Software Engineering
System Requirements Specifications (SRS) Document

Project: BioTool

02/06/2024

Version: 1.0

By: Alex Olson, Sola Yun, Will Hutcheon

Table of Contents

1. Introduction

1.1. Purpose

The Biotool application aims to enable scientists and managers to quickly find biological data from protein/organism data banks, produce phylogenetic graphs, and other information derived from biological data, and easily compile them in a holistic report.

1.2. Document Conventions

The purpose of this Software Requirements Document (SRD) is to describe the client-view and developer-view requirements for the Biotool web application. Client-oriented requirements describe the system from the client's perspective. These requirements include a description of the different types of users served by the system. Developer-oriented requirements describe the system from a software developer's perspective. These requirements include a detailed description of functional, data, performance, and other important requirements.

1.3. Definitions, Acronyms, and Abbreviations

Python	Java: A programming language originally developed by James Gosling at Sun Microsystems. We will be using this language to build the Restaurant Manager.] Python:
HTML	Hypertext Markup Language. This is the code that will be used to structure and design the web application and its content.
Flask	Flask: Python web framework, used to create and run our application.
MVC	Model-View-Controller. This is the architectural pattern that will be used to implement our system.
PyCharm	An integrated development environment (IDE) for Python. This is where our system will be created.
API	Application Programming Interface. This will be used to implement a function within the software where the current date and time is displayed on the homepage.

1.4. Intended Audience

[Describe which part of the SRS document is intended for which reader. Include a list of all stakeholders of the project, developers, project managers, and users for better clarity.]

End User: 2.1 Product Features, 2.3 Operating Environment, 3.1 Primary
Project Manager:
Administrator:

1.5. Project Scope

The goal of the software is to provide a quick and reliable web app to be used by researchers, managers, and admins of a company to facilitate the processing of usually unwieldy biologic data. This aligns with the overall business goals of the company as the less time is spent on routine data procurement and processing, the more time can be spent doing critical analysis and creating value for the business.

The benefits of the project to business include:

- Operational efficiency: cutting down on time spent querying databases as users do not need to spend their valuable time working with low-level details, and can spend more time on high-level ideas.
- Employee happiness: Employees will be more available to do creative and meaningful work instead of being encumbered with rote data processing.

1.6. Technology Challenges

We have not run into any technical challenges yet.

1.7. References

We have not used any reference material yet

2. General Description

2.1. Product Features

The product features include the ability for end users to create accounts, and administrators to manage those accounts. Users can then query databases for biologic data, save the data, and produce reports containing phylogenetic trees, and other chosen computed data that they may need. After a report is generated, a user can view, print, or delete their own reports. The manager can view, edit, print, or delete any user reports, and can also choose which data gets included in the user's reports. The administrator can create and delete accounts along with changing passwords.

2.2. User Class and Characteristics

Our web app will not require users and managers to have any technical knowledge about computers except to be able to access a web page and create an account. Users and managers will be expected to have knowledge of biology and an understanding of the data being processed.

2.3. Operating Environment

The application is designed to be accessed on the internet from any internet-capable device, including but not limited to Windows, Mac, Unix-like OS, iOS, and Android.

2.4. Constraints

My main thought about constraint is how the user will interact with the API. In the end, we need the API to return a string of nucleotide data (ie: “ATCGTCCGA”) to be processed. But how is the end user going to input information to search for the specific protein/gene/organism, and how are we going to verify the input, if we are at all.

2.5. Assumptions and Dependencies

The software will be dependent on Flask to create and execute the MVC architecture that will be developed within PyCharm. The application will also use the MyGene API (http://mygene.info/) that will return data about desired nucleotide strings, proteins, or entire genes.

3. Functional Requirements

3.1. Primary

- **FR0: The system will allow the user to look up biologic information using a symbol (i.e.: CDK2 for the protein "cyclin-dependent kinase 2"), and save this information in a database.**
- **FR1: The system will allow the user to generate a phylogenetic tree using selected items from the database.**
- **FR2: The system will allow the user to generate a report with the specific details of the selected biologic data, including the phylogenetic tree, and comparative data derived from the chosen nucleotide strings.**

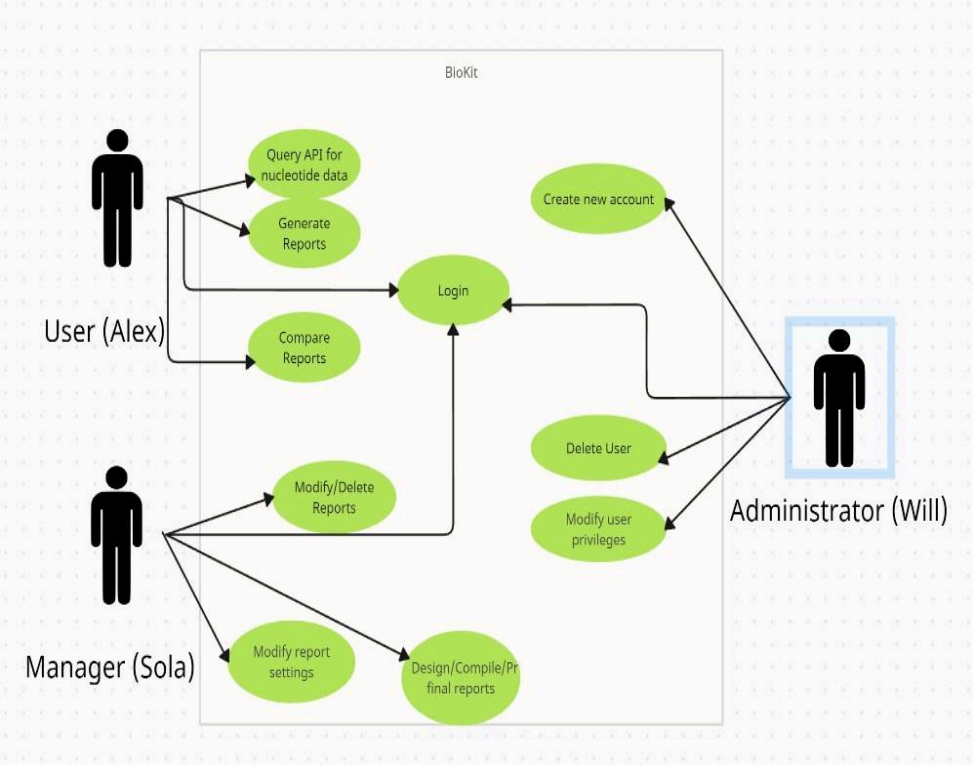
3.2. Secondary

- **Password protection for information only accessible to end users and their managers.**

- **Authorization scheme so that individual users will not be able to view or change other end user's reports.**

3.3. Use-Case Model

3.3.1. Use-Case Model Diagram



3.3.2. Use-Case Model Descriptions

3.3.2.1. Actor: User (Alex)

- Query API for nucleotide data : The user can search for the nucleotide string / protein / gene / organism to save to the database.

- Generate Report: The user can generate a report using chosen biologic data from the Database

- Login: The user can enter a username and password to login to the application and authenticate themselves.

- Compare reports: The user can review their own saved reports, and see the summarized data side by side for comparison.

3.3.2.2. Actor: Manager (Sola)

- Modify/Delete reports: The manager can modify existing reports to change the information displayed and can delete the reports.

- Modify report settings: The manager can modify the settings to change the look of the final reports, including font sizes and color, spacing, and what is included.

Compile/Print reports: The manager can select multiple reports to create a final compiled product to print.

- Login: The manager can enter a username and password to login to the application and authenticate themselves.

3.3.2.3. Actor: Administrator (Will)

- Create new account: The Admin can create a new account for a User, Manager, or another Admin and assign them a username and password.

- Delete user: The Admin can delete an existing User, Manager, or Admin (but only if there is more than one)

- Change passwords: The Admin can change the password of any previously created account.

- Login: The user can enter a username and password to login to the application and authenticate themselves.

33.3. Use-Case Model Scenarios

3.3.3.1. Actor: User(Alex)

- **Use-Case Name:** Query API

  - **Initial Assumption:** The user has access to the webapp and has the user role.

  - **Normal:** The user will enter the keyword for the protein / gene / organism they would like to include in their report

- **What Can Go Wrong:** The user queries the API with an invalid identifier. The system should alert the user about this error.
- **Other Activities:** n/a
- **System State on Completion:** The biologic data is saved in the next available space of the CSV file, and is accessible when compiling a report.

- **Use-Case Name:** Generate Report
  - **Initial Assumption:** The user has access to the webapp and has the user role.
  - **Normal:** The user selects the data to be compiled from a check box of all available specimen data. Once desired data is chosen, a phylogenetic tree is produced, and the related information is compiled and saved.
  - **What Can Go Wrong:** n/a
  - **Other Activities:** n/a
  - **System State on Completion:** A report is compiled and saved to persistent storage, and viewable by the user who created it and the manager.

- **Use-Case Name:** Compare reports

- **Initial Assumption:** The user has access to the webapp and has the user role.

- **Normal:** The user will select the previously generated reports of their own, and be able to view them side by side.

- **What Can Go Wrong:** There are no previously generated reports, the system will notify the user.

- **Other Activities:** n/a

- **System State on Completion:** The system state is the same as before comparing reports.

3.3.3.2. **Actor:** Manager (sola)

- **Use-Case Name:** Modify reports

  - **Initial Assumption:** The manager has access to the webapp and has the manager role.

  - **Normal:** The manager opens the report page, and selects the report they would like to view. Then the manager is able to add or remove specific data sections,

  - **What Can Go Wrong:** There is no report saved for the manager to view. The page will notify the manager that there are none yet.

- **Other Activities:** A manager can delete the viewed report.

- **System State on Completion:** The report is saved again with the desired modifications, or if deleted, removed from persistent storage.

- **Use-Case Name:** Modify report settings

- **Initial Assumption:** The manager has access to the webapp and has the manager role.

- **Normal:** The manager opens the settings page and can modify the settings to change the look of the final reports, including font sizes and color, spacing, and what is included.

- **What Can Go Wrong:** n/a

- **Other Activities:** n/a

- **System State on Completion:** The report settings are changed for future generated reports.

- - **Use-Case Name:** Print reports

- **Initial Assumption:** The manager has access to the webapp and has the manager role.

- **Normal:** The manager opens the print page, and selects the saved reports they would like to include in the printed product. The manager then is able to print the finished report.

- **What Can Go Wrong:** There is no report saved for the manager to view. The page will notify the manager that there are none yet.

- **Other Activities:** n/a

- **System State on Completion:** The report is printed and the system returns to the state it was in previously.

3.3.3.3. Actor: Admin (Will)

- **Use-Case Name:** Create new account

  - **Initial Assumption:** The admin has access to the webapp and has the admin role.

- **Normal:** The admin selects the user who needs their password changed from a drop-down box. The admin is prompted to input a new password for the user. The password is then saved.
- **What Can Go Wrong:** n/a
- **Other Activities:** n/a
- **System State on Completion:** The user's password is changed.

- **Use-Case Name:** Delete user
  - **Initial Assumption:** The admin has access to the webapp and has the admin role.
  - **Normal:** The admin selects the user to delete from a drop-down menu. After the user is selected, the system verifies that is not the last admin account, and prompts the admin for confirmation that they want to delete the account.
  - **What Can Go Wrong:** That there is only one admin account, and is selected to be deleted. The system will notify the admin, and not delete the account.
  - **Other Activities:** n/a

- **System State on Completion:** The account is deleted from persistent storage.

- **Use-Case Name:** Modify user privileges

  - **Initial Assumption:** The admin has access to the webapp and has the admin role.

  - **Normal:** The admin selects the user to modify from a drop-down menu. After the user is selected, the admin adds or removes privileges by selecting check boxes.

  - **What Can Go Wrong:** n/a

  - **Other Activities:** n/a

  - **System State on Completion:** The modified user's new privileges are saved.

4. Technical Requirements

4.1. Interface Requirements

4.1.1. User Interfaces

4.1.2. Hardware Interfaces

The application is designed to be accessed on the internet from any internet-capable device, including but not limited to Windows, Mac, Unix-like OS, iOS, and Android.

4.1.3. Communications Interfaces

It must be able to connect to the internet.

The communication protocol, HTTP, must be able to connect to the MyGene.info API.

4.1.4. Software Interfaces

We will use Flask with Python along with HTML to build the front end.

For persistent storage, we will use a CSV file.

We will also use Flask to connect the frontend and backend.

-

5. Non-Functional Requirements

5.1. Performance Requirements

- NFR0(R): The local copy of the biologic data database will consume less than 50 MB of memory
- NFR1(R): The system (including the local copy of the biologic database) will consume less than 100MB of memory
- NFR2(R): The novice user will be able to create a report in less than 5 minutes.
- NFR3(R): The expert user will be able to create a report in less than 1 minute.

5.2. Safety Requirements

5.3. Security Requirements

- NFR4(R): The system will only be usable by authorized users.

5.4. Software Quality Attributes

5.4.1. Availability

The web app will maintain a high availability, and if it crashes, will be able to restart and load in persistent data in under a minute.

5.4.2. Correctness

The web app will use unit testing along with useful and comprehensible error handling to ensure the correctness of the product.

5.4.3. Maintainability

This web app is designed to be easily modifiable, to assist in debugging, fixing, and extending the apps capabilities. The application is easily learned by new developers so training time is minimal.

5.4.4. Reusability

This web app will be reusable as long as the API is maintained.

5.4.5. Portability

This web app will be able to be ran on any device that is browser capable.

5.5. Process Requirements

5.5.1. Development Process Used

We are using the Agile Development Process

5.5.2. Time Constraints

Our time is constrained by the following due dates:

February 27th: Prototype presentation is due

April 2nd: Second iteration of design document is due

April 16th: First code review is due

April 30th: Final project is due

5.5.3. Cost and Delivery Date

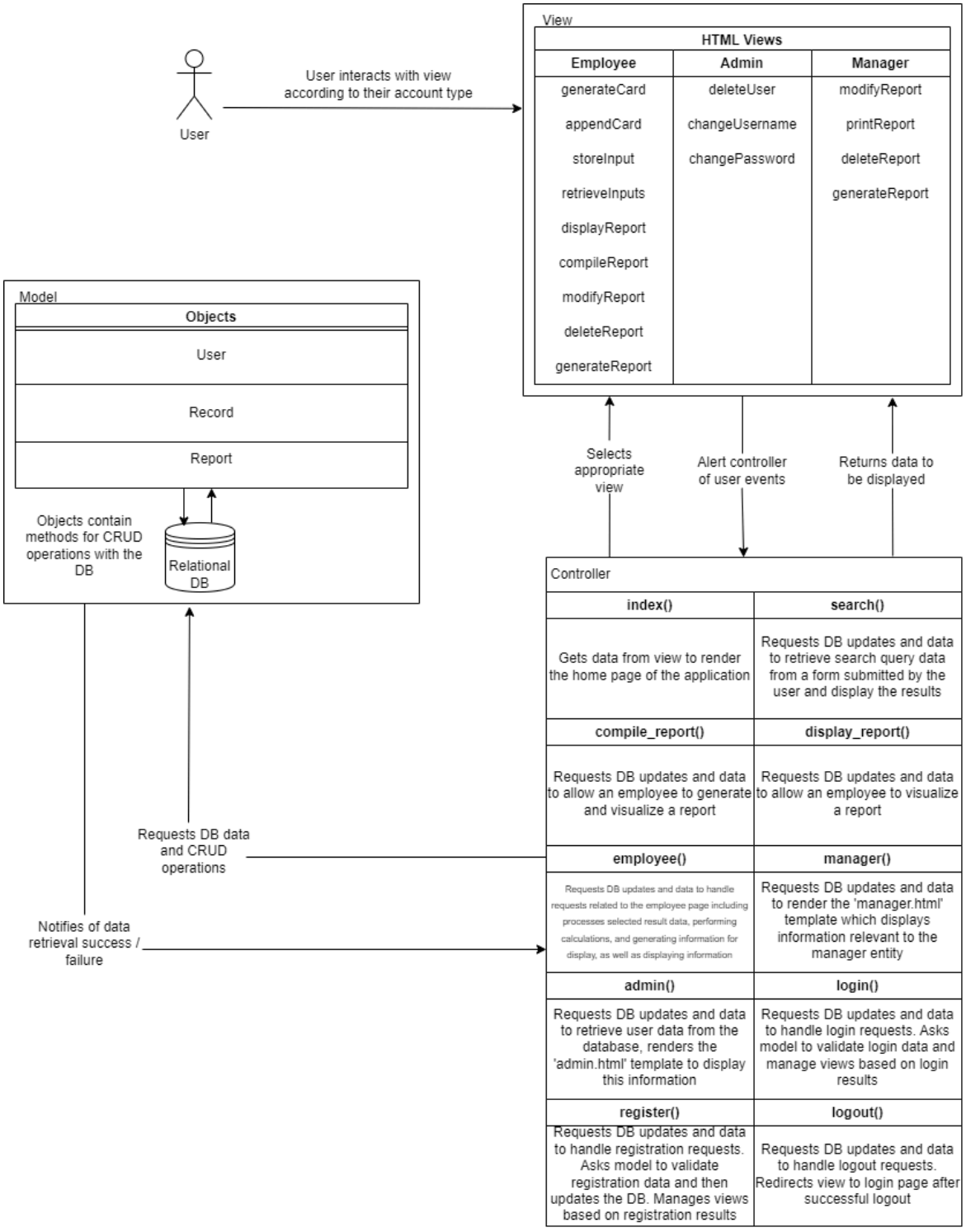
There is no cost for the software

The final product will be delivered by April 30th 2024

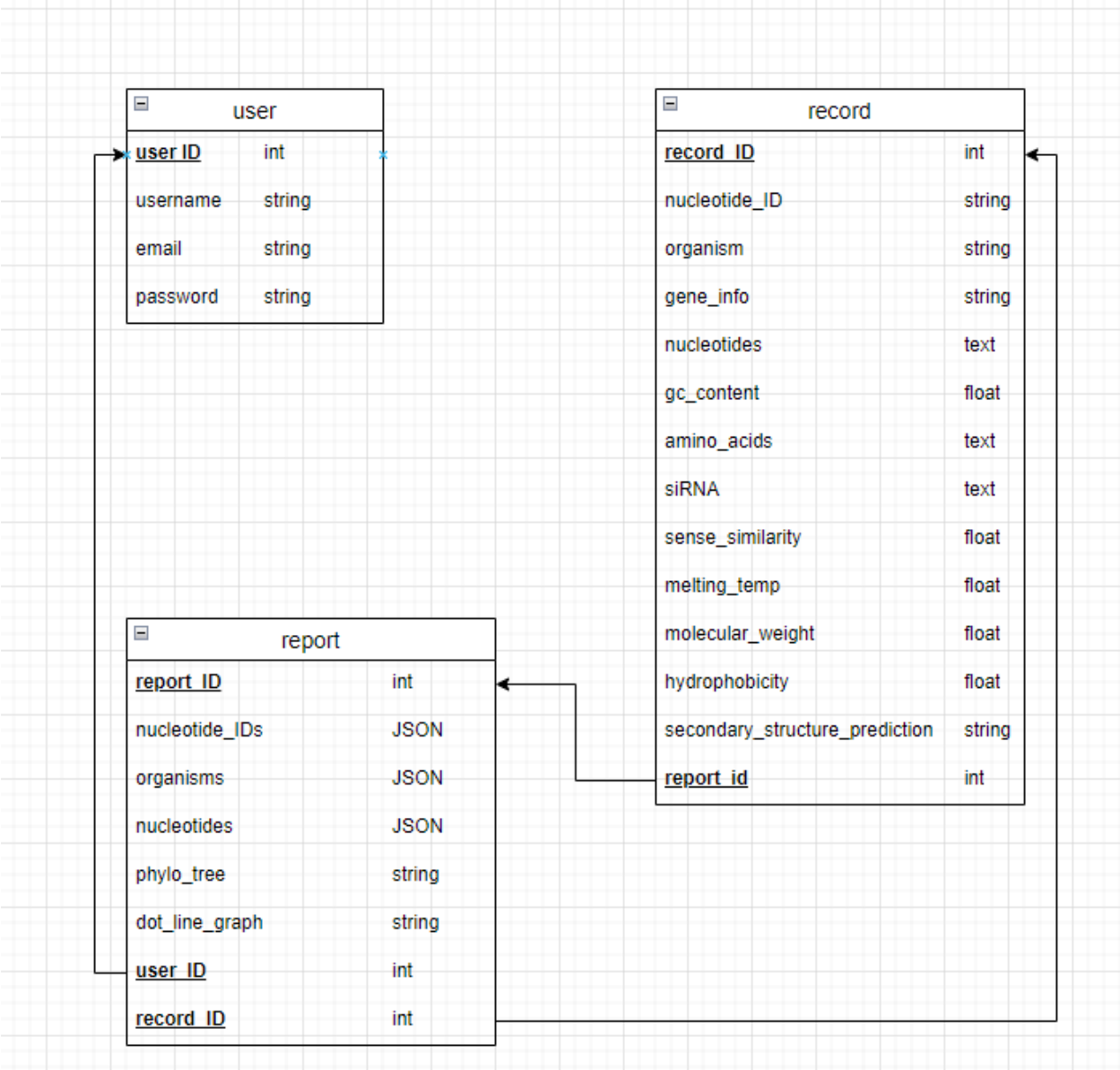
5.6. Other Requirements

6. Design Documents

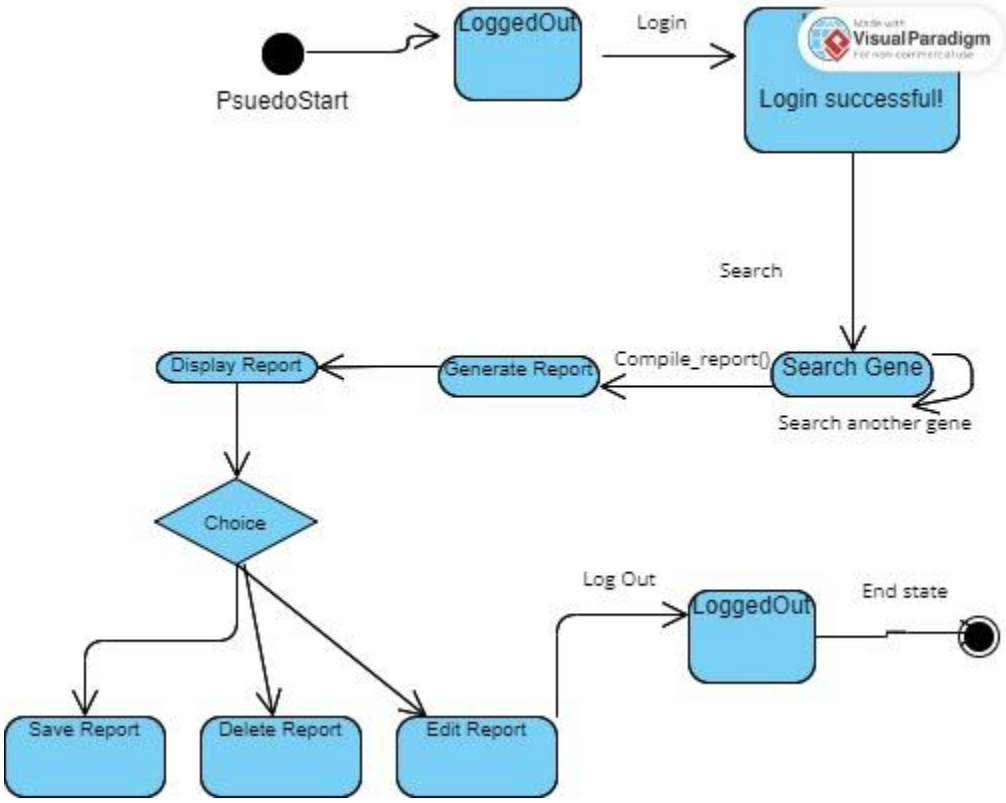
6.1. Software Architecture



6.2. High-Level Database Schema

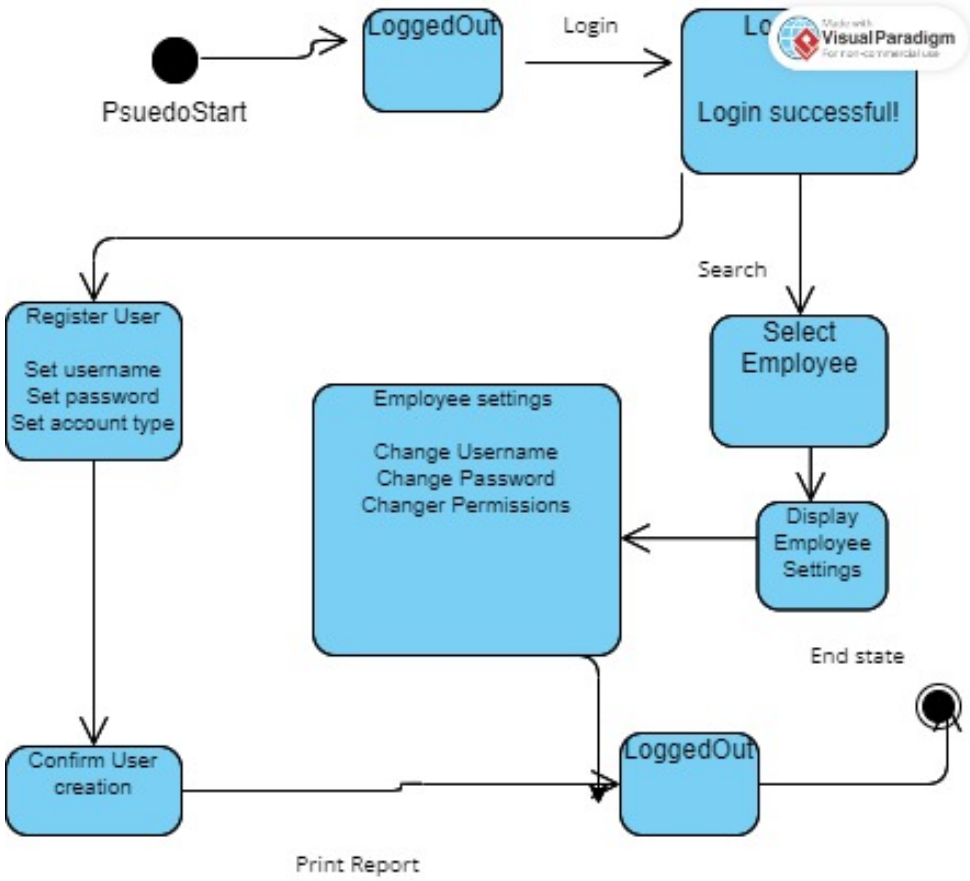


6.3. Software Design



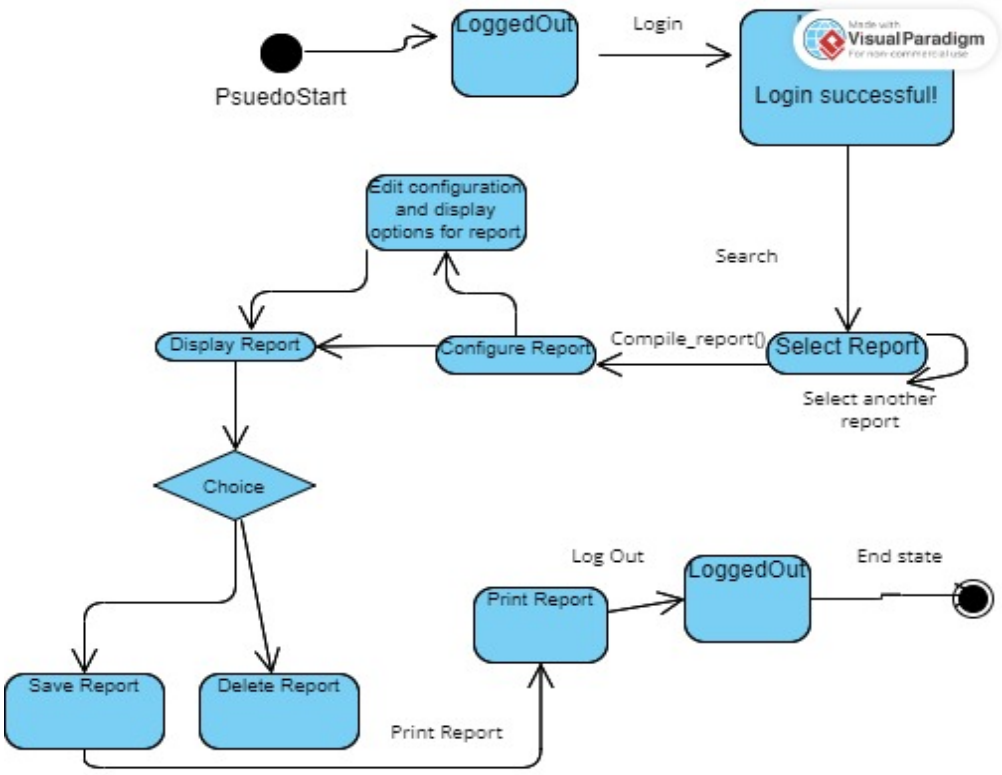
6.3.1 Employee

6.3. Software Design



6.3.2 Admin

6.3. Software Design



6.3.3 Manager

6.4. UML Class Diagram