. The gathered and processed data were visualized as line graphs. The graphs aimed to be real timed which means they were refreshed within a specified time interval.
The second sub-package is implementing alerting mechanisms. A system which alerts the user depending on the options provided were implemented (via e-mail).

(5) **WP5 – Mobile Application Development (Android)**
The sub-package is implementing an android application which includes all the features web framework has.

1.3. USE CASE DIAGRAMS



AUTOMATED MONITORING SOLUTIONS
User
Administrator
Displaying Graphics with the Selected Queries
Renaming Tabs of the System and Grouping Graphics
Log In to the System
Organizing the Kept Data About the Metrics
Changing the Frequency of Data Transferring to Graphics

Figure 6 : Use Case Diagram of Automated Monitoring Solutions

Use Case Names:

- Changing the Frequency of Data Transferring to Graphics
- Organizing the Kept Data About the Metrics
- Displaying Graphics with the Selected Queries
- Renaming Tabs of the System and Grouping Graphics
- Log In to the System

Use Case Scenarios:

Use Case Scenario 1: Changing the Frequency of Data Transferring to Graphics

Use Case Name:	Changing the Frequency of Data Transferring to Graphics
Use Case ID:	UC1
Included Use Case(s):	UC5
Primary Actor(s):	Administrator
Description	Administrator changes the period of time which includes adding new data and refreshing real time graphic.
Precondition:	Administrator must have already logged in.
Trigger:	Administrator changes the frequency of data refreshing on real time graphic
Main Success Scenario:	1. Administrator navigates the part that includes changing frequency functionalities 2. Administrator decides the new frequency of the refreshing graphic 3. Administrator fills the input part on the framework related with the frequency of graphic refreshing 4. Administrator submits the new value to the system 5. System changes the corresponding part of the user profile file
Alternate Scenarios:	

Use Case Scenario 2: Organizing the Kept Data About the Metrics

Use Case Name:	Organizing the Kept Data About the Metrics
Use Case ID:	UC2
Included Use Case(s):	UC5
Primary Actor(s):	Administrator
Description	Administrator changes the kept data – mostly saved long time ago and should deleted to reduce the load of these data on data system -
Precondition:	Administrator must have already logged in.
Trigger:	Administrator notices that the data saved long time ago to database have became redundant burden for data system because these data are not viewed by owner of them anymore
Main Success Scenario:	1. Administrator detects the time interval which has an importance for the user – to illustrate user does not display the data belong to two months ago or more 2. Administrator deletes or reduce the data kept on the data system to reduce the burden of data system – to illustrate, as above if the data are not used anymore which belongs to two months ago are deleted or reduced by taking the average of longer time interval
Alternate Scenarios:	

Use Case Scenario 3: Displaying Graphics with the Selected Queries

Use Case Name:	Displaying Graphics with the Selected Queries
Use Case ID:	UC3
Included Use Case(s):	UC5
Primary Actor(s):	User
Description	User fills the corresponding query parts of the framework to display the graphic/graphics that constructed based on these queries
Precondition:	User must have already logged in.
Trigger:	User want to display the status of the system or the values of a specific metric/ metrics for chosen application name, physical server name that the application runs on, and the metric name with maximum time/ minimum time/ maximum value/ minimum value
Main Success Scenario:	1. User decides to see the values of a metric/ metrics on the graphic 2. User chooses the application name and physical server name that runs the application – if he/she does not choose the physical server name, all metric values on these servers that runs the application are displayed on graphic - 3. While displaying the graphics on the screen, if user wants to put boundaries for x-axis (minimum time/ maximum time) or y-axis (minimum value/ maximum value), he/she can fill the corresponding areas on the screen 4. After submitting the values to the system, back end server answers to request and sends the needed data to front end server. Front end server constructs the graphic/ graphics.
Alternate Scenarios:	

Use Case Scenario 4: Renaming Tabs of the System and Grouping Graphics

Use Case Name:	Renaming Tabs of the System and Grouping Graphics
Use Case ID:	UC4
Included Use Case(s):	UC5
Primary Actor(s):	User
Description	After constructing the graphic on a tab, the user rename the tab and grouping the new added graphic to other related graphics on a renamed tab – to illustrate “My Application_1 Related Graphics” -
Precondition:	User must have already logged in.
Trigger:	If added graphic is related with another graphics on the screen at that time, or if user wants to see more than one graphic groups which are related with each other, he/she can group them and rename the container tab.

Main Success Scenario:	1. User wants to add graphic which is related to another graphic/ graphics on the screen at that time 2. User groups them together in a tab to provide ease of use 3. The name of tab can be changed by the user not to forget why these graphics on the tab were grouped together
Alternate Scenarios:	

Use Case Scenario 5: Log In to the System

Use Case Name:	Log In to the System
Use Case ID:	UC5
Included Use Case(s):	
Primary Actor(s):	User
Description	User (or Administrator) log in to the system with his/her own personal information.
Precondition:	User must have already in the database.
Trigger:	User clicks the “Log In” button after entering the necessary information which belongs to his/her.
Main Success Scenario:	1. User navigates the “Log In” section 2. User enters his/her own credentials. 3. System checks if the entered parameters are valid. 4. System creates a new session for the user.
Alternate Scenarios:	2a: If the user provides invalid login information, framework notifies and redirects user to login section.

2. SYSTEM DESIGN

2.1. MODULE STRUCTURE

The module structure is explained via Class Diagram of the system.

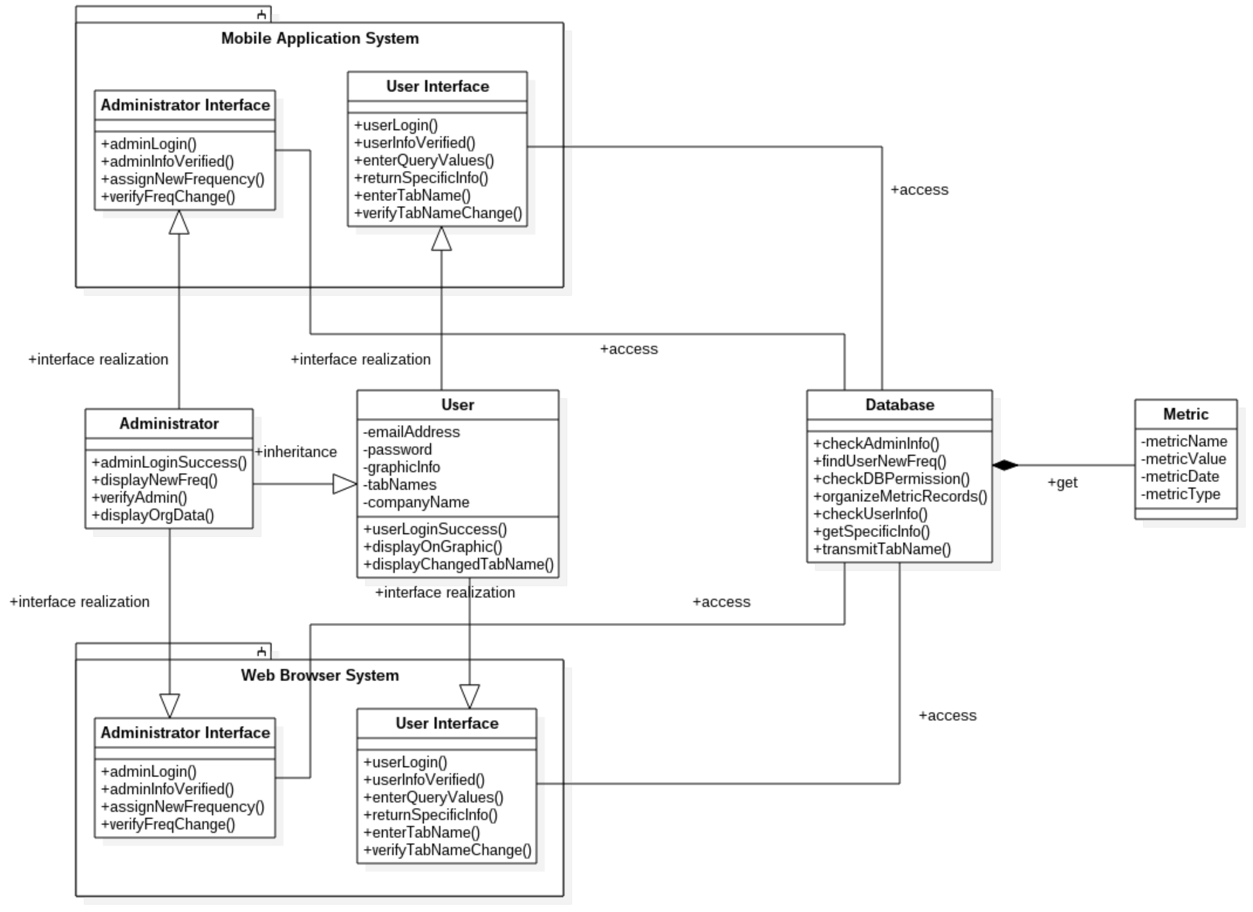


Figure 7 : Class Diagram of Automated Monitoring Solutions

Class Diagram Explanation:

Administrator Interface:

- **adminLogin():** Login functionality for administrator.
- **adminInfoVerified():** Administrator information is verified with the one in the data system.
- **assignNewFrequency():** New data frequency (resolution of data) is assigned by administrator for a specific user.
- **verifyFreqChange():** The change of the frequency of data for the specific user is done and verified.

User Interface:

- **userLogin():** Login functionality for user.
- **userInfoVerified():** User information is verified with the one in the data system.
- **enterQueryValues():** The query values are entered. At the first step, application name, physical server name, and metric name are entered. Then, time and value boundaries are entered at the second step and request is sent to back end server.
- **returnSpecificInfo():** Specific metric values for the corresponding queries are sent from data system to interface.

- **enterTabName():** Grouping related graphics with a tab name.
- **verifyTabNameChange():** The entered new tab name is verified by data system and returned to the interface.

Administrator:

- **adminLoginSuccess():** Login operation is succeeded by administrator.
- **displayNewFreq():** Administrator displays the effects of new frequency value on the system.
- **verifyAdmin():** Administrator information is verified for the data system access.
- **displayOrgData():** Administrator sees the effects of the data organization on data system which is deleting/ taking mean of the data values for specific time interval and deleting the original values.

User:

- **userLoginSuccess():** Login operation is succeeded by the user.
- **displayOnGraphic():** User displays the new graphic which was drawn with the given query values.
- **displayChangedTabName():** User displays the changed tab name on the system.
- **emailAddress:** Email address of the user.
- **password:** Password of the user.
- **graphicInfo:** The graphic information that default page of the user includes.
- **tabNames:** Tab names of the user's default page.
- **companyName:** Company name of the user.

Database:

- **checkAdminInfo():** The entered administrator email address and password match is checked with the ones in the data system.
- **findUserNewFreq():** Find the specific user in the data system whose data resolution should have been changed.
- **checkDBPermission():** For the administrator, when the issue is data organization (deleting/taking mean of the data in a time interval) check the permission for the administrator information.
- **organizeMetricRecords():** Deleting/taking mean of the data in a time interval request is taken from the administrator interface.
- **checkUserInfo():** The entered user email address and password match is checked with the ones in the data system.
- **getSpecificInfo():** Get the specific graphic information from the data system with the entered query values and names.
- **transmitTabName():** Tab name which the user has changed is transmitted to the data system.

Metric:

- **metricName:** Name of the metric.
- **metricValue:** Value of the metric.
- **metricDate:** The date which the metric is logged by the system.
- **metricType:** The type of the metric (e.g. String, Integer, Long Integer, Date, …)

2.2. SOURCE CODE STRUCTURE

In this part of the document, the important parts of the code structure in GitLab and the functionalities of these parts are explained. The ones related with the testing process are explained in Part 3.2.

1. Rest Folder:
    1. server.js file: The file which includes the code parts that makes web service work.
    2. app Folder:
        1. routes.js file: This file includes the code pieces which answers the requests and route them to required url paths.
        2. query.js file: This file includes the queries which are produced for the database system MongoDB.
        3. models Folder: This folder includes the data semantics of the system.
    3. config Folder: This folder includes the file with related to configuration of our system. Besides that the code implementation which is related with the login operation is here, passport.js file.
    4. views Folder: This folder includes the front end design files of the specific urls.
    5. node_modules Folder: The libraries we used in NodeJS files.
2. Logger Folder:
    1. /src/main/java/com/raad/ams Folder:
        1. abs Folder: The abstract classes of logger class and each log file record are here.
        2. api Folder: The interface classes are here.
        3. impl Folder: The implementation files are gathered here.
        4. utils Folder: This folder includes the utilities.
    2. pom.xml file: This file includes the connection between the libraries which we used.
3. LogServer Folder:
    1. /src/main/java/logserver Folder:
        1. LogBean.java: This class includes the semantics we used while converting log records to JSON object format.
        2. LogServer.java: The main log server codes which reads the log file and sends the records to RESTful web service.

2.3. COMPONENT DIAGRAM

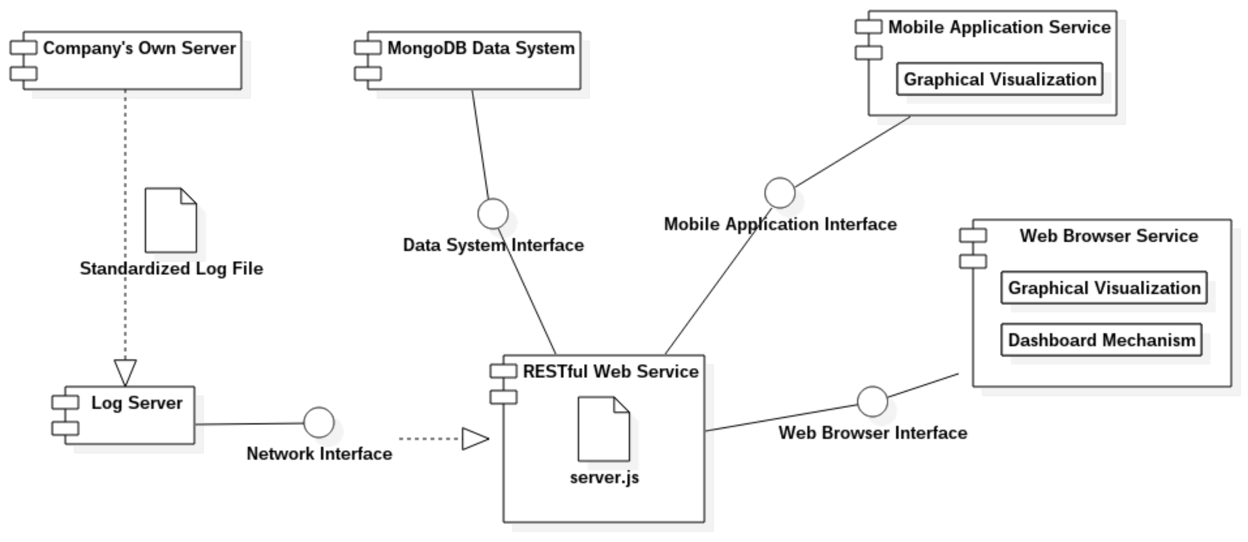


Figure 8 : Component Diagram of Automated Monitoring Solutions

2.4. DEPLOYMENT DIAGRAM

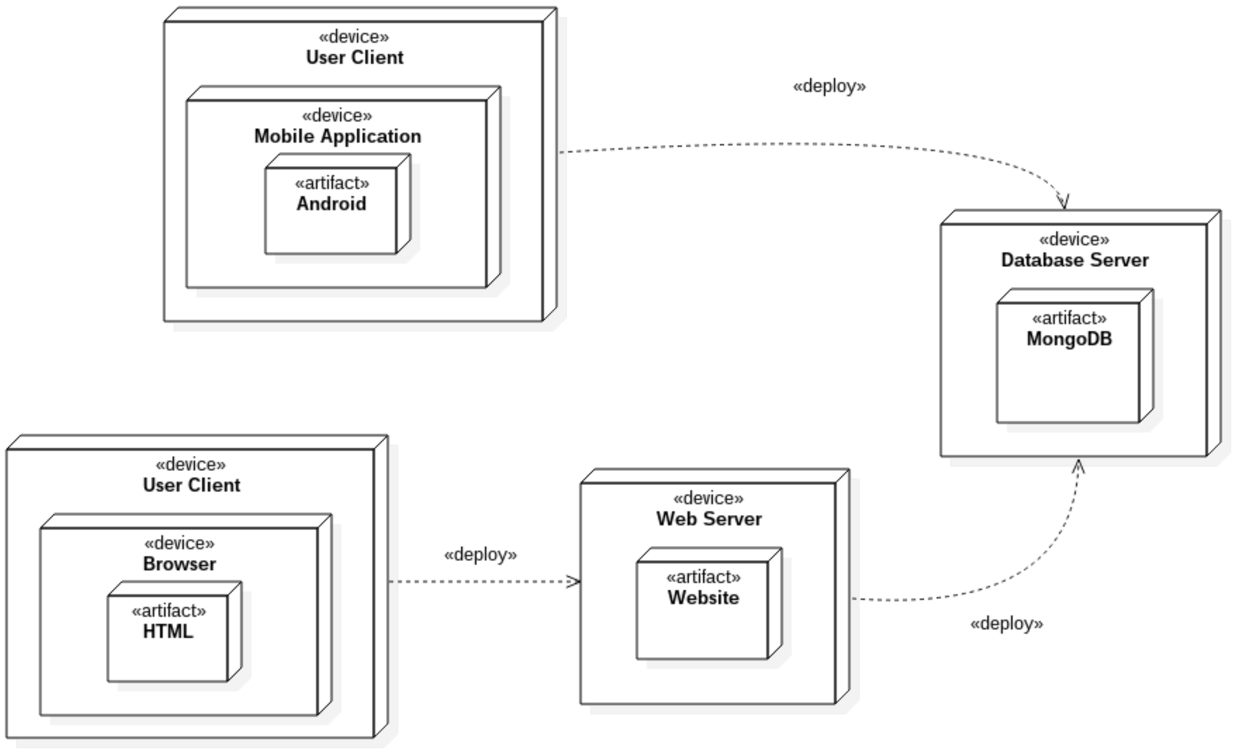


Figure 9 : Deployment Diagram of Automated Monitoring Solutions

3. TESTING

3.1. TEST PLAN AND SCENARIOS

The general mechanism of the "Automated Monitoring Solutions" begins with the user companies' own servers. Through the developing process, we could not work with the real companies' servers. Instead, we developed test servers which produce random metrics and metric values in different value ranges. We completed our testing process on the logger class functionalities, log file format, and network connection between log server and MongoDB via JSON formatted objects thanks to our test servers. At the end, we wrote down these metric values we gathered on the graphics.

An example testing scenario:

I. A dummy test server which belongs to Company_1 (there are more than one company records in our database) produce "Request Handle Time" metric in time.
II. This metrics are gathered by Log Server within a standardized log file. This log file standardization is done by us and creation of these log files is completed by our Logger Class module. The metrics which are wanted to be monitored are identified to Logger Class module.
III. After taking the log file, Log Server turns the log file records to JSON object and sends these objects to RESTful Web Service.
IV. The RESTful Web Service records these objects to corresponding area in the database.

V. When a request is received from the front end server, the query values (application name which this metric belongs to, physical server name which holds this type of metric information, and boundary values for this metric) are processed by back end server and required records for that query's result are sent to front end server to construct a graphical visualization.

Some little parts differs in the scenario are given in the test results part.

3.2. TESTING CODE STRUCTURE

Besides the code files which are responsible for the working of the system, we have some testing codes in the general code structure. These code file names and their place in the general code structure are given below.

1. Logger Folder:
    1. /src/main/java/com/raad/ams/ Folder:
        1. appServer file: This includes our dummy test servers which is a server-client model example.
    2. /src/test/java/ Folder:
        1. junit Folder: This includes the unit test files.
2. LogServer Folder:
    1. /src/test/java Folder:
        1. junit Folder: This includes the unit test files.

3.3. TEST RESULTS

We perform some other testing scenarios which includes

- more than one company record transferring to the system to confirm the way of writing metrics to their owner company's area on database
- creating users' own space on database which holds their email address, password, default front end page graphic areas design (i.e. which tab have which graphic, which metrics are displayed on these graphics, what are the boundary values of x-axis and y-axis of these graphics, what are the names of the tabs as default)
- entering new resolution values for real time graphical monitoring (i.e. refreshing graphic in every "determined" time interval)

After testing these parts, we had healthy results in the end. We could displayed the desired metrics on desired tabs and graphics.