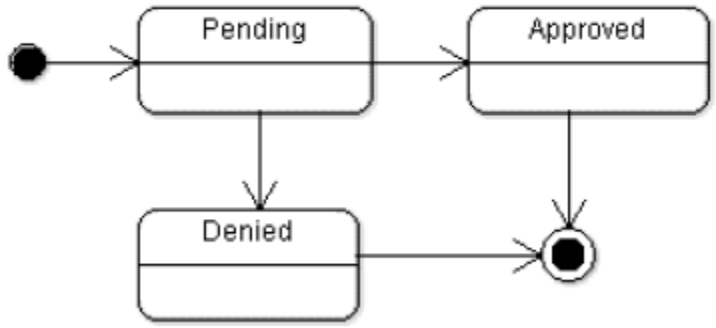
Project 1: Expense Reimbursement System

Executive Summary

The Expense Reimbursement System (ERS) will manage the process of reimbursing employees for expenses incurred while on company time. All employees in the company can login and submit requests for reimbursement and view their past tickets and pending requests. Finance managers can log in and view all reimbursement requests and past history for all employees in the company. Finance managers are authorized to approve and deny requests for expense reimbursement.

NOTE: the datatypes in the physical model are for OracleSQL, not PostgreSQL so you'll need to translate the datatypes to the reasonable equivalent in PostgreSQL.

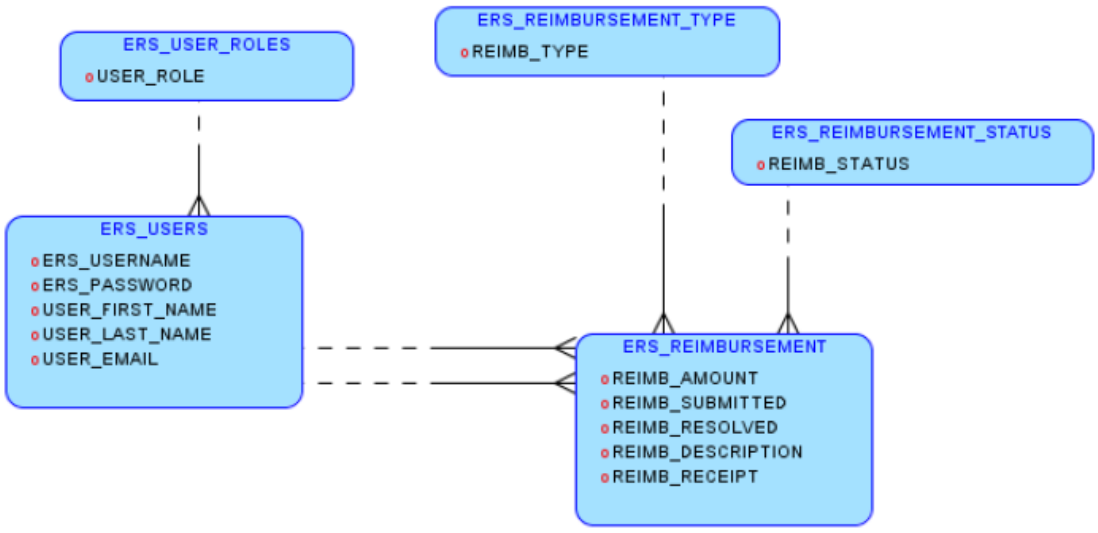
State-chart Diagram (Reimbursement Statuses)



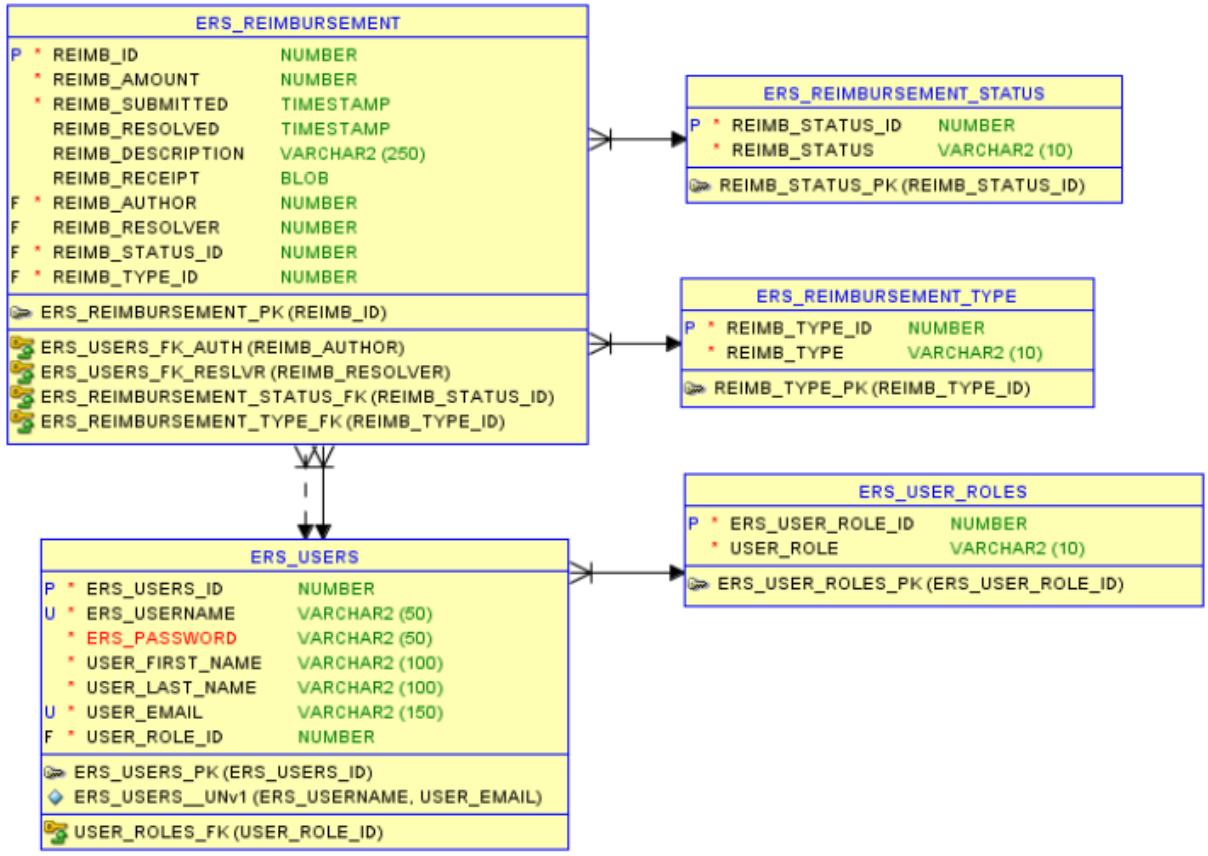
Reimbursement Types

Employees must select the type of reimbursement as: LODGING, TRAVEL, FOOD, or OTHER.

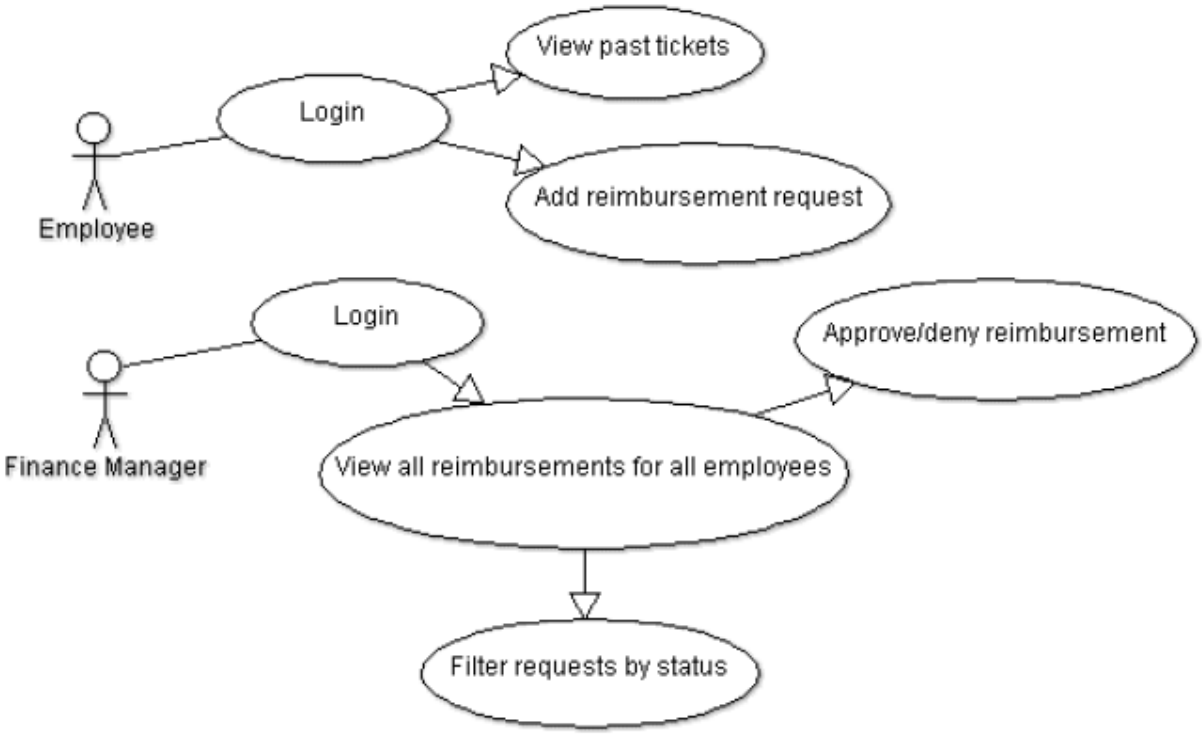
Logical Model



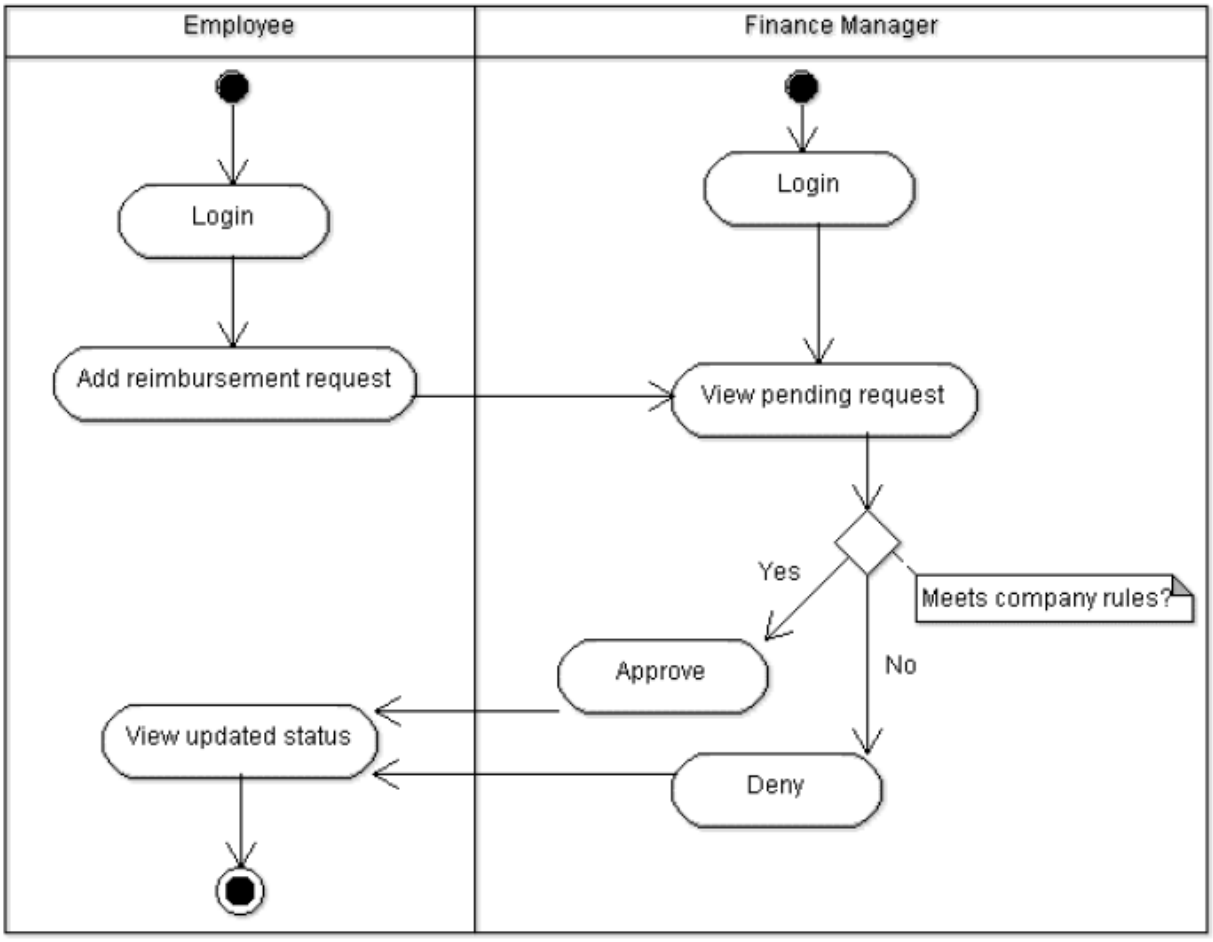
Physical Model



Use Case Diagram



Activity Diagram



Technical Requirements

- Use the concept of “Separation of concerns”. Organize your code to make it neat
- The application shall deploy onto a Server
- The back-end system shall use JDBC to connect to a PostgreSQL database
- Your API endpoints should be deployed using Websockets
- The front-end view should use HTML, CSS, and JavaScript to present the application to the end user.
- Your app should use AJAX calls to access your server-side functionality.
- The web pages should look presentable (try using css and bootstrap); I’d rather not see a website from 1995.
- Use JUnit to implement testing

- (Optional) Passwords should be hashed in Java and securely stored in the database
- (Optional) Users can upload a document or image of their receipt when submitting reimbursements
- (Optional) The application will send an email to employees letting them know that they have been registered as a new user, giving them their temporary password

**Please take the deadline seriously.

**Do NOT spend too much time stuck on a single blocker without asking a batch-mate for help.

Tips on how to start

If you're having trouble wrapping your head around how to start, here is my preference for starting project 1:

1. Start with your SQL schema. Create all your tables (entities) and entity relationships. You should be able to populate your entities with data before even touching javascript or html; attempt to run simple CRUD operations on each of your tables to verify that your SQL statements are firing.
    a. The "User Role", "Reimbursement Type", and "Reimbursement Status" tables are all LOOK UP TABLES (enum values). ATTENTION: the Database Administrator (you) will need to pre populate these look up tables with data before doing anything else; because they will have not null foreign keys pointing them.
2. Once your database is working properly, go to javascript and create your model layer (aka model files aka class files). So create the necessary object representations of your database tables.
    a. In our case we'll be creating classes for "User", "Reimbursement", "ReimbursementStatus", "ReimbursementType", and "UserRole"
    b. (you may find that you don't HAVE to create classes for Status, Type, and Role depending on how you implement your server)
3. Once your models exist, set up your JDBC to interact with your DB. Don't make or setup Websockets...1 step at a time because if you throw too much in then you won't know what is causing the issue when an issue arises...JUST do your JDBC for now. Make sure the basic crud methods work.
    a. Just run the JS file without making the server.

4. Once my DB schema and JDBC is functioning, I like to create my API endpoints using Websockets. I then test the endpoints to ensure they work INDEPENDENTLY of HTML pages.

Once I'm done with my API then my server is ready; now I can start on the front end. Again, this is a guideline for starting the project. This is not the only way to start the project but if you don't know how to start I'd say use this guideline.