**Revature – Project 1 Outline**

Technologies used: .NET 6, …

User Requirements (Model)

*Employee (Base class): (fields will be auto propertied in C#)*

* Data field for the employee id (primary key, int)
* [Data field indicating whether this person is a manager or not (int)](https://stackoverflow.com/questions/51450981/login-for-admin-and-ordinary-user#:~:text=As%20a%20general%20rule%2C%20the,database%20table%20for%20the%20users.)
* Data field for the email (string)
* Data field for the password (string)

*Constructor:*

* Construct an employee object using an email, password, and primary key. User passes in email and password; system determines the primary key. Default value for the field indicating the employee’s role is 0 (normal).

*Reimbursement Ticket (Base Class): (fields will be auto propertied in C#)*

* Data field for the ticket ID (primary key, int)
* Data field for the employee ID (foreign key, int)
* Data field for the type/reason for the ticket (string)
* Data field for the dollar amount for the ticket (double)
* Data field for the description of the ticket (string)
* Data field for the status of the ticket (int)

*Constructor:*

Construct a ticket object using the employee id, reason, amount, description, and primary key. User passes in the employee id, reason, amount, and description. The system determines the primary key. Default value for the status field is 0 (pending).

*Required Functionality (Minimum Viable Product User Stories)*

# I can register with a unique email and password combination

* Need function to register a new employee to the system.
  + Have an Action method that does a post request using a unique email and password combination. Returns true or false depending on whether the user was successfully registered. Log the registration attempts, with information.
  + Business layer gets list of employees from the repository layer. Using the arguments from the action method, create an employee object; If arguments don’t satisfy our conditions return false. Add the new employee to the list. Send back the list of employees to the repository layer to create/update our list of employees and return true. Log these cases.
  + Repository layer will need a method to “get” our list of employees, and a method to “update/post” our list of employees. Log whenever these methods get called.
    - The getter method will read from our employee database (for now, it’s a JSON file). No arguments, returns a list of employee objects from our database (or an empty list, if no database exists… log this).
    - The post/update method will take a list of employee objects as an argument and write/overwrite our employee database, returns true (return false if something happens).

# I can login to the application with a unique email and password combination

* Need function to validate a login request using an email and password.
  + Have an Action Method that does a get request using email and password strings as arguments. Returns an employee object if there is an employee with that matching login combination, otherwise [return null](https://stackoverflow.com/questions/1626597/should-functions-return-null-or-an-empty-object). Log the login attempts.
  + Business layer takes email and password as arguments and gets a list of employee objects from the repository layer. Compare the email and password passed from the action method with the employees from our database. If there is a match return the appropriate employee, otherwise return null. Log these cases.
  + Repository layer fetches a list of employee objects.

# As an Employee, I can submit a new reimbursement ticket with a type (travel, food, car rental, etc.), a dollar amount, a description, and a default status of "Pending"

* Need reimbursement ticket object
  + Create ticket class & class requirements
* Need function to let an employee create a reimbursement ticket
  + Have an action method that does a post request and takes in the employee id, ticket type, dollar amount, and description as arguments to create a ticket object. Returns true or false depending on whether a reimbursement ticket is successfully made. Log the ticket attempts.
  + Business layer gets list of tickets and employees from the repository layer. First check if the employee exists in the system. Then, using the arguments from the action method, create a ticket object; If arguments don’t satisfy our conditions return false. Add the new ticket to the list. Send back the list of tickets to the repository layer to create/update our list of tickets and return true. Log if the operation was successful or we ran into problems.
  + Repository layer will need a method to “get” our list of tickets (we get our employees from an existing method), and a method to “update/post” our list of tickets. Log when the methods get called.
    - The getter method will read from our ticket database (for now, it’s a JSON file). No arguments, returns a list of ticket objects from our database (or an empty list, if no database exists… log this).
    - The post/update method will take a list of ticket objects as an argument and write/overwrite our ticket database, returns true (errors out if something happens).

# As a Manager, I can access a queue of pending tickets that only managers can see.

* Need function to return a list of PENDING reimbursement tickets if the employee has manager permissions.
  + Have an action method that does a get request and takes in an employee id and returns a list of pending tickets. Returns null if the employee does not have permissions. Log that an employee is trying to access the pending tickets.
  + Business layer gets list of tickets and employees from the repository layer. First, check if the employee with the specified id exists and has permissions; return null if any of these cases fail. Then we create a new list of tickets. Iterate through the tickets we got from the database and add tickets that have the pending status to our new list. Return the list of pending tickets. Log if we were successful, or if we failed a check or something.
  + Repository layer will fetch our list of tickets.

# As a Manager, I can "approve”, or "reject" tickets submitted by employees

* Need function that allows employees with the manager role to approve tickets submitted by employees.
  + Have an action method that does a post/put request and takes in a ticket ID argument to edit a ticket object as well as an employee id argument to check if the employee has permissions to edit. Returns true or false depending on whether the ticket was approved successfully. Log that an employee is trying to approve a ticket.
  + Business layer gets list of tickets and employees from the repository layer. First, check if the employee with the specified id exists and has permissions; return false if any of these cases fail. Then, using the ticket ID, edit the status of the specified ticket to “approved”. If the specified ticket does not exist or the employee trying to edit is not a manager, return false. Otherwise, send back the list of tickets to the repository layer, return true. Log whether the ticket was approved or not (failed check, etc.).
  + Repository layer will fetch and update our list of tickets using the get and post methods. It will also fetch our list of employees.
* Need function that allows employees with the manager role to reject tickets submitted by employees.
  + Almost the same as approving, but instead of changing the status to “approved”, change the status to “rejected”. Log the same way as in the above method, but instead of saying approved or not, say rejected or not.

# As an Employee, I can view all my own previous reimbursement ticket submissions.

* Need function to return a list of reimbursement tickets belonging to a specific employee.
  + Have an action method that does a get request and takes in an employee id and returns a list of tickets belonging to that employee. Returns null if the employee does not exist. Log that an employee is trying to access their tickets.
  + Business layer will get a list of tickets and employees from the repository layer. First, check if the employee exists; Return null if they don’t. Then we create a new list of tickets. Iterate through the tickets we got from the database and add tickets that have the specified employee id to our new list. Return the list of the employee’s tickets. Log if we were successful and give information about how many entries, etc., or if we failed along the way.
  + Repository layer will fetch our list of tickets and employees.

# As an Employee, I can edit my user account information.

* Need function to edit a specific employee from our database of employees.
  + Have an action method that does a post/put request and takes in an employee id and the fields we want to edit (OVERLOADING: change perms, change email, change password) and returns true or false depending on whether the employee was successfully edited. Log that an employee is being edited.
  + Business layer will get a list of employee objects from the repository layer. First, check if the employee exists using the employee id and reference it. Do our changes, send back the list of employees, and update our database. Log if we were successful or ran into problems.
  + Repository layer fetches the list of employees and updates it.