Technical Document

Employee Manager

Company Name: EPI-USE

Members: Christof Steyn

Contact: christofsteyn98@gmail.com

Table Of Contents

Introduction

The Employee Manager System is an innovative solution that streamlines the process of organizing and visualizing organizational structures. This software offers an intuitive interface for storing and managing user data while dynamically constructing visual hierarchy tree graphs. These graphs illustrate the reporting structure within your organization, providing a comprehensive and easily navigable overview of the relationships between users, their roles, and their reporting managers.

Key Features:

- User Data Management: The system allows you to input, store, and manage comprehensive user data including roles, personal information, contact information, and reporting information.
- Visual Hierarchy Tree Graph: The software dynamically generates visual representations of your organization's hierarchy, simplifying complex relationships and offering a clear and concise overview of reporting structures.

Architectural Design Strategy

I have decided to approach the system design using the strategy of **design based on quality requirements.** There are many quality and functional requirements that need to be implemented and thus I have a well defined base onto which I can start implementing the system on.

Architectural Style

Although the system can be considered as a Multitier Architectrure or Layered Architecture, that consists on the client and server side, I believe that the system can be described more in depth by the following architectural patterns; Database-centric Design and Domain-Driven Design.

Multitier or Layered Architecture

The main systems architecture is a Layered Architecture comprising four layers. The Presentation Layer is what the user will see in their web browser and is the part of the system they will be directly interacting with. The Business/Application Layer is where all the systems logic takes place. Here requests are processed from the presentation layer and passed down. The Domain Layer is next. Here all the system models, controllers and other services provide functionality to the layer above it and pass data from the database upwards. Lastly, the Database Layer. This is where the system's data is stored in the database for writing and retrieval and makes up the center of the system.

Database-Centric Design

This design pattern was chosen due to the crucial role that the database plays in the system. As Employee Manager is an organization management tool, it needs to be constantly referring and reading data from the database. As it is also using a relational database management system (RDBMS), as opposed to a file-based data structure or an in-memory system, I believe this architectural pattern aptly encompasses the base structure of the system.

As stated above the system does form the broad structure of an MVC pattern and thus the database makes up the model portion of it, which further justifies the need to construct the system using this approach.

Domain-Driven Design

The third design pattern being focused on is the Domain-driven design. This approach focuses on structuring the controller and functions to match the domain according to the specifications. Under this approach the aim is to structure the classes, methods, variables and functions in such a way that it matches the business logic. This approach will also allow developers to modularise the code and thus increase the reusability and maintainability of the functions in the system for efficiency.

Architectural Quality Requirements

Availability

Specification

The system should be operable at all times to provide the most accurate and up to date version of the database to ensure that the company and all users involved can work to the highest standards. Any number of users should be able to use the system and there should be no noticeable change in performance or speed. Due to the system being of a Database-centric design it is important that the system is able to connect to the database at all time and display the most recent and accurate data stored.

Quantification

Due to the system being hosted on an AWS EC2 t2.medium instance, it can allow for complete availability at all times. Due to the nature of the CI/CD pipeline, maintenance can happen offline and when changes are made and pushed to the repository the website is instantly rebuilt with the changes and no loss in performance should take place.

Security

Specification

Security is another very important aspect to the system. Due to the personal and sensitive data on the employee, access to the system needs to be limited and properly secured. No unauthorized access to the system can take place as it is important that data cannot be altered or deleted by anyone except the users that exist on the system.

Quantification

Only users that have an account on the system i.e. the users is added to the database with an email and password, can access the system. This limits authorization and access to the system from anyone outside the organization. User credentials are being stored within local session storage once a user is logged in.

Performance

Specification

The system should be operable at all times to provide the most accurate and up to date version of the database to ensure that the company and all users involved can work to the highest standards. Any number of users should be able to use the system and there should be no noticeable change in performance or speed. Due to the system being of a Database-centric design it is important that the system is able to connect to the database at all time and display the most recent and accurate data stored.

Quantification

Due to the system being hosted on an AWS EC2 t2.medium instance, it can allow over one hundred users to use the system concurrently. EC2 also allows for burstable performance. This means that they have a baseline level of CPU performance, but they can also burst above the baseline when needed. Another important aspect to note is that all database transactions should be locked while performing any write changes so that data is not lost or the wrong data is returned at a later stage.

Durability

Specification

As the system is a Database-centric design, the durability is part of the ACID (atomicity, consistency, isolation and durability) property which guarantees transactions have been committed and stored properly and permanently in the database. As there may be multiple read/writes to the system happening simultaneously it will be important that the information stored is the correct and most up to date version.

Quantification

We can achieve durability by writing all transactions to our RDS database instance. Incase of any failure, the database can be restored to a version before the transaction as AWS creates snapshots of the database and creates a full backup at specified intervals. As the client has specified the system to be aligned with ISO standards (as described in the SRS document), no entries may be deleted from the database. Instead users and assets will be declared as revoked and their information will removed from

the main database and stored in the secondary “delete” database where all information that has been revoked will be stored .

Usability

Specification

The system should be easy to use and easy to navigate. The interface should be easy to understand and allow the user to come familiar with it with little to no effort. Interactions in the system should be seamless and promote productivity and help increase the throughput of work.

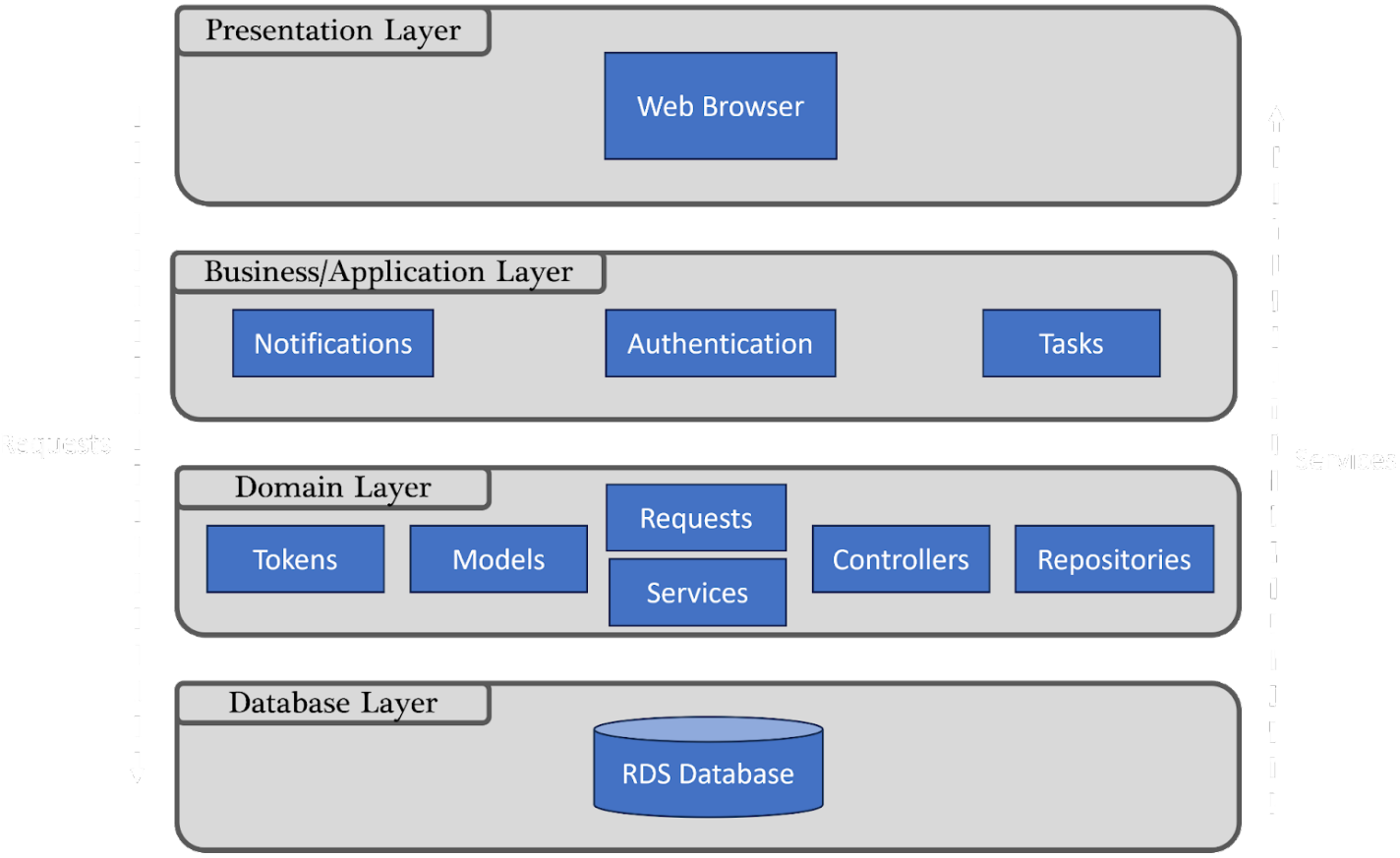
Quantification

As stated above, the interface should be easy to understand and to navigate the website to utilize all aspects of the system. A user manual will be provided to help users understand the system and to be able to use it. The user manual will also provide some deeper insights to the system that can help with future development and system growth.

Architectural Design and Patterns

System Design

As stated earlier the system has a Layered Architecture. The Presentation Layer communicates with the Business/Application Layer that in turn makes requests to the Domain Layer that lastly reads and writes to the Database Layer where the database is located. As the system is Database-Centric, most of the processes interact with the database either in the form of reads or writes. Requests from the client side invoke the backend that communicates with the database, that then provides information back to the user. Lastly the system is Domain-Driven in Design too, meaning the models and functions are set up in a manner that the client has specified and will understand. Located below is a diagram of the system:



Design Patterns

Factory Pattern

For creating the UI components, a factory pattern is used where the rendering logic is abstracted into a factory that creates different types of users based on their role in the organization. This can be seen when viewing the tree graph.

Decorator Pattern

This pattern could be used for adding functionality or styles to the different types of nodes in the UI, such as adding different icons or styling for CEOs, managers, and employees.

Singleton Pattern

Singletons are used when a maven dependency is imported to the system. Creating it as a singleton ensures that only a single instance of the dependency is created and is the only instance of it referenced and used in the system. All dependencies are located in the pom.xml file in the Backend directory of the repository.

Builder Pattern

The builder pattern is used when data is written to the database as well as read from the database. The system is composed of models that the builder uses to create instances of that model to be used with the data retrieved or the data being written to the database.

Composite Pattern

In the context of rendering the hierarchical structure, the Composite pattern is applied in the frontend to create a tree-like structure. Each user node acts as a component that can have child nodes (representing managers and employees).

Architectural Constraints

In any project there will be constraints to the system that would need to be accounted for or solutions to be made to mitigate these constraints. In the Employee Manager system the main constraints have appeared due to the requirements laid out by the client as follows:

- Cloud based
- Remote Data storage

Cloud Based

One of the project requirements are that the system should be hosted and deployed to a cloud based server so that the system can be accessible from anywhere.

Remote Data storage

Another requirement was that all data should be stored in a remote database, this means no data should be mocked or ‘hardcoded’ within the system.

Technology Choices

Below are the technologies that I used in the development of the Employee Manager system.

Frontend

ReactJS

React is a cross platform development tool that uses Javascript, HTML and CSS. With large amounts of experience in these languages, React seemed like a very good option for a frontend framework. It is also a very popular web development language so future development of the system would be seamless. A Pro for React is that it runs smoothly on the user side due its use of native components, however a Con is that it connects to backend components via a bridge and that can cause it to run slightly slower. The Bootstrap framework was used within HTML to help style and develop the components that are being displayed on the webpage.

Backend

Spring Boot

Spring Boot is developed in Java and makes developing web applications and microservices fast and easy through autoconfiguration, an opinionated approach to configuration and the ability to create standalone applications. It also allows for dependency injections that lets objects define their own dependencies that Spring will later inject into them to use their service. As it is coded in Java it was the top choice as I have extensively worked in java in recent years. Spring Boot also integrates nicely with React and the infrastructure between the tools is easy to set up and understand.

Database

MySQL

MySQL is a mature and widely-used relational database management system (RDBMS). It is well-suited for projects requiring structured data and relationships. In your case, managing user hierarchies where each user has a direct link (such as manager ID) to another user (their manager) aligns well with the relational nature of MySQL.

Hosting

AWS

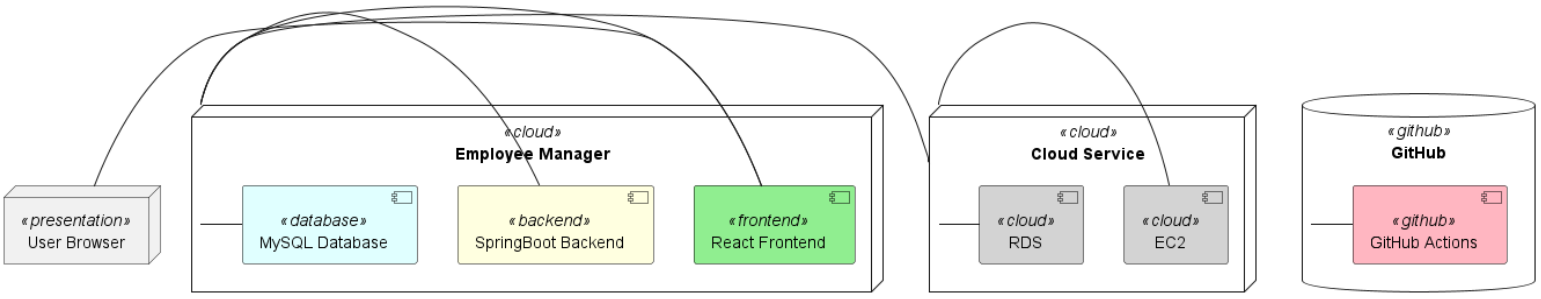
Amazon Web Services (AWS) is the world's largest cloud computing server provider. It is self hosting and provides many flexible solutions that can fit any customer’s needs. It provides reliable APIs to optimize your infrastructure as you see fit. It is also cheaper than Firebase and provides a pay-as-you-go option for the amount of space and storage you use. With the flexibility, performance and availability that EC2 and RDS offers, AWS was a no-brainer when it came to choosing hosting services.

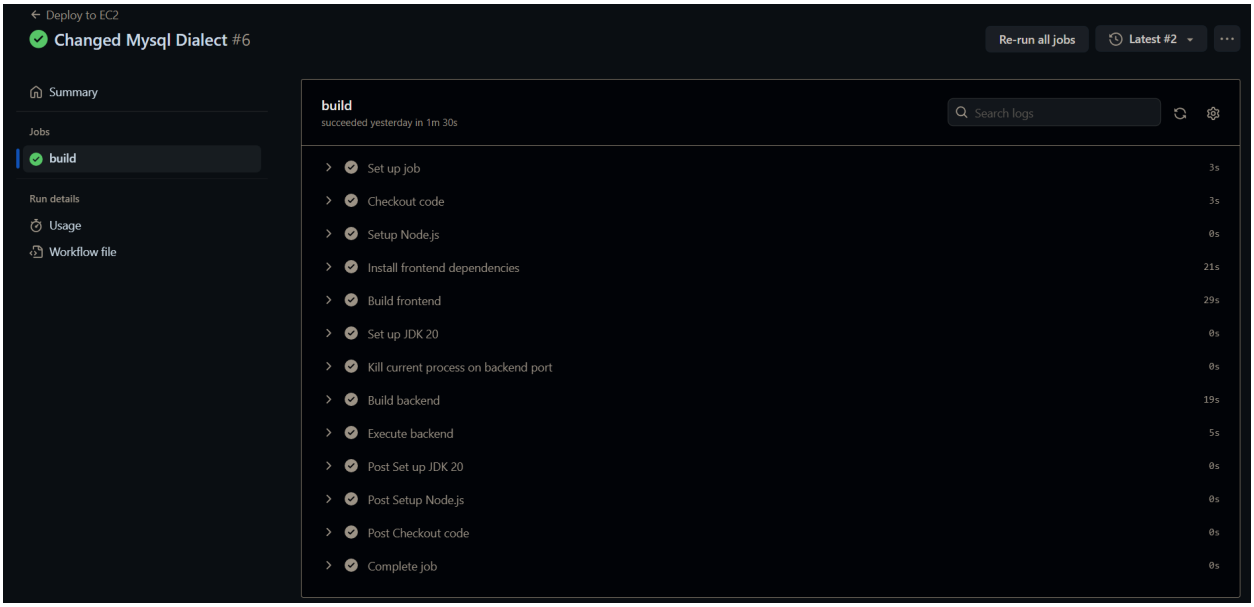
Deployment Model

The build and deployment process for both the frontend and backend projects of this system is fully automated by using the Github Actions service.

The React Frontend and Springboot Java backend projects are being pulled onto an Ubuntu-based AWS EC2 server via a GitHub self-hosted runner. Each time there are any new commits and code pushes to the repository the EC2 server will fetch the updated project and load it onto the server. After this happens the CI-CD deployment script takes over and installs all the necessary modules, dependencies and drivers as well as building and executing both the frontend and backend on the server. The database is constantly being hosted on AWS RDS and doesnt need any deployment or initialization.

Below is a detailed diagram that explains how the deployment works as well as a few screenshots of the deployment script executing on Github Actions.





Testing

For this project I used the React Frontend test(Jest Framework) for all of the frontend unit tests. I created a few tests to test some of the important generated frontend components.

All unit tests related to this project are automated and configured within the projects CI/CD pipeline. Tests execute each time code is being pushed to the remote repository.

Below is a few screenshots showing the tests as well the the automation of the tests.