Technical Document

Contents

Overview of System Architecture 2
Presentation Layer (Frontend) 2
Layout Components: 2
Page Components 2
Reuseable Components: 2
Functional Components: 3
Server Communication Components: 3
Validation Components: 3
Application Layer (Backend) 3
GET Endpoints: 3
POST Endpoints: 3
Data Layer (Database) 4
Tables 4
ER Diagram of database 5
Class Architecture Diagram 6
Technologies Used 7
Frontend 7
Backend 9
Database 9
Design Patterns 10
Layered Architecture Design Pattern 10
Component Based Design Pattern 10
REST API Design Pattern 10
Facade Design Pattern 10

Overview of System Architecture

The system uses a Layered Architecture, which allows distinct layers to have their own responsibilities, concerns, scalability and maintainability. Below is an overview of each layer:

Presentation Layer (Frontend)

The presentation layer provides a UI for the user to navigate and use the web application to edit the database.

Layout Components:

Components that are used for the layout of the web application.

- **NavBar**: Contains the navigation bar with controls to traverse between pages
- **Cascading Style Sheets**: Contains the styling for the HTML elements.

Page Components

Different pages of the web application.

- **App:** Provides routing between all the different web pages if the user has a logged in token, otherwise routes the user to the Login page.
- **Login:** Provides the login page to the user. The user can log in using credentials that are stored in the database, if the credentials are correct, the user gets a token that is stored in localstorage to enable them to remain logged in.
- **EmployeeTableView:** Displays the employee details in a table and hierarchical tree and allows the user to edit the employee details.
- **EmployeeCreator:** Provides a page for the user to enter details of and create a new employee.
- **HomePage:** Provides a landing page for the user to select which page they would like to traverse to.
- **RoleEditor:** Provides a page for the user to edit, delete, and create new business roles for the organization.

Reuseable Components:

Components that contain HTML elements that are reused or have the ability to be reused.

- **EmployeeEditingForm:** Allows the user to edit/enter details of an employee, used on both EmployeeTableView and EmployeeCreator.
- **EmployeeTable:** Provides a table to view employee data.
- **EmployeeTree:** Provides a hierarchical tree view of the organization's employees.

Functional Components:

Components that handle specific functions not related to other component headings.

- **EmployeeTreeRootCreator:** Formats the employee data into the correct format for the EmployeeTree.
- **useToken:** Used to determine if the user has a log in token in localstorage or to set the token.

Server Communication Components:

Components that communicate with the backend server.

- **getData:** Used to get data from the database using get requests.
- **sendData:** Used to send data to the database using post requests.

Validation Components:

Components used to validate data that the user enters.

- **employeeValidation:** Ensures that employee details are valid.
- **roleValidation:** Ensures that a role's details are valid.

Application Layer (Backend)

The backend layer serves as a RESTful API for the frontend to allow the frontend to make HTTPS requests to the backend related to the database and other functions.

GET Endpoints:

The frontend uses these requests to request that the backend server get and return the data in a JSON format from the database.

- **/get/roles:** Fetches the roles from the database and then returns them to the frontend.
- **/get/employees**: Fetches the employees from the database and then returns them to the frontend.
- **/login:** Determines whether a user account exists for the user trying to log in, returns a token if the user is valid allowing them to access the web application.

POST Endpoints:

The frontend sends post requests containing details for the request in the body of the request in a JSON format.

- **/api/employee/edit/submit**: Details for an edit of the employees table of the database is received, these changes are then implemented on the table in the database.

- **/api/employee/delete**: Details for the deletion of an item in the employees table is received, this deletion is then implemented on the table in the database.
- **/api/employee/create/submit:** Details for a new item for the employees table is received, this item is then inserted into table on the database.
- **/api/role/edit/submit:** Details for an edit of the roles table of the database is received, these changes are then implemented on the table in the database.
- **/api/role/create/submit:** Details for a new item for the roles table is received, this item is then inserted into the table on the database.
- **/api/role/delete/submit:** Details for the deletion of an item in the roles table is received, this deletion is then implemented on the table in the database.

Data Layer (Database)

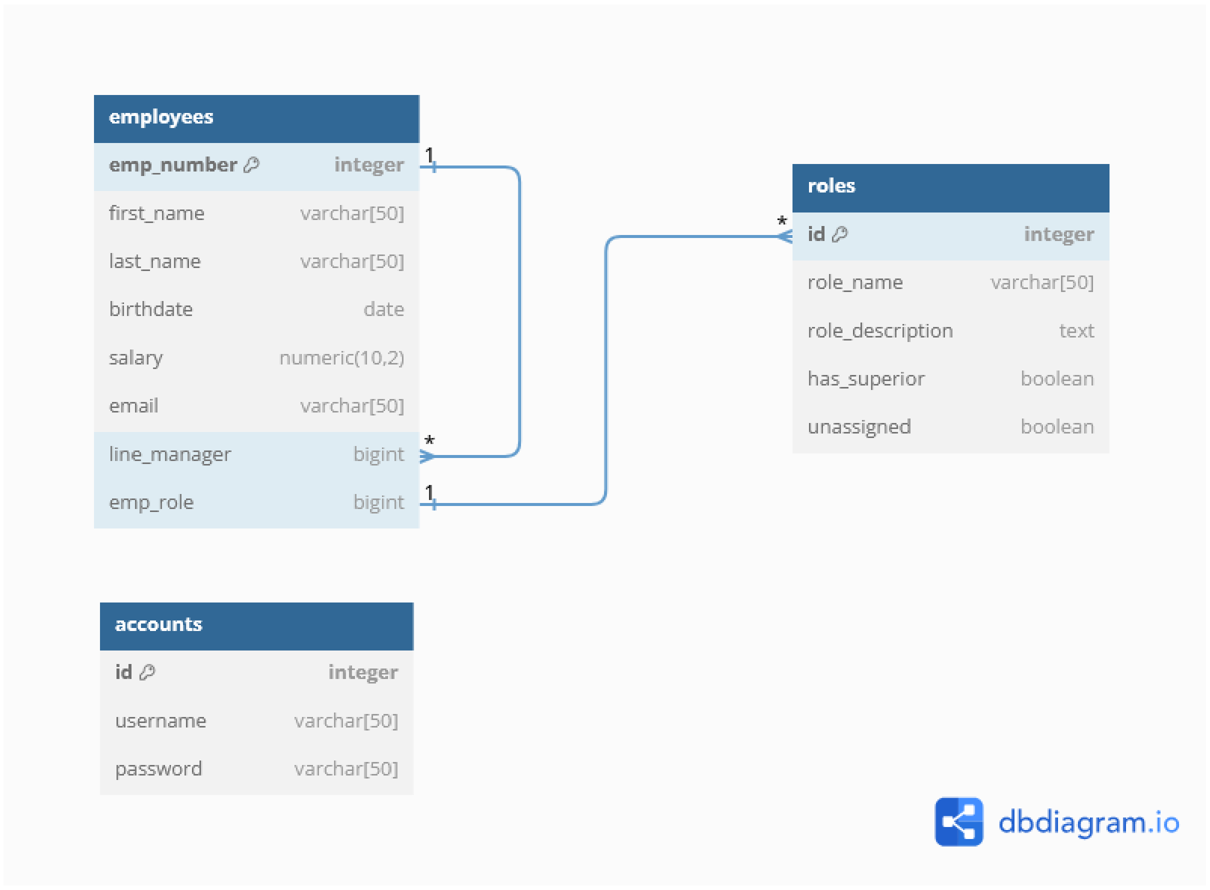
The data layer contains the database hosted on Amazon RDS.

Tables

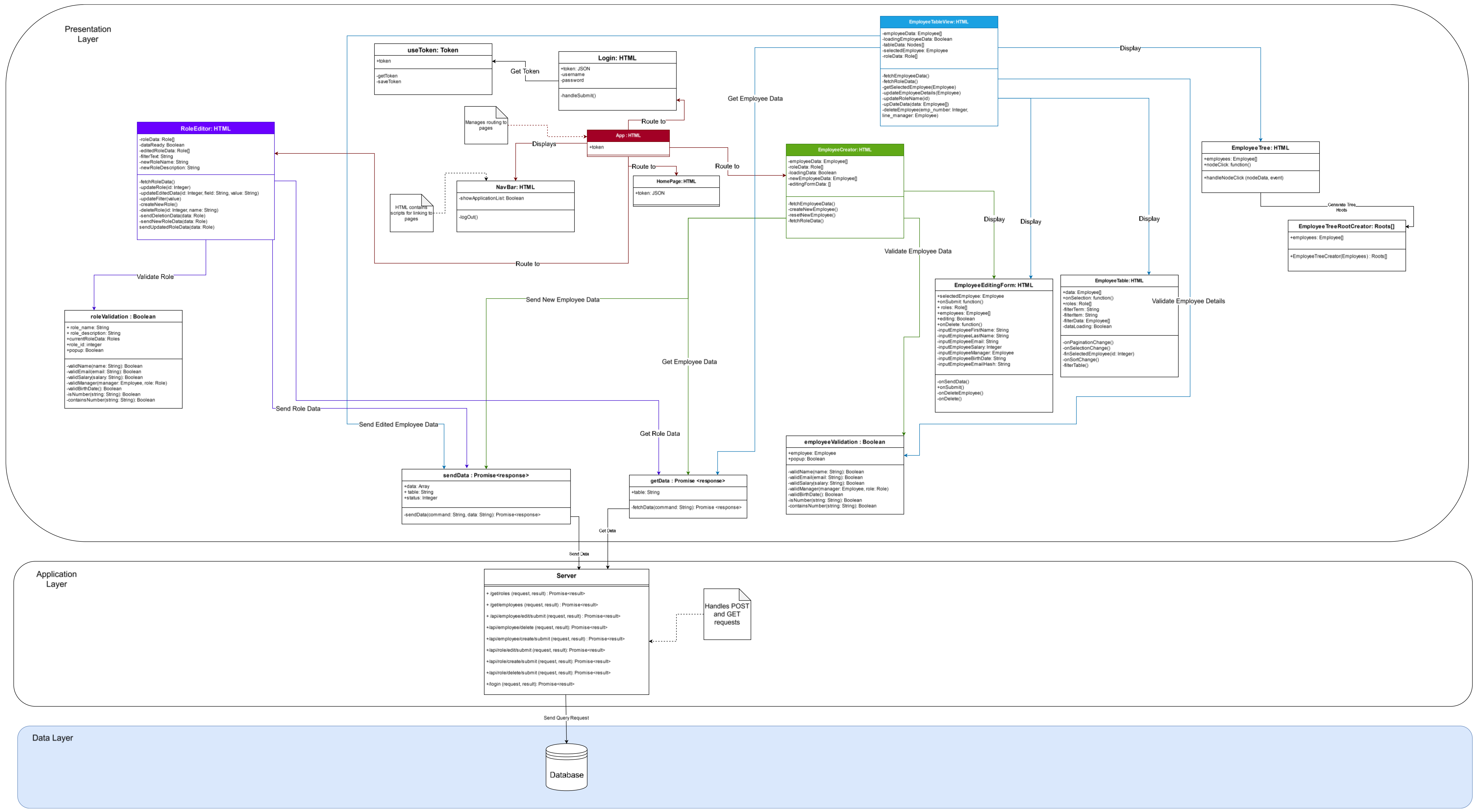
- **Employees:** Contains details about the employees of the organization. Column details:
  - **emp_number:** Primary key of the table, contains the employee's employee number.
  - **first_name:** Contains the first name of the employee.
  - **last_name:** Contains the last name of the employee.
  - **birthdate:** Contains the birthdate of the employee.
  - **salary:** Contains the salary of the employee.
  - **emp_role:** Contains the id of the role that the employee is assigned to.
  - **line_manager:** Contains the employee number of the line manager of the employee.
  - **email:** Contains the email address of the employee.
- **Roles:** Contains details about employee roles. Column details:
  - **id:** Primary key of the table, contains the id of the role.
  - **role_name:** Contains the name of the role.
  - **role_description:** Contains a short description of the role.
  - **has_superior:** Boolean to set whether an employee with the role requires a line manager.
  - **unassigned:** Special Boolean value, it is set to true for the unassigned role which employees without a role are assigned to.
- **Accounts:** Contains account details for logging into the web application. Column details:
  - **username:** The username of the account.
  - **userpassword:** The password of the account.

ER Diagram of database

A diagram outlining the relationship between the tables of the database.



Class Architecture Diagram


Presentation Layer
useToken: Token
+token
-getToken
-saveToken
Get Token
Login: HTML
+token: JSON
-username
-password
-handleSubmit()
Manages routing to pages
Route to
App : HTML
+token
Displays
HTML contains scripts for linking to pages
NavBar: HTML
-showApplicationList: Boolean
-logOut()
Route to
HomePage: HTML
+token: JSON
Route to
RoleEditor: HTML
-roleData: Role[]
-dataReady: Boolean
-editedRoleData: Role[]
-filterText: String
-newRoleName: String
-newRoleDescription: String
-fetchRoleData()
-updateRole(id: Integer)
-updateEditedData(id: Integer, field: String, value: String)
-updateFilter(value)
-createNewRole()
-deleteRole(id: Integer, name: String)
-sendDeletionData(data: Role)
-sendNewRoleData(data: Role)
sendUpdatedRoleData(data: Role)
Route to
Get Employee Data
EmployeeTableView: HTML
-employeeData: Employee[]
-loadingEmployeeData: Boolean
-tableData: Nodes[]
-selectedEmployee: Employee
-roleData: Role[]
-fetchEmployeeData()
-fetchRoleData()
-getSelectedEmployee(Employee)
-updateEmployeeDetails(Employee)
-updateRoleName(id)
-upDateData(data: Employee[])
-deleteEmployee(emp_number: Integer, line_manager: Employee)
Display
EmployeeCreator: HTML
-employeeData: Employee[]
-roleData: Role[]
-loadingData: Boolean
-newEmployeeData: Employee[]
-editingFormData: []
-fetchEmployeeData()
-createNewEmployee()
-resetNewEmployee()
-fetchRoleData()
EmployeeTree: HTML
+employees: Employee[]
+nodeClick: function()
+handleNodeClick (nodeData, event)
Generate Tree Roots
EmployeeTreeRootCreator: Roots[]
+employees: Employee[]
+EmployeeTreeCreator(Employees) : Roots[]
Display
Display
Display
Validate Employee Data
Validate Role
roleValidation : Boolean
+ role_name: String
+ role_description: String
+currentRoleData: Roles
+role_id: integer
+popup: Boolean
-validName(name: String): Boolean
-validEmail(email: String): Boolean
-validSalary(salary: String): Boolean
-validManager(manager: Employee, role: Role)
-validBirthDate(): Boolean
-isNumber(string: String): Boolean
-containsNumber(string: String): Boolean
Send New Employee Data
EmployeeEditingForm: HTML
+selectedEmployee: Employee
+onSubmit: function()
+ roles: Role[]
+employees: Employee[]
+editing: Boolean
+onDelete: function()
-inputEmployeeFirstName: String
-inputEmployeeLastName: String
-inputEmployeeEmail: String
-inputEmployeeSalary: Integer
-inputEmployeeManager: Employee
-inputEmployeeBirthDate: String
-inputEmployeeEmailHash: String
-onSendData()
+onSubmit()
-onDeleteEmployee()
-onDelete()
EmployeeTable: HTML
+data: Employee[]
+onSelection: function()
+roles: Role[]
-filterTerm: String
-filterItem: String
-filterData: Employee[]
-dataLoading: Boolean
-onPaginationChange()
-onSelectionChange()
-finSelectedEmployee(id: Integer)
-onSortChange()
-filterTable()
Validate Employee Details
Get Employee Data
Send Role Data
Send Edited Employee Data
Get Role Data
employeeValidation : Boolean
+employee: Employee
+popup: Boolean
-validName(name: String): Boolean
-validEmail(email: String): Boolean
-validSalary(salary: String): Boolean
-validManager(manager: Employee, role: Role)
-validBirthDate(): Boolean
-isNumber(string: String): Boolean
-containsNumber(string: String): Boolean
sendData : Promise<response>
+data: Array
+ table: String
+status: Integer
-sendData(command: String, data: String): Promise<response>
getData : Promise <response>
+table: String
-fetchData(command: String): Promise <response>
Get Data
Send Data
Application Layer
Server
+ /get/roles (request, result) : Promise<result>
+ /get/employees (request, result) : Promise<result>
+ /api/employee/edit/submit (request, result) : Promise<result>
+/api/employee/delete (request, result): Promise<result>
+/api/employee/create/submit (request, result) : Promise<result>
+/api/role/edit/submit (request, result): Promise<result>
+/api/role/create/submit (request, result): Promise<result>
+/api/role/delete/submit (request, result): Promise<result>
+/login (request, result): Promise<result>
Handles POST and GET requests
Send Query Request
Data Layer
Database

Technologies Used

Frontend

- **Framework**: React.js
    - **Description**: React.js is a popular framework for developing static web applications.
    - **Reason for usage**: React.js was used as it is a popular framework with a lot of documentation and libraries which aided with development. React.js projects are designed to work as a web application which suits the needs of the project.
- **Deployment Platform**: Amazon Amplify
    - **Description**: Amazon service for hosting static web applications or websites.
    - **Reason for usage**: Amazon Amplify allows for quick setup of a compiled web application created with React.js. The quick setup allows for easy and fast deployments of updated versions of the web application if required. Amazon Amplify creates a secure HTTPS address to the web application which can be accessed from anywhere which is required for this project.
- **APIs:**
    - **Gravatar**
        - **Description:** Allows for the linking of a profile picture to an email.
        - **Reason for usage:** Used for getting employee profile pictures from Gravatar if they have a Gravatar account linked to their email.
- **React.js libraries used:**
    - **React-bootstrap:**
        - **Description:** React-bootstrap is a rebuild of the *Bootstrap* framework which is a popular frontend framework. It allows for easy creation of components such as Buttons, Spinners, Offcanvas items, etc.
        - **Reason for usage:** React-bootstrap was used as it allowed for quick creation of nice-looking UI elements such as buttons it was also used for an Offcanvas for the application selection from the navigation bar of the application.
    - **React-axios:**
        - **Description:** React-axios allows for asynchronous requests to a specified address.
        - **Reason for usage:** React-axios allowed for asynchronous requests to the backend for retrieving or sending data.

- **React-router:**
    - **Description:** A routing library that allows giving a component a URL and allows routing between the components to allow for multiple web pages to be used.
    - **Reason for usage:** Allowed for multiple webpages to be used and allowed for navigation between them. This allowed the creation of the multiple pages of the web-application.
- **React-d3-tree:**
    - **Description:** A component that allows the representation of data as a hierarchical tree graph.
    - **Reason for usage:** Used for the creation of the employee hierarchy. It also allowed a lot of customization to display the tree graph how I wanted it to look.
- **CryptoJs:**
    - **Description:** JavaScript library providing crypto standards.
    - **Reason for usage:** Used to generate the SHA256 Hash from an employee email to get their gravatar profile picture.
- **React-Select:**
    - **Description:** Allows for the creation of advanced selection boxes with features such as searching within the selection box, multi-selection, and custom styles.
    - **Reason for usage:** Allowed for the creation of searchable selection boxes when selecting an employee's line manager.
- **React-table-library**
    - **Description:** A library that can be used to create customizable tables in React.js. Allows for filtering, pagination, and sorting.
    - **Reason for usage:** Used to create a table of employees and allowed for the implementation of sorting, filtering, and pagination that was customized to suit the needs of the project.

Backend

- **Environment:** Node.js
    - **Description:** Node.js is a JavaScript runtime environment that runs on the V8 JavaScript engine and executes JavaScript code outside of a web browser.
    - **Reason for usage:** Node.js is a popular environment for JavaScript backends and allows for the frontend to make requests for data from the database as well as send data to the backend to edit data in the database.
- **Web Framework:** Express.js
    - **Description:** A backend web application framework for building RESTful APIs with Node.js.
    - **Reason for usage:** Allows for the backend to make HTTP/HTTPS requests to the backend to get data from the backend or to send data to the backend. The backend can then manage the requests and perform server-side actions.
- **Deployment Platform:** Railway
    - **Description:** Railway is a platform that allows for the deployment of web or cloud applications.
    - **Reason for usage:** Allows for quick deployment of the backend server and provides a secure address using HTTPS for the frontend to make requests to.

Database

- **Database:** PostgreSQL database
    - **Deployment:** Amazon RDS
    - **Reason for usage:** Amazon RDS allowed for the creation and hosting of a secure access-controlled PostgreSQL database which can be accessed by the backend.

Design Patterns

Layered Architecture Design Pattern

The system was created using a layered architecture separating the components into the presentation, application, and data layer. This allows for isolation of concerns, scalability, maintainability and responsibilities.

Component Based Design Pattern

Different elements have been turned into reusable components, such as the editing form, the navbar, and the employee table and employee tree graph. All of these components have the possibility or have been reused in the system.

REST API Design Pattern

The system uses a RESTful API for sending and receiving data between the frontend and the backend.

Facade Design Pattern

The getData and sendData components allow the frontend to easily make requests to the backend without having to worry about the underlying subsystem for sending and receiving data from the backend, instead this is all handled by the getData and sendData components.