CS673 Software Engineering
Team 3 - HRMasery
Software Design Document

Your project Logo
here if any

Team Member	Role(s)	Signature Date
Jianle Xie	Team Leader	*Jianle Xie* 9/9/2023
Yuxuan Wang	QA leader	*Yuxuan Wang* 9/9/2023
Abigya Devkota	Design and Implementation Leader	*Abigya Devkota* 9/9/2023
Jingjing Tang	Configuration Leader	*Jingjing Tang* 9/10/2023
HaoLun Li	Requirement Leader	*HaoLun Li* 9/11/2023
WeiHao Mai	Security Leader	*Weihao Mai* 9/11/2023
Zhiwei Lin	Backup Leader	*Zhiwei Lin* 9/11/2023

Revision history

Version	Author	Date	Change
1	Abigya Devkota	09/22/2023	Provided detailed description of software architecture.
2	Yuxuan Wang	09/24/2023	Changed structure and scope of project functionalities
3	Jianle Xie	09/24/2023	finalizes edits and update OverView
4	Jianle Xie	10/09/2023	Consolidate SecurityDesign

● Introduction

In this section, give an overview of this document, and also address the design goals of your software system.
<an overview of what this document is about>

We are determined to create an employee management system (HRMasery) using various tech stacks. HRMasery will be built with Java Spring Boot as it's backend server, React as it's frontend user interface (UI), and H2 as it's database component. HRMasery will utilize CRUD (create, read, update, delete) operations as its primary functionalities. In its simplest version, HRMasery allows users to login and can view various buttons to perform its CRUD operations. The simple user interface will interact with the backend server and store data in a database. There will be 2 different kinds of users (staff and manager), thus we will implement authentication and access control in HRMasery.

The overall architecture of HRMasery is a MVC (model-view-controller) model with 3 tiers. The View consists of the UI, capable of accepting and sending requests to create, read, update and delete a record. The Controller will handle requests, both from the user and the database. The Model will store the user record/information.

The system is also a 3 tier architecture: client tier, application tier, and database tier. The client tier allows the client to access the application through the browser. The application tier is broken into 3 parts: presentation layer, business logic layer, and database access layer. The database tier stores all the records, and can make changes/updates to the records.

Our simplified version will have 7 entities with various attributes: Employee, Manager, Department, Role, UserAccount, AttendanceEntry, and LeaveRequest. Manager inherits from Employee, UserAccount has a composition relationship with Employee, Employee has an aggregation relationship with department, and Employee has association relationship with AttendanceEntry, LeaveRequest, and Role. The

database design will revolve around the class diagram. The relationships between each entity will be either one-to-many or many-to-one.

Our design pattern goal is to implement a repository pattern. The repository pattern separates the data access component from the business logic, thus decreasing code duplication and improving code maintainability. Our repository entities implement the JPA repository and take advantage of existing JPA methods. However, we will implement our own query if necessary.

The UI of the system is a simple linear structure. It would present a login page, home pages for both types of users, and different pages for different functionalities. Each homepage will feature the user's full name The managers home page will have the CRUD function implemented into as buttons. We will attempt to implement a search bar as well.

For our security, we will attempt to implement Spring Security with Java Web Token (JWT). It will be divided into 2 parts, generating JWT and validating JWT. User's username and password are validated against the database of user credentials, if valid, then a token will be generated. Only with such a token, can users bypass Spring Security. When the user requests a GetMapping, the token is sent along with the request. The user's information is retrieved from the system's database and validated separately with the token.If valid, then the GetMapping will return, otherwise an error will return. This will be the basic functionality of our security and more features will be added in the future.

● Software Architecture - Abigya

In this section, you will describe the decomposition of your software system, which includes each component (which may be in terms of package or folder) and the relationship between components. You shall have at least one diagram to show the

whole architecture of . The interface of each component and dependency between components should also be described. If any framework is used, it shall be defined here too.

HRMasery High Level Architecture

(ref: Bacikova)

HRMasery Functional View
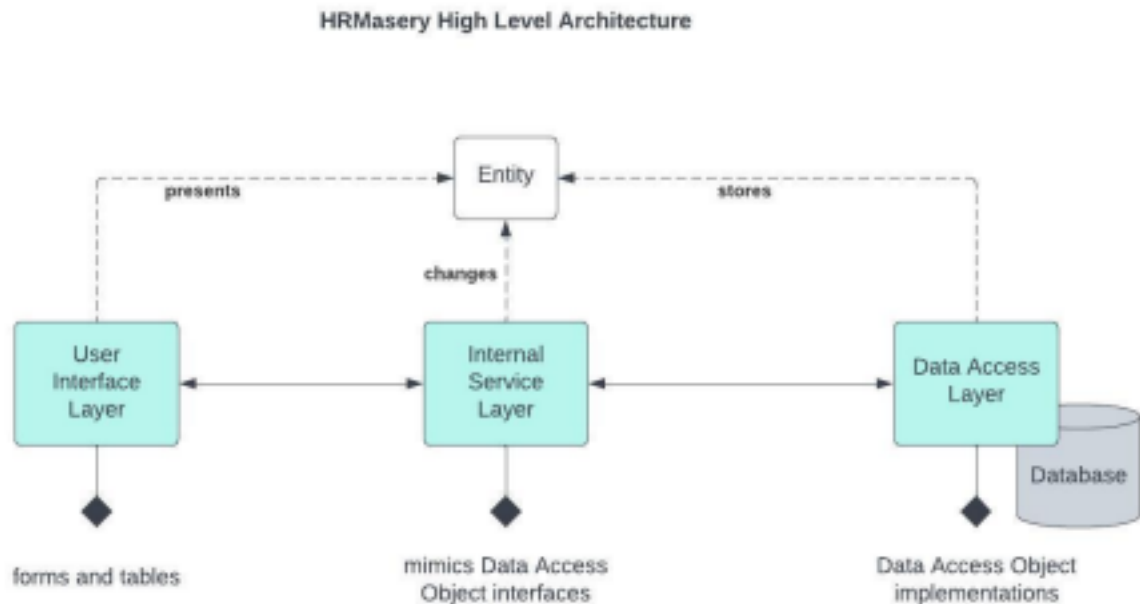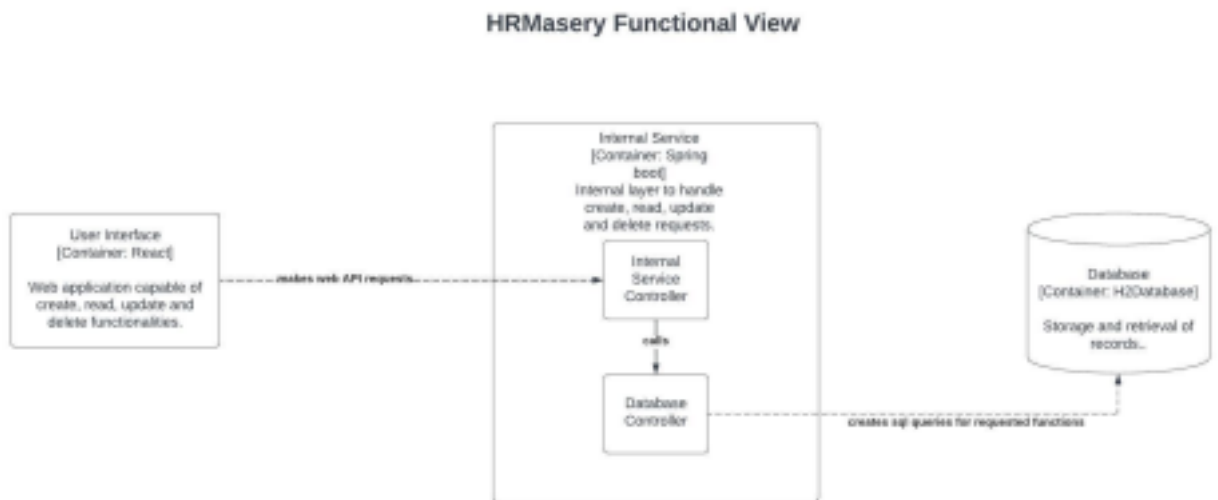
The components in this system are the user interface, the backend api and the database. The user interface provides a seamless experience for users to interact with the application. The backend follows the Model View Controller design pattern to handle web api requests and also connects to the database to store and retrieve records.

The user interface built using React.js will handle client requests including adding a new user, updating an existing user, deleting an existing user and viewing all current users. To effectively handle client requests, the user interface will call the internal service using web api

requests such as post, pull and delete. The internal service built using java will use two controllers : one to handle requests from the user interface and one to interact with the database. Ultimately, any request from the user interface will first go through the Internal Service Controller for validation and processing and result in the appropriate query for the database.

Software Architecture:

Utilizes a MVC model:
- **User(view)** - User interface capable of accepting and sending requests to create, read, update and delete a record.
- **Controller (handled requests)** - Internal Service will have two controllers - one to handle requests from the UI and another to interact with the database. Mappings from the UI to handled:
  Post mapping: handles creating a new record
  Put mapping: handles updating an existing record
  Delete mapping: handles deleting an existing record
  Get mapping: handles getting existing records (may be with or without a condition) ○
  Model (handled database) - stores the most accurate representation of the user record i.e source of truth for user data

This application will implement closed layered architecture where the user interface will only interact with the internal service and the internal service will explicitly interact with the database. This is to maintain separation of logic and ensure some security when handling requests from an external user. Any user request will be properly validated by the internal service before making changes to the database which ensures less intervention and provides more opportunities to strengthen each layer.

The different tier of the software:
- Client tier (the browser) -> The client will access the application through the user interface using a web browser. At this time, only the web application of HRMasery is supported.

- The application tier is divided into 3 layer:
  - Presentation layer: This will be the user interface built with React.js. It will be a simple application with regards to the embellishment but will consist of all essential functions related to the project.
  - Business logic layer: This will be a Java application built using Spring Boot and maven. It will utilize the ease and versatility of Spring Boot framework to manage code effectively and reduce critical defects and code smells.
  - Data access layer: This will be the H2 database that will store the user data. It will integrate simple database queries such as SELECT, DELETE, ALTER AND CREATE. The queries will align with the web api requests provided by the presentation layer.

- Data store tier
  - Database system : The database will utilize springboot's H2 console to store and alter the record created by a user. This is a relational database which will utilize a schema that represents the data that defines a client of HRMasery. It will also

have a primary key that uniquely identifies each new user and provides an accurate point of access to their information which will be carefully handled by the internal service and accurately displayed by the user interface.

● Class Diagram-Jingjing

In this section, you will provide a detailed description of each component (or package) and use one or multiple class diagrams to show the main classes and their relationships in each component.

I propose a basic class diagram based on common features and attributes present in standard HR systems. This diagram contains 7 entities and I add some basic attributes and operations of each entity.

Relationships identify:

1. Inheritance:

Manager to Employee Relationship: Manager is a subclass of Employee, so this relationship represents inheritance. The Manager class inherits attributes and operations from the Employee class.

2. Composition:

UserAccount to Employee Relationship: If we assume that a UserAccount cannot exist without an associated Employee (meaning if you delete an Employee, their UserAccount is also deleted), then this relationship is a composition.

3. Aggregation:

Department to Employee Relationship: A Department can have multiple Employees, but even if a Department ceases to exist, the Employees can still exist. This relationship can be seen as aggregation

4. Association:

Employee to AttendanceEntry Relationship: Employees have associated AttendanceEntries, but these entries might not be strictly dependent on the Employee in a compositional sense.

Employee to LeaveRequest Relationship: Similarly, Employees create LeaveRequests, but the requests might not be strictly dependent on the Employee. This is also an association.

Employee to Role Relationship: Employees can have roles, and roles can be assigned to employees. This many-to-many relationship can be seen as an association.

Role

-roleID: Int
-roleName: String
-description: String

+createRole()
+updateRoleDetails()
+deleteRole()
+getRoleByID()
+listAllRoles()
+assignRoleToEmployee()
+removeRoleFromEmployee()

Department

-departmentID: Int
-departmentName: String
-budget: Float

+createDepartment()
+updateDepartmentDetails()
+deleteDepartment()

UserAccount

-userID: Int
-username: String
-password: String
-lastLoginDate: Datetime

+createRole()
+updateRoleDetails()
+deleteRole()
+getRoleByID()
+listAllRoles()
+assignRoleToEmployee()
+removeRoleFromEmployee()

Employee

-id: Int
-name: String
-age: Int
-email: String
-phoneNumber: String
-address: String
-jobTitle
-salary: Float

+addEmployee()
+updateEmployeeDetails()
+deleteEmployee()
+getEmployeeByID()
+listAllEmployees()
+searchEmployeeByName()

Manager

-numberOfReports: Int

+listDirectReports()
+assignEmployeeToManager()
+removeEmployeeFromManager()

AttendanceEntry

-entryID: Int
-clockInTime: Datetime
-clockOutTime: Datetime
-date: Datetime

+clockIn()
+clockOut()
+getAttendanceByDate()
+getAttendanceByEmployeeID()
+updateAttendanceEntry()
+deleteAttendanceEntry()

LeaveRequest

-requestID: Int
-startDate: Datetime
-endDate: Datetime
-leaveType: String
-status: String

+createLeaveRequest()
+updateLeaveRequestStatus()
+getLeaveRequestByID()
+listAllLeaveRequests()
+deleteLeaveRequest()

Relationships:
- Role — assign to — Employee (0..n / 0..m)
- Department — Has — Employee (1 / 0..n)
- Department — managed by — Manager (1 / 1)
- UserAccount — belongs to — Employee (1 / 1)
- Employee — report to — Manager (0..m / 0..n)
- Employee — belongs to — AttendanceEntry (1 / 0..n)
- Employee — create by — LeaveRequest (1 / 0..n)

● UI Design (if applicable) - Jianle

In this section, you can describe your UI design. You can include both your initial design before the implementation and the screenshots of your UI after the implementation.

Prior to implementation

Since our goal is a minimal viable product, we will design the UI with a linear structure, keeping everything simplified without any fillers and/or complications. The designs of these UI pages are subjected to change, but our goal is ultimately its functionality being unchanged.

<Login Page>

HRMasery would have a login screen that welcomes users to HRMasery. The login page would have a space to enter the user's username, coupled with a placeholder "Username". Below the Username, is a place to enter the user's password, also coupled with a placeholder "Password". Right below each entry box is an option for the user to recall their password/username in case the user can not remember them. Afterwards, the user can proceed to the next page by clicking on the "Login" button.


<Home Page>

Depending on what kind of account the user uses, the application will take the user to 2 different kinds of homepage. As an admin user, the homepage will feature a search bar on the top of the page, followed by a Sort and Create Role button. The admin user's first and last name will display across the page with their role underneath. The rest of the page will be dedicated to the current available roles. Each role will have a title and a description. Each role will have 4 icons next to their role title. These 4 icons represent update, delete, create staff, and delete staff respectively.

The admin can search all staff users by name via the search bar. Underneath the search bar is 2 options, Sort and Create Role. The Sort feature will sort all the different staff roles accordingly (i.e. by alphabetically order, most, and/or least staff counts).

<Update Role>

When the update icon is clicked, a popup window appears and the user will be able to update the role's description, and an update button finalizes the update while a 'x' icon on the upper right corner exits the operation unsaved. The delete icon will pop up an additional warning to the user. The popup will have the role that's to be deleted and asks the user if they're sure that they want to delete the role. The 'Confirm' button saves the deletion and the 'x' cancels the operation.

<Create Staff>
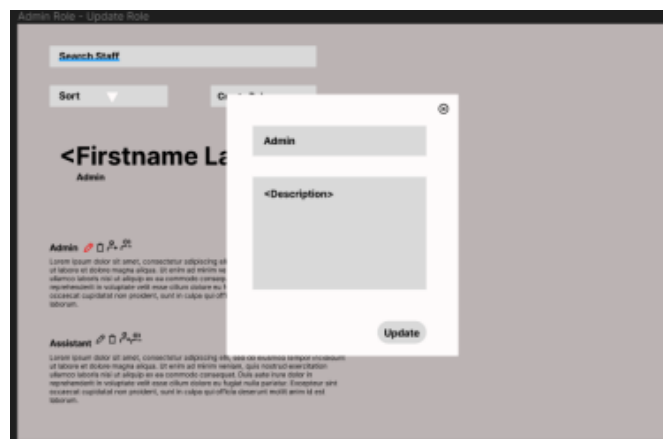
The create staff icon next to each role creates a new entry for that role. When clicked on, a window will appear with the role's title and will ask for the new entry's first name, last name, and address. A 'Create' button click will finalize the creation and a 'x' will exit the operation unsaved. When deleting a staff, the admin user will locate the role of the target staff first. Then the admin user will click on the delete staff icon. This will bring up a page with the role title on the top, a search bar to search for the target staff, and a list of each staff with first name, last name, and address. The admin user can click on the delete icon next to each staff's first name, and the application will prompt the admin user if they want to delete the target's role, first name, last name, and address. A 'Confirm' saves the operation and a 'x' cancels and exists the operation.

<Staff User Home Page>

If the user is a non-admin user, then a page with the user's first name and last name will display across their homepage with their role listed underneath their name. Underneath their role is their address with an update icon next to it. Their description is also written on the page. Non-admin users can update their address via the update button. This will prompt the user to enter their new address, and an update button finalizes it.

● Database Design (Haolun)

1. Employee:

   a) employeeID: Primary Key. Unique identifier for the employee.

   b) firstName: Text. Employee's first name.

   c) lastName: Text. Employee's last name.

   d) designation: Text. Position or role of the employee.

   e) contactInfo: Text. Contact details of the employee (email, phone, etc.).

   f) managerID: Foreign Key referencing Manager. Determines which manager supervises this employee.

   g) departmentID: Foreign Key referencing Department. Department the employee belongs to. h)

   roleID: Foreign Key referencing Role. The role of the employee in the organization. i)

   passwordHash: Text. Hashed version of the user's password.

2. Manager:

   a) managerID: Primary Key. This can also be a Foreign Key referencing Employee to capture the inheritance relationship.

   b) departmentID: Foreign Key referencing Department. The department that the manager oversees. 3.

AttendanceEntry:

   a) entryID: Primary Key. Unique identifier for each entry.

   b) employeeID: Foreign Key referencing Employee.

   c) clockInTime: Timestamp. Time the employee clocks in.

   d) clockOutTime: Timestamp. Time the employee clocks out.

4. LeaveRequest:

   a) requestID: Primary Key. Unique identifier for each leave request.

   b) employeeID: Foreign Key referencing Employee.

   c) startDate: Date. Start date of the leave.

   d) endDate: Date. End date of the leave.

   e) status: Enum ('Pending', 'Approved', 'Denied'). Status of the leave request.

   f) managerID: Foreign Key referencing Manager. Manager who approved or denied the request.
5. Role:

   a) roleID: Primary Key. Unique identifier for each role.

   b) roleName: Text. Name of the role (e.g., "Admin", "Developer", "Sales").

6. Department:

    a) departmentID: Primary Key. Unique identifier for each department.

    b) departmentName: Text. Name of the department.

    c) managerID: Foreign Key referencing Manager. Manager overseeing this department. 7.

UserAccount:

    a) userID: Primary Key. Unique identifier for each user.

    b) employeeID: Foreign Key referencing Employee.

    c) username: Text. Login username for the user.

Relationships:

1. Employee to Manager: Many-to-One. Many employees can be managed by one manager, represented by the managerID in the Employee table.
2. Employee to Department: Many-to-One. Many employees belong to one department, represented by the departmentID in the Employee table.
3. Employee to Role: Many-to-One. Many employees can have the same role, represented by the roleID in the Employee table.
4. Employee to AttendanceEntry: One-to-Many. One employee can have many attendance entries.
5. Employee to LeaveRequest: One-to-Many. One employee can make many leave requests.

Database ER diagram (crow's foot)
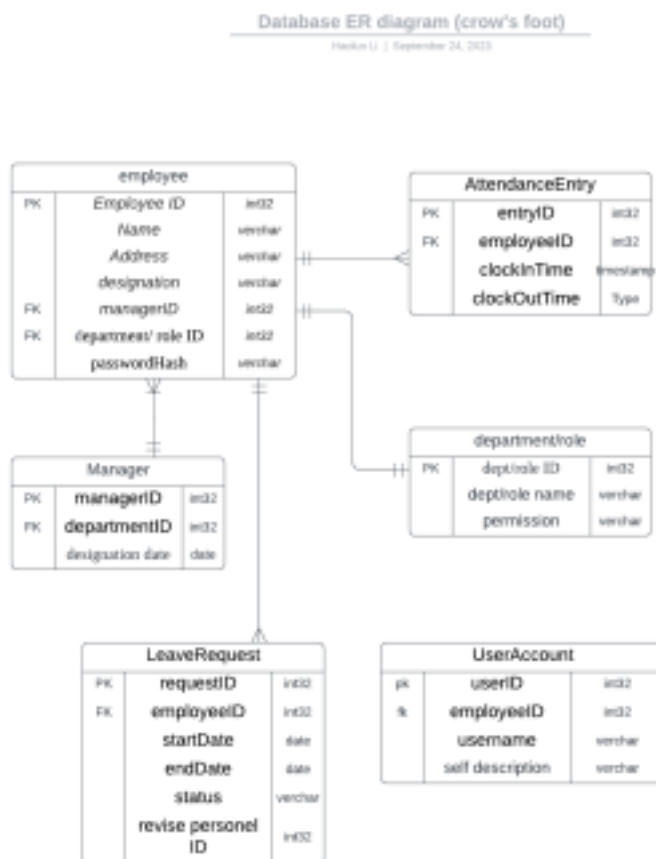
● Security Design(Weihao Mai)

In this section, you shall describe any security design in your software system.

We will be using the spring security dependency as our security for authentication and authorization. Instead of form based authentication that is provided by spring security, we will also incorporate Json Web Token (JWT) dependency as well. Our essential requirement will be authentication and authorization.

(1)
```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.5</version>
</dependency>

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.5</version>
</dependency>

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.11.5</version>
</dependency>
```

For authentication, when a user logins in with appropriate username and password, that information is passed from the frontend UI to the server as a PostMapping. If the username and the password exist in a database, then the server returns a JWT. That token is added on to every request from the client to the server, authenticating that the request came from a source that the server recognizes.

For authorization, we will implement a role based access control. The server can recognize what kind of role a user has upon login. Different roles will have access to certain methods in the server. An admin role can perform administrator functions vs. a user role can only view their own information.

Finally, our security will have logging capabilities and reset passwords. Logging functionality will record all user activities and system events. While reset password functionality will aid users to set a new password in the event that user can't recall their previous password.
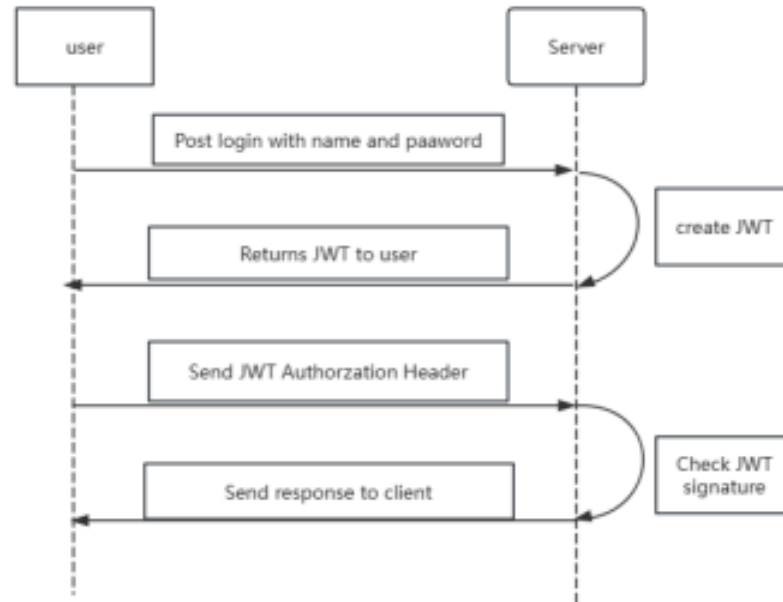
Authentication flow chart

● Business Logic and/or Key Algorithms (Zhiwei Lin) In this
section, you shall describe any key algorithms used in your software system, either in
terms of pseudocode or flowchart, or sequence diagrams.

As of current build, we our mvp shall implement only the basic CRUD operations, but in the future we plan to implement the following business logic.

HRMasery:

Database tables: the basic two tables we need:
   role
   user(Employee)
Because a user can have multiple roles and a role can contain multiple users, we need a third table to maintain the relationship between users and roles
     User —---------------- Role
      m —------------------ n
     employee_role

System management:
   (1). Role management:
           Basic CRUD methods to manipulate the database.
   (2). User management :
           1. Basic CRUD methods to manipulate the database.
           2. User condition paging query(We may need a VO class to encapsulate the
query criteria).

3. Assign roles to users.
 Graph illustration:

   (3). Menu management: Users with different roles have different menu and function permissions when logging in to the background management system. We need a data dictionary that shows hierarchical relationships. For example, there are "User management" and "Role management" under system management. And " User management" includes operation write , add, delete, modify, update and so on.
      1. Basic CRUD methods to manipulate the database.
      2. Assign menus to roles.
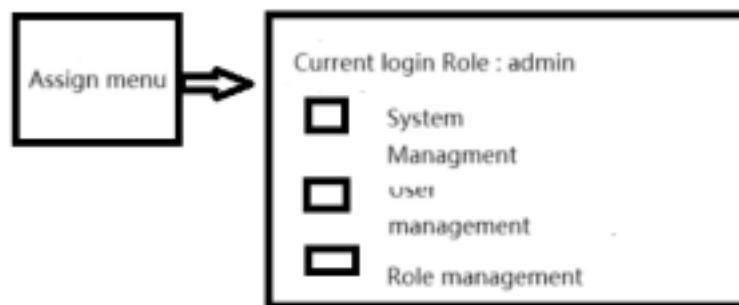
    3. Third data tables we need : sys_menu. We need two fields in this table, id and parent_id. Use id=parent_id to find the hierarchy. Note that parent_id=0 means the top-level data.
   Graph illustration:

ID	ParentID	Name
1	0	Sysstem Management

| ID	ParentIId	Name
2	1	User Management

ID	ParentId	Name
5	1	Role Management

    4. Fourth data tables we need: sys_role_menu: There are also many-to-many relationships, where one role can operate multiple menus and one menu can have multiple role operations.

● Design Patterns (Yuxuan Wang)

In this section, you shall describe any design patterns used in your software system.

We will use a repository pattern for the HRMasery project.It's the design pattern used in data access and database interaction.
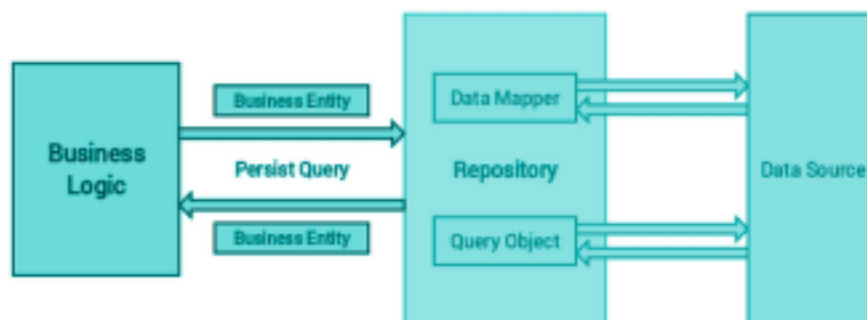
- ● Data Source: it can be a database or a web service
- ● Repository: It queries and maps the data from database to business logic. It centralizes the data logic or Web service access logic. It can be adapted as the overall design of the application evolves. It contains methods
- ● Business logic: It's the customer side. It can store business entities or query information from data sources by calling the methods in the Repository.

The repository pattern helps improve high code maintainability, enables easier testing of business and data access logic separately and reduces code duplication.

● Any Additional Topics you would like to include.

● References

○ UI:
- https://info.keylimeinteractive.com/what-is-linear-ux

○ Design Pattern:
- https://cubettech.com/resources/blog/introduction-to-repository-design-pattern/

○ Security:
- https://www.javainuse.com/spring/boot-jwt

○ Software Architecture
- Bacikova, MIchaela. "Multi-Tier Application Architecture Used by CRUDCOMP CRUD Applications ..." Research Gate, www.researchgate.net/figure/Multi-tier-application-architecture-used-by-CrudComp-CRUD-applications_fig1_286980769. Accessed 17 Sept. 2023.

● Glossary