Pentad X-5 Assignment 4 Report

Enhanced design and architecture (self-healing loop)

PREPARED FOR

EECS 4314 Advanced Software Engineering

PREPARED BY

Muhammad Arifur Rahman

Samuel On

Jaiveer Singh

Dong Jae Lee

Karmit Patel

Website: http://pentad-x5.unaux.com/

Github: https://github.com/arifrahmanca/Smart_Employee_Management_System

SEMS Application: https://employee-managment-f5252.firebaseapp.com/

Source Code: https://github.com/KarmitP98/Employee-Managment.git

Table of Contents

1. Overview

The primary goal of this document is to provide an update to the proposed enhancement to our Smart Employee Management System (SEMS). SEMS is a cloud-based software application designed to reproduce the traditional employee management system in a virtual environment. The proposed enhancement is to log user requests through the application for root cause analysis or failure mitigation. The document will discuss the components and interfaces that require changes at a high level and the potential risks that accompany the system. The overview will include the following information: proposed enhancements, enhanced use cases, enhanced sequence diagrams, enhanced component diagram, enhanced class diagram, subsystem diagram, effects of enhancement, enhancement of architecture, alternatives for enhancement, plants for testing, potential risks, mitigation tactics and lessons learnt.

2. Proposed enhancement

The proposed enhancement in this sprint of development is to add a self-healing loop to the system to achieve zero downtime. This enhancement is well motivated as it will increase the availability of the SEMS system, creating a more attractive product for end-users. The self-healing loop will follow a MAPE-K architecture as such it will be split into 4 phases which can be viewed as two broader phases, the detection of failures and the mitigation of such failures. The detection of failures phase consists of monitoring and analyzing, while the mitigation of failures is done in the planning and execution phase. The monitoring phase will get raw data from the system components after which the analysis component will analyze that raw data to detect faults within the system. Faults within the system can include anything from the failure of certain components to provide functionalities to violations of certain non-functional requirements like having a response time of less than 2 seconds. As such when a fault is detected a plan is created by the system to fix such error and then that plan is executed, which changes variables of the system so the detected fault is no longer apparent in the system. Self-management loops like a self-healing loop can be extremely beneficial as they include the ability to automate the process of dealing with faults that can occur within a system. Which in turn decreases the amount of time it takes to deal with such faults ultimately decreasing the amount of downtime of the system.

3. Enhancement of Architecture

The enhancement proposed and implemented is a logger logic that logs correct access to the system as well as errors that occur which can be used for analysis. Essentially every instance where a user attempts to access SEMS there will be a log in the database that

reflects the attempt whether it was successful or not. These logs will have simple indicators that can be used by the admins for analysis. With this basic functionality added to SEMS, 3-Tier and MVC (Model-View-Controller) architecture of the system was altered.

There were a couple additions that were required within the 3-Tier architecture to ensure that the enhancement described above was implemented correctly. First within the middle controller or business layer of the architecture, a logger needed to be added to ensure the actions taken by the user in the presentation or client layer were recorded. Once the logger in the business layer records the log from the user in the client layer it will pass on the data to the lower database or data layer where the database for the system resides. The architecture will perform almost identical to the previously proposed description without the logging feature simply with an addition of another set of information that needs to be handled which would be the user logs.

Another modification took place on the MVC architecture due to the enhancement. The admins or the HR department in SEMS were granted another set of features within their view, the logs. The user actions are logged for the admins of the system in the model and the admins have another set of information they can access in the view by utilizing the controller. Lastly the previously proposed SOA (Service-Oriented Architecture) remains unchanged with the enhancement.

4. Alternatives for Enhancement

For the proposed architecture above, our team had alternative options to implement the additional feature. First consideration was the logger logic for user actions. Currently SEMS will record all user actions into the logger, and it will all be saved into the database. By saving all user actions even if they are not errors, the logs will continue to grow in the database if they are continually stored without much restriction. Alternative to the current method, the team could have implemented an improved method of logging where the logs will be segregated into different categories such as false positives, errors, and proper access. By separating out the logs into different categories the admins will be able to discard logs that are not required for analysis and free up space within the database. Another consideration our team has taken was the placement of the logger logic within the 3-Tier architecture.

Although the proposed enhancement has placed the logger logic and data transfer features within the business layer of the 3-Tier architecture, there could be a point made for placing it inside the client tier. Instead of having to store all the logs in the database, the logger could be placed into the client tier and the logs will not utilize much space of the database in the data tier. This ultimately means the logs will be saved temporarily for viewing or extracting

for analysis before being deleted. By implementing the logger in such a way, we could eliminate the extra overhead created by passing on the logs from the client tier to the data tier and back. These two alternative methods of enhancement were considered while designing the enhancement but ultimately not chosen for the final implementation.

5. Interactions between Subsystems

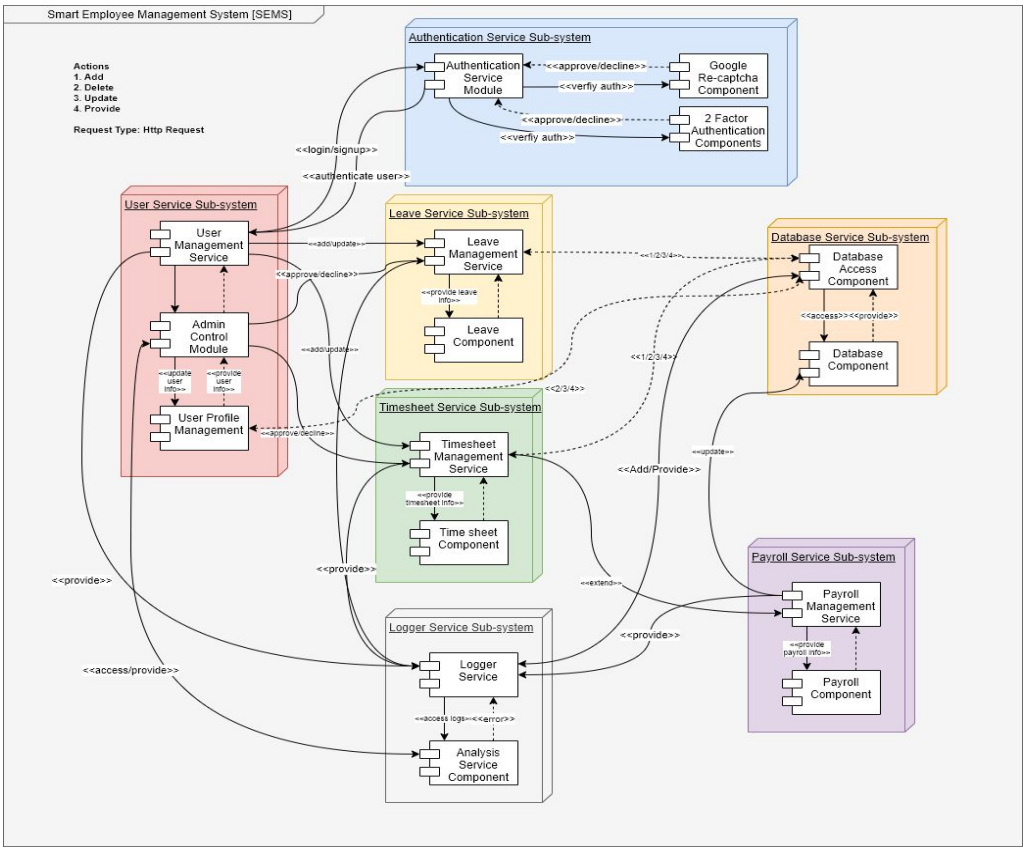

Figure - 1: Enhanced Sub-system diagram

We have introduced a new Subsystem in our software named Logger Service Sub-system.

- Logger service subsystem works in the background. It is accessed asynchronously in our system, i.e. any calls to a component, access to the database, input any value or changes in the background data will trigger logger service.
- Logger service has access to the database subsystem as well as every other subsystem that performs any data manipulation in the system.
- Logger subsystem consists of logger service that is directly accessible by every other subsystem, and Analysis Service Component that is only accessible via an asynchronous call from the user service subsystem.
- The main job of the analysis service is to read all the logs accessed by the logger service from the database and analyze whether the log is an error or not.

6. Illustration of enhancements using diagrams

6.1 Use Case : SubmitLeave

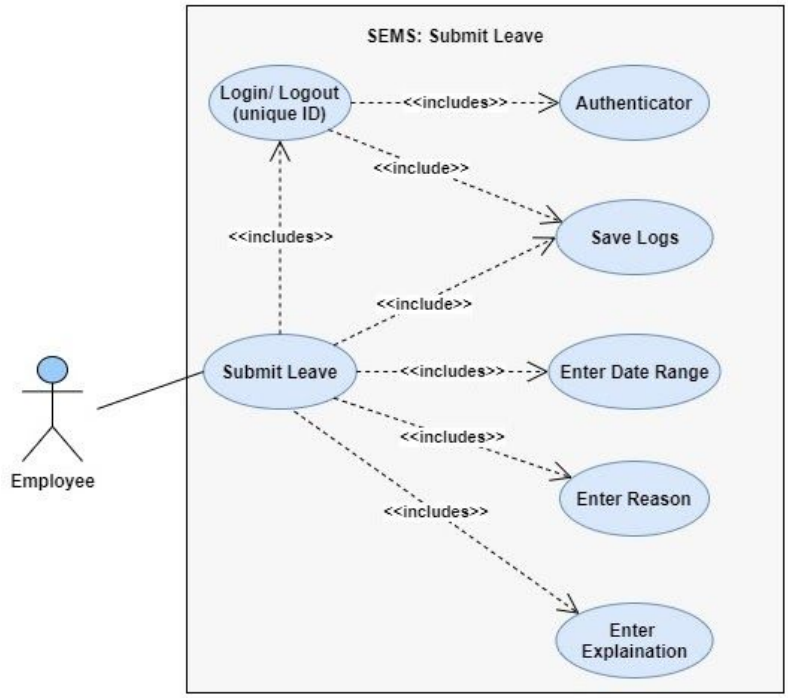

Figure - 2: Use case (Employee submits leave request)

Participating actors: Employee
Entry condition: Employee is logged in to the system
Flow of Events:

1. The employee requests access of the LeaveService to submit leave
2. The system returns LeaveService to the employee
3. The LoggerService logs the event and saves it to the database
4. The employee requests to submit leave to the system
5. Leave request submitted and saved into the database
6. The LoggerService logs the event and saves it to the database

Exceptional Flows:

1. a) The employee fails to access the LeaveService
   b) The LoggerService logs the failed attempt and saves it to the database
   c) The system retries to return LeaveService to the employee
2. a) The employee fails to submit the leave request
   b) The LoggerService logs the failed attempt and saves it to the database
   c) The system retries to submit leave request

Exit condition: Employee logged out from the system
Non-functional requirements: Accessing the LeaveService and submission of leave request by employee, each will proceed within 2 seconds.

6.2 Use Case: AccessSystemLogs

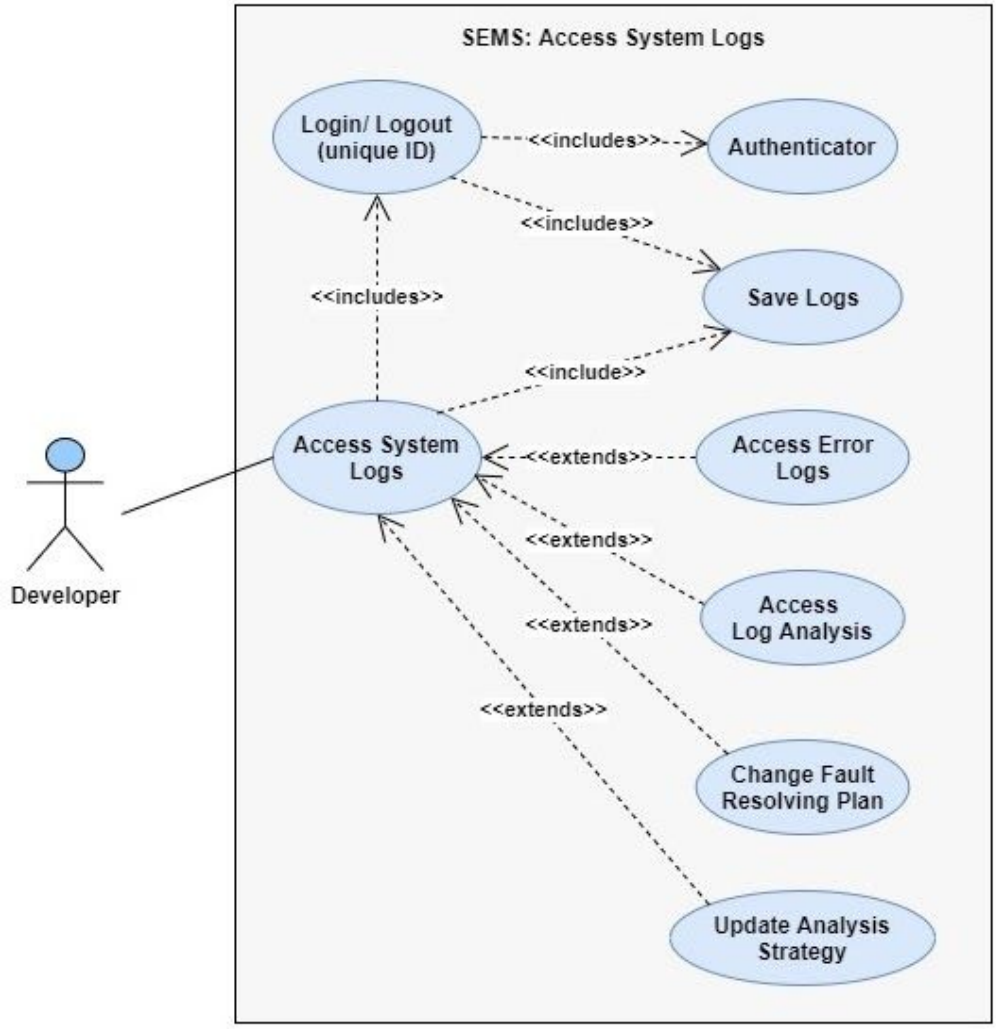

Figure - 3: Use Case (Developer accessing system logs)

Participating actors: Developer
Entry condition: Developer is logged in to the system
Flow of Events:

1. The developer requests access of system logs from the database
2. The system returns logs to the developer
3. The LoggerService logs the event and saves it to the database

Exceptional Flows:

1. a) The developer fails to access system logs
   b) The LoggerService logs the failed attempt and saves it to the database
   c) The system retries to return system logs to the developer

Exit condition: Developer logged out from the system
Non-functional requirements: Developer's request to access system logs proceed within 2 seconds.

6.3 Enhanced Sequence Diagram

The sequence diagram below represents sub-system interactions and data flow while an HR employee approves the payroll of an employee. Accessing data for all old components will remain the same. The only exception is that whenever a user accesses a component, that component will communicate directly with the Logger Service and the Logger Service will create a log for that request and save it into the database. In cases where any component fails to respond or fails to retrieve data from the database, an error log associated with an error message will also be saved into the database for that fault.

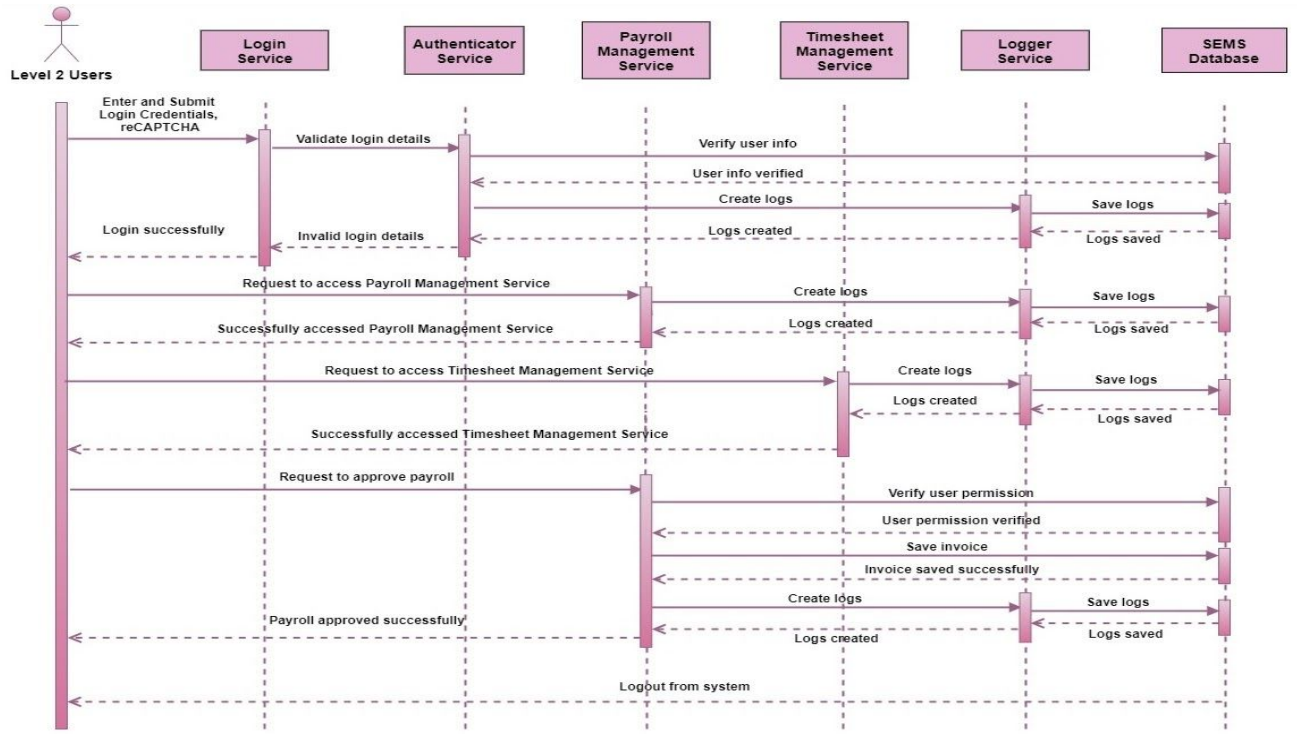

Figure - 4: Sequence Diagram showing HR Employee is approving payroll.

After providing accurate login credentials and reCAPTCHA, the user is verified and authenticated to log into the system. The Login Service interacts with the Authenticator Service to authenticate and the database to verify user's login information. The logging log is saved into the database by the Logger Service. Then the user can access the Payroll Management Service. The log for accessing the Payroll Management Service is saved to the database by the Logger Service. Before approving the payroll, the HR employee can access the Timesheet Management Service to verify worklogs of employees. Again, the Logger Service will log this event and save it. Finally, the HR employee can approve the payroll, save into the database, and the associated log is also saved by the Logger Service.

6.4 Enhanced Component Diagram:

To fulfil the required enhancement, we have added 2 (two) new components to our system. They are: Logger Service and Analysis Service. The main functionality of the Logger Service is to log every event following requests from the end users. It logs all the successful as well as error logs while accessing components by users. Attempts to access any component by the users will invoke the Logger Service and it will save all logs to the database. The Analysis Service accesses all the logs from the database and performs the analysis of the error logs which will be further used to mitigate system faults. The following diagram shows all of the components of our system including the added ones.

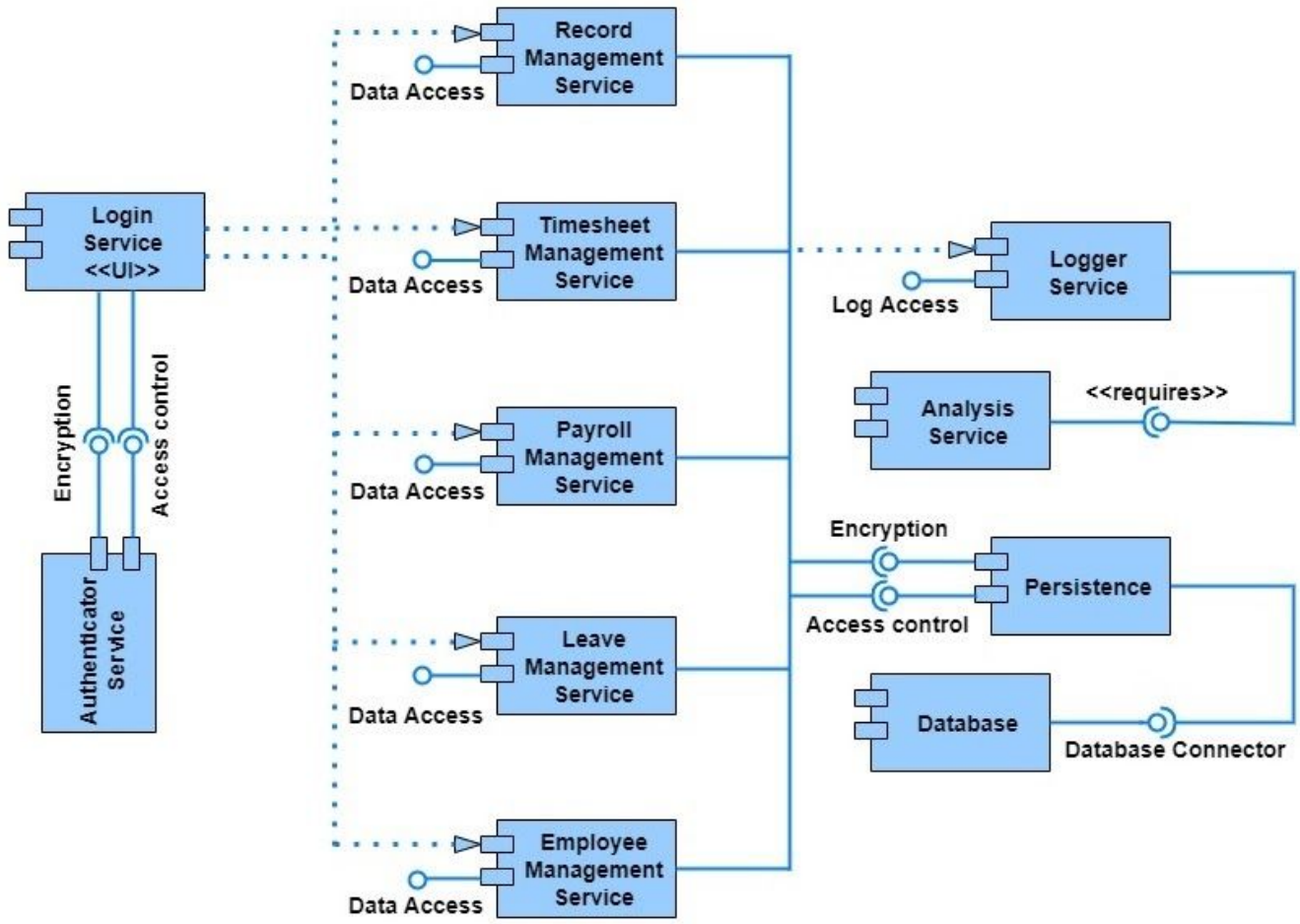

Figure - 5: Component Diagram showing added components due to enhancements

6.5 Enhanced Class Diagram:

The changed architecture is represented in the class diagram below with added classes and changed features highlighted with red. The model has direct access of the System logs through a Data Access Object Layer from the database. Depending on the analysis done by the Analysis class, the controller will execute planned action to mitigate the fault occurred by the system and send corresponding responses to the UI. Architecture design for all other classes remains the same as the previous implementation.

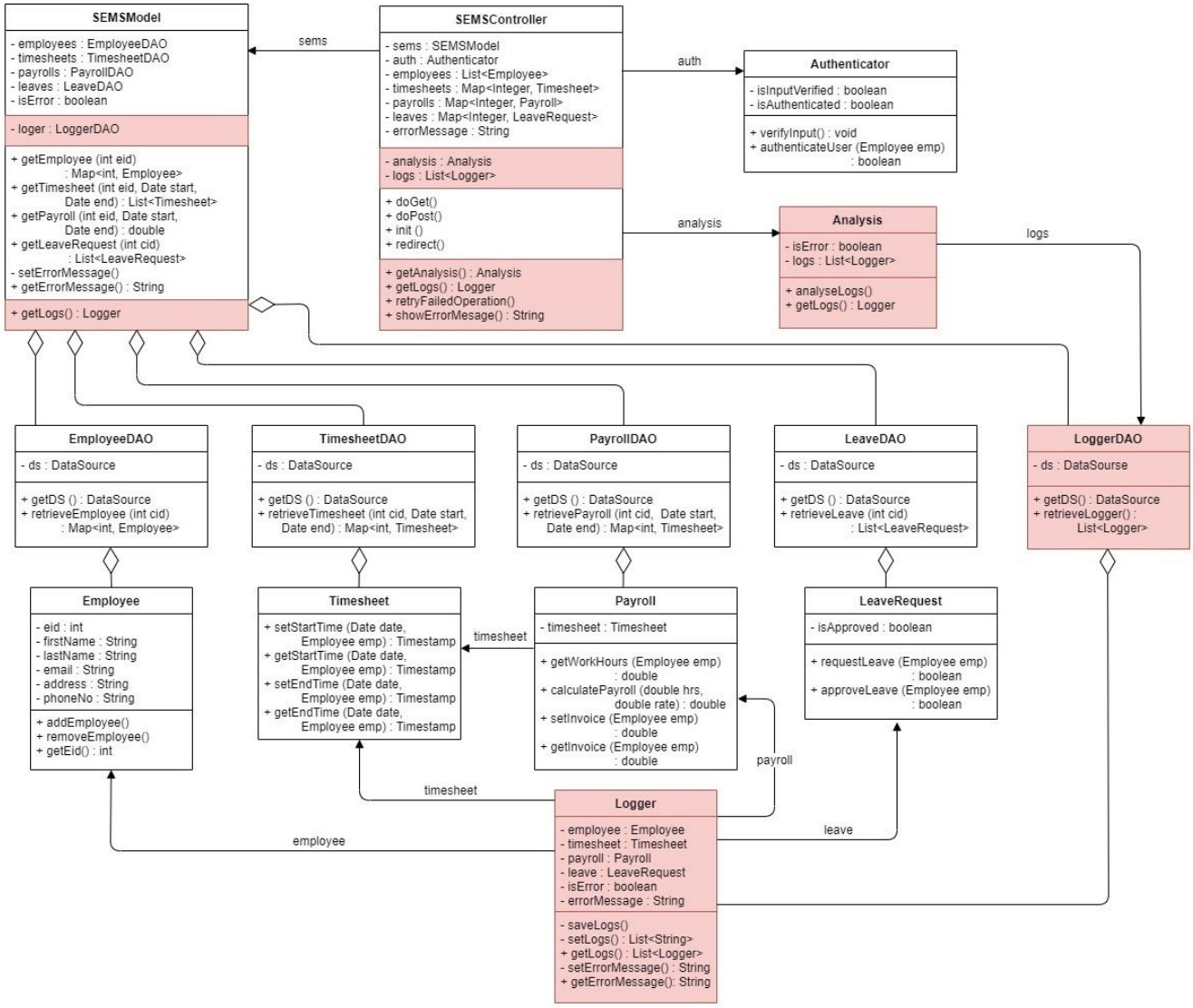

Figure - 6: Class Diagram with added classes and features after enhancements

6.6 Deployment Diagram:

The deployment policy did not change following the enhancements. Our software has deployed in the Firebase cloud as before which provides the platform, Backend-as-a-Service (BaaS) for our web application. We used Firebase Admin SDK to deploy our SEMS.war application to Firebase.

The software uses the Firebase database which is a cloud-hosted NoSQL database embedded in the Firebase cloud. This NoSQL database provides real-time updates of data and ensures the performance of our system. The database server is backed up automatically which ensures the safety of user's data. The cloud server also provides the interface for the clients to use the software through web browsers using HTTP internet protocols that ensures the availability of the system over the web.

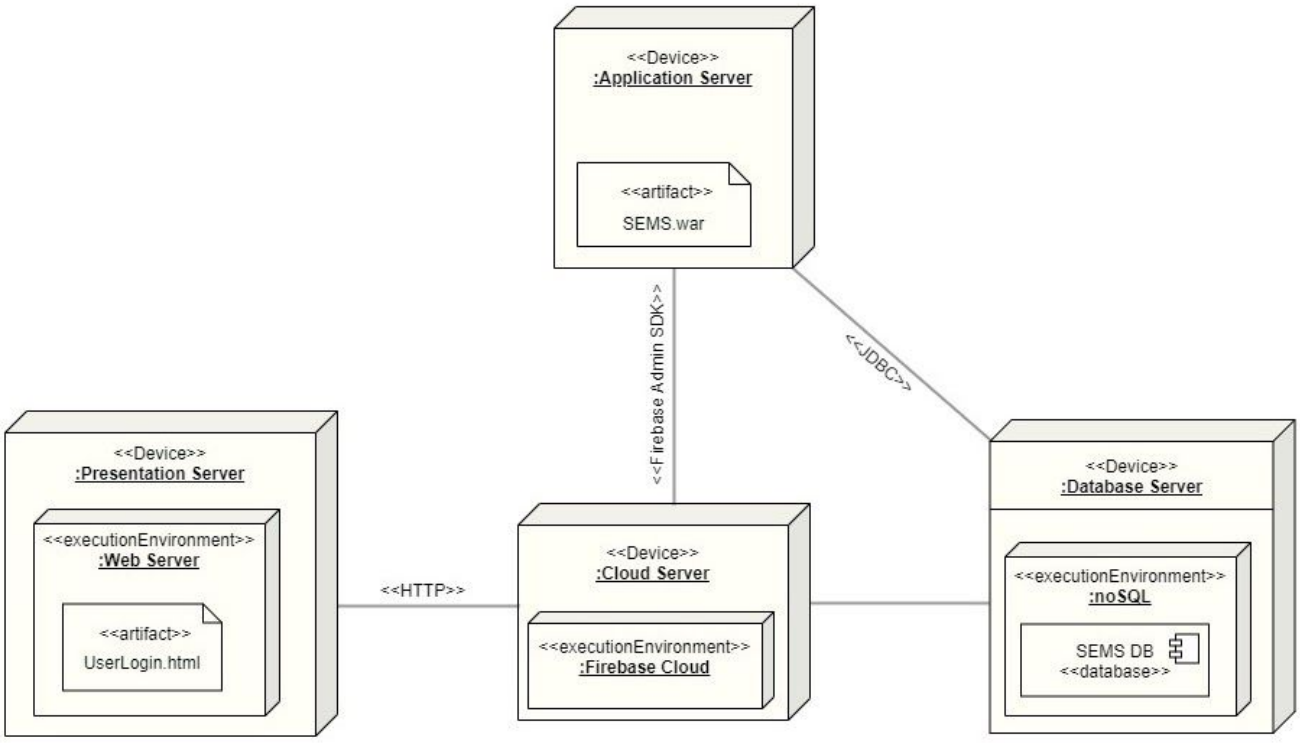

Figure - 7: Deployment Diagram showing nodes and their communication protocols

7. Effects of Enhancement

The effects of the enhancement on the maintainability, evolvability, testability, performance, security, and privacy are presented

7.1 Maintainability

- Our new enhancement provides the users with a way to manage and report accurate eros that occurs in the system. At the same time, it increases the maintenance from a developer's point of view.
- As the new subsystem is accessed by all the other components and subsystems, any changes in the current components will result in a broken system. Hence, all subsystems must be notified of any changes in the sub-system and vice versa.

7.2 Evolvability

- Any new component added to the system, must access the logger sub-system. Hence, the evolvability is not limited but resource intensive for the system.

7.3 Testability

- The new component provides easy testability of any new or existing components. The analysis service component logs all the errors in the system that can be debugged later.

7.4 Performance

- The new sub-system is accessed by all the components of the system; hence a minor performance hit is acceptable during heavy usage.
- Although adding the new enhancement will not hinder database access or current component functionality in any way, shape, or form. The logger runs in background and is accessed by all the components asynchronously, i.e. the logger is event driven.
- It might affect the performance of the database since data is being constantly written and accessed front he database but that can be mitigated by utilizing a categorization method to log only errors that are system failures, critical functionality errors or database access errors.

7.5 Security & Privacy

- The enhanced system does not output or access any more data than it did before enhancement, hence there are no drawbacks or issues regarding user security and data privacy while using the application.
- The new enhancement is intended to determine any system and service failure and not manage or monitor user data and data transfer between components.

8. Plans for Testing

It is apparent that the proposed enhancement of adding a self-healing loop to the system creates many more interactions within the system as the self-managing loop interacts with the many components of the system in multiple ways. Specifically, during its monitoring and execution phases. The monitoring phase collects data from the components of the system after which if faults are detected a solution is executed by changing certain variables within the system in the execution phase.

To test the interactions of the enhancement and the components it interacts with, the system can be tested to see what happens when certain conditions are met within the system. Specifically, when certain errors are encountered. So, in this case we proposed the enhancement of adding a self-healing loop to achieve zero downtime. As such we can extensively test the application to see if the logging done by the system is correct, if it detects certain errors, and how it will deal with fixing such errors. We can have test cases in which certain faults are intentionally made to see if the system detects such error and if it can deal with it. Through doing this we could see if the interactions made between the components are correct and if the enhancement functions as it should.

9. Potential Risks due to enhancement

Due to the enhancement proposed and implemented in the system, it exposes the system for risks in sectors such as security, maintainability, and performance. The addition of the logger features exposes the system to another attack vector for the threat actor to access. Since there could be imperfection in the implementation of the feature and perhaps the data access by the system, addition of another feature exposes the system in an alternative way.

In terms of maintainability, the additional feature of the logger increases work of maintenance of the developers and the designers of the system. Considering the logger is a major functionality of the system that will require frequent software maintenance, it will create a burden for the developers when a crucial error is found. With regards to the performance effect of the logger features is that it will add another overhead for the user action as it needs to be recorded and saved in the database.

10. Self-healing loop and Mitigation Tactics

The following diagram represents the self-healing loop of our system and gives an overview of the mitigation tactics:

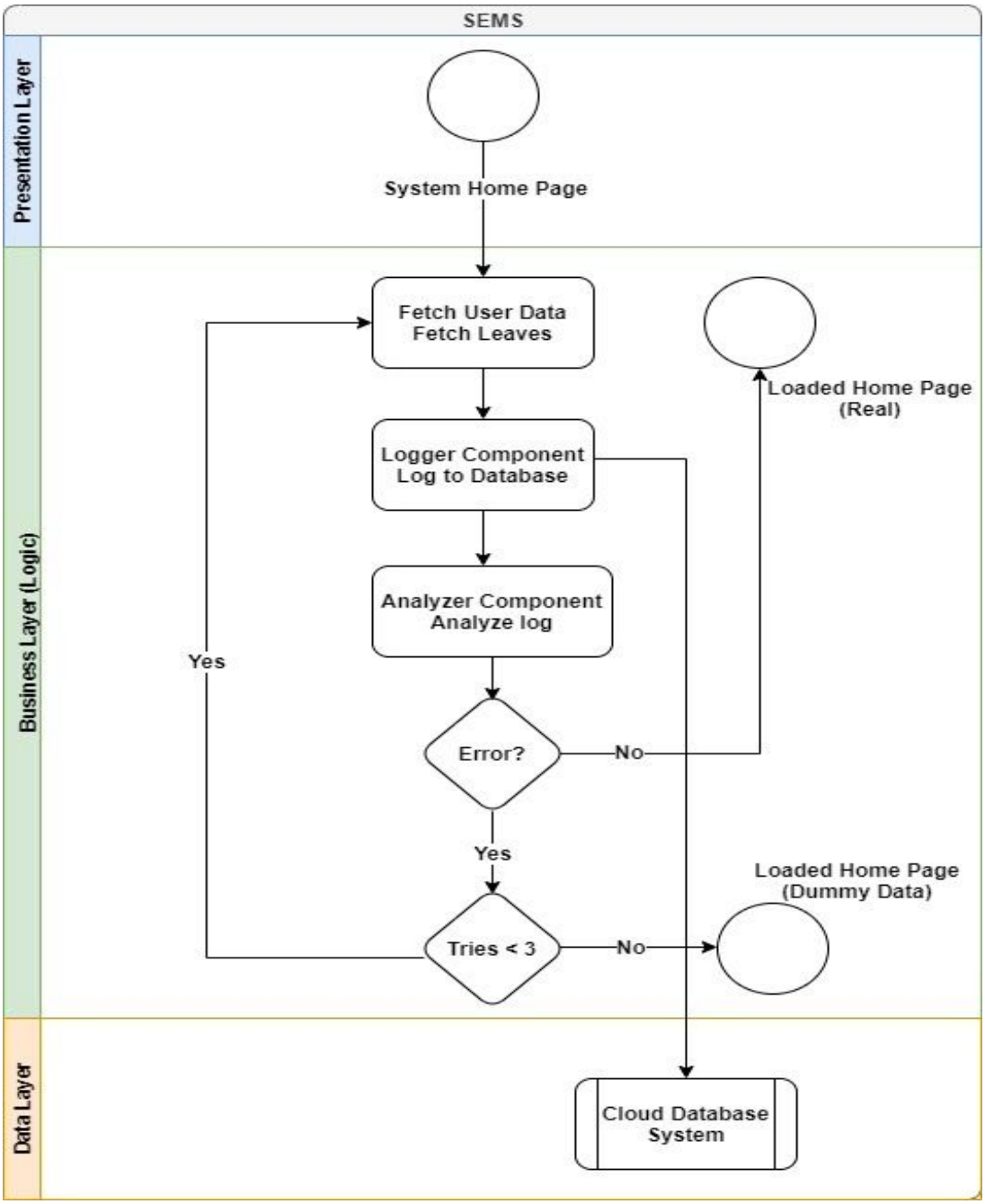

Figure - 8: Self-healing loop showing the tactic "Retry failed operations"

Our system will use **“Retry failed operations”** as a mitigation tactic to ensure the Availability of our system with zero downtime. The system will analyse all the error logs occurred following HTTP requests from the end users. Depending upon the resultant analysis from the Analysis Service, the controller then executes the planned action which in this case is to redirect the request to retry. Suppose the scenario in the above diagram where an employee is trying to retrieve the LeaveService from the database, but the database is unable to respond to this request for some inevitable reasons. This fault is logged in the system by the Logger Service with an error message and the request will be redirected by the controller as described.

Since 95% of all response times are targeted to be less than 5 seconds by our system, the controller will redirect the request if it passes more than 5 seconds. The controller keeps track of the retry attempt and will allow a maximum of 3 consecutive retries. If the system does not respond within these 3 retry attempts, then the controller will update the model to show a default error message to the client on the browser. In the scenario when 3 retry events fail to respond, the Logger will log another error message to the system. These error logs can be used by the developers for debugging or to further enhance the mitigation tactics or analyzing policies.

11. Lessons Learned

In this phase of software development, the enhancement of adding a self-healing loop to the system was proposed. As such many different scenarios were encountered specifically to do with how this new proposed enhancement will fit into the existing system and how it will change it. During this process we learned what kind of data a self-healing loop monitors, how it analyses that data to detect faults or potential causes of faults. Further we looked at specific faults that can occur within a system and how we would deal with such faults if they were encountered. Helping us better understand how a self-healing loop acts on a system. We also learned how difficult analysis of data can be to detect faults within a system. Moreover, we learned how difficult the healing of such a fault can be. Systems are complex and creating a self-management loop within it can be very complicated depending on its purpose.