**CS3354 Software Engineering**

**Final Project Deliverable 1**

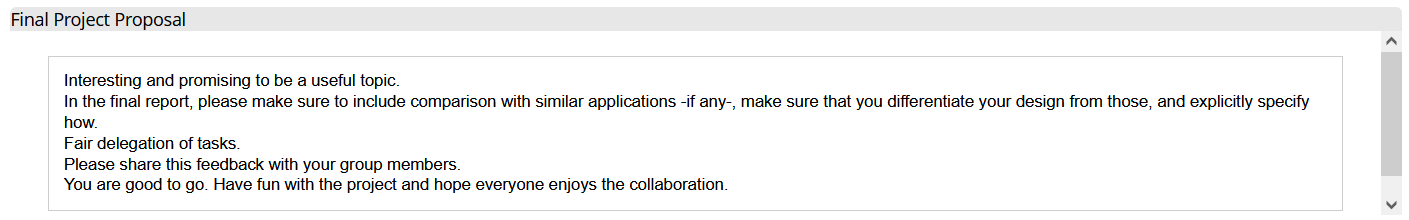
**Split Squad**

By: Tan Vo, Daniel Nguyen, Stephen George, Quan Pham,

Mir Ali, Samuel Preston, Amaan Babul, Shriniketh Mukundan

**1. [5 POINTS]** Please attach here the Final Project draft description (that contains the instructor feedback). It is ok to include a picture of the original document. Address the feedback provided for your proposal by listing what you did / plan to do to comply with those proposed changes and or requests for additions to your project.

|  |
| --- |
| **Description** |
| We'll be creating a mobile expense-sharing app to streamline the process of splitting costs among multiple people. Our goal is to simplify shared financial management for roommates, households, travel groups, teams, and event organizers to promote transparency and reduce conflicts in real-life scenarios. |



|  |
| --- |
| In compliance with the feedback, we will include the comparison with similar applications to make sure that our design is unique from our competitor. |

**2. [10 POINTS]** Setting up a GitHub repository. Please use your utdallas email accounts only for each group member.

1.1. Each team member should create a GitHub account if you don’t already have one.

1.2. Create a GitHub repository named 3354-teamName. (whatever your team’s name will be).

1.3. Add all team members, and the TA as collaborators. Our TA will post his GitHub info on EL:

**TA GitHub id:**

**TA email:**

1.4. Make the first commit to the repository (i.e., a README file with [team name] as its content).

1.5. Make another commit including a pdf/txt/doc file named “project\_scope”. If you choose a predefined topic (one of the 4 topics described in the “Project Topic Ideas” section of this document), the contents of the file should be identical to the corresponding project in this section. If you choose other topics, the contents should follow a similar structure.

1.6. Keep all your project-related files in your repository as we will check them. Include the URL of your team project repository into your project deliverable 1 report.

**Important Note:**

* Tasks 1.3 - 1.5 should be performed by different team members. We will check the commit history for these activities.
* Do not include credentials (e.g., UTD ID) in the repository.
* Only commits performed before the deadline will be considered. Do not forget to push your changes after you have done the work!

**3. [5 POINTS]** Delegation of tasks: Who is doing what. If no contribution, please specify as it will help us grade each group member fairly.

|  |  |  |
| --- | --- | --- |
| **Task** | **Task Description** | **Task Owner** |
| 2.1.3 | GitHub Invitation | Samuel Preston |
| 2.1.4 | README | Tan Vo |
| 2.1.5 | Project Scope | Stephen George |
| 4 | Process Model | Quan Pham |
| 5.a | Functional Requirements | Daniel Nguyen |
| 5.b | Nonfunctional Requirements | Mir Ali |
| 6 | Use case Diagram | Samuel Preston |
| 7 | Sequence Diagram | Tan Vo |
| 8 | Class Diagram | Shriniketh Mukundan |
| 9 | Architecture Design | Amaan Babul |

**4. [5 POINTS]** Which software process model is employed in the project and why. (Ch 2)

|  |
| --- |
| The agile process model is best suited for the development of our expense sharing app. Agile methodologies, such as Scrum, offer the flexibility for us to create new requirements/functionalities and make changes to the application per user feedback. Any development ideas can be addressed in a scrum meeting, and the development team would focus and work on each topic on a daily basis. In the first scrum meeting, the team will come together with the customers and lay out the basic requirements and functionalities for the app (can be seen in project\_scope and functional/non-functional requirements specified in other sections of this deliverable) and begin the development process. For every meeting afterwards, the team will attempt to implement each functionality and requirement and have the customers review them. Each feature should be kept simple and adaptable for the development process to take after the agile process model. The increment phase and development process will end when customers are mostly satisfied with the final product (every requested feature is implemented with little to no issues). |

**5. [15 POINTS]** Software Requirements including

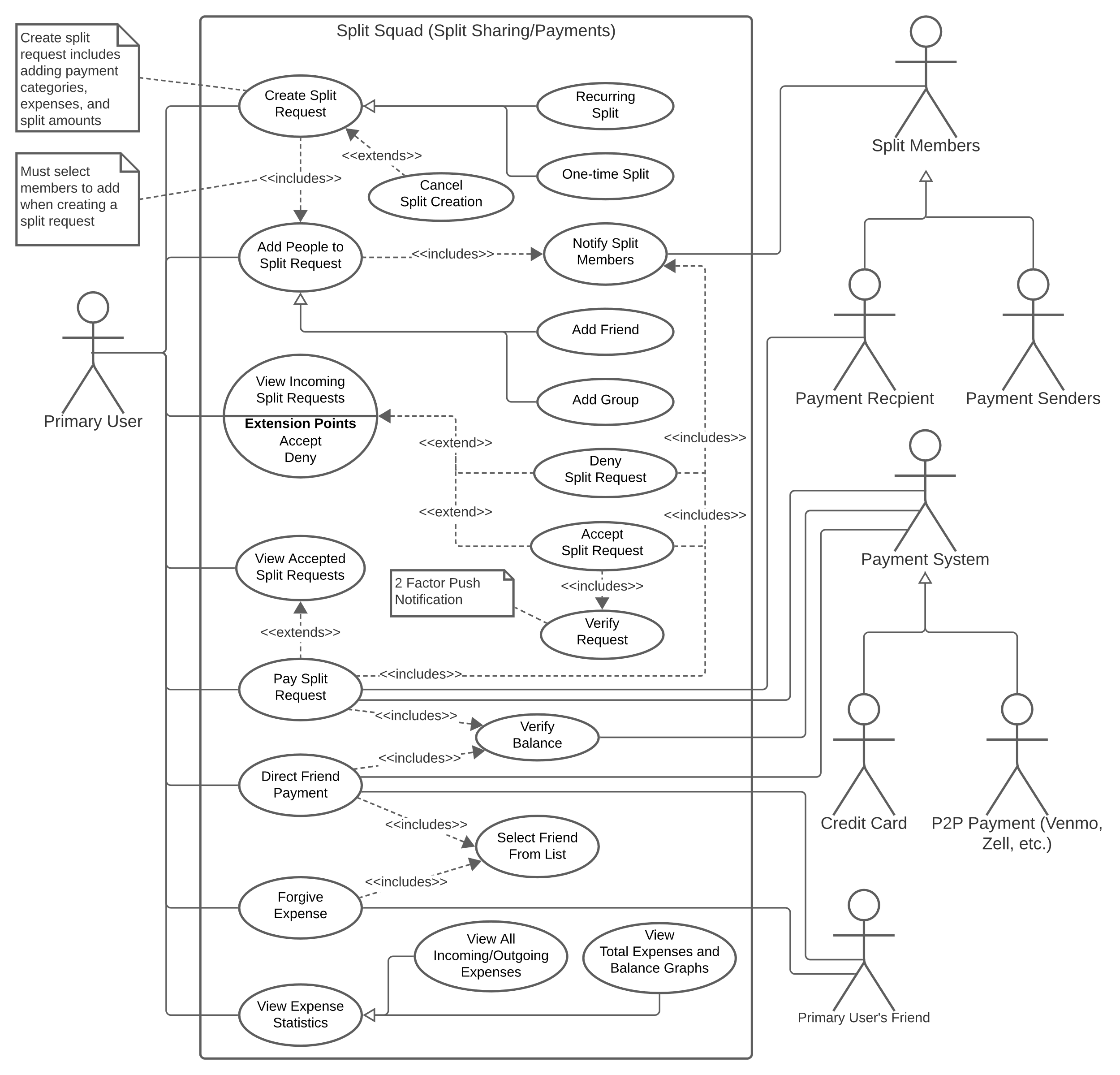
**5.a.) [5 POINTS]** Functional requirements. To simplify your design, please keep your functional requirements in the range of minimum 5 (five) to maximum 7 (seven). (Ch 4)

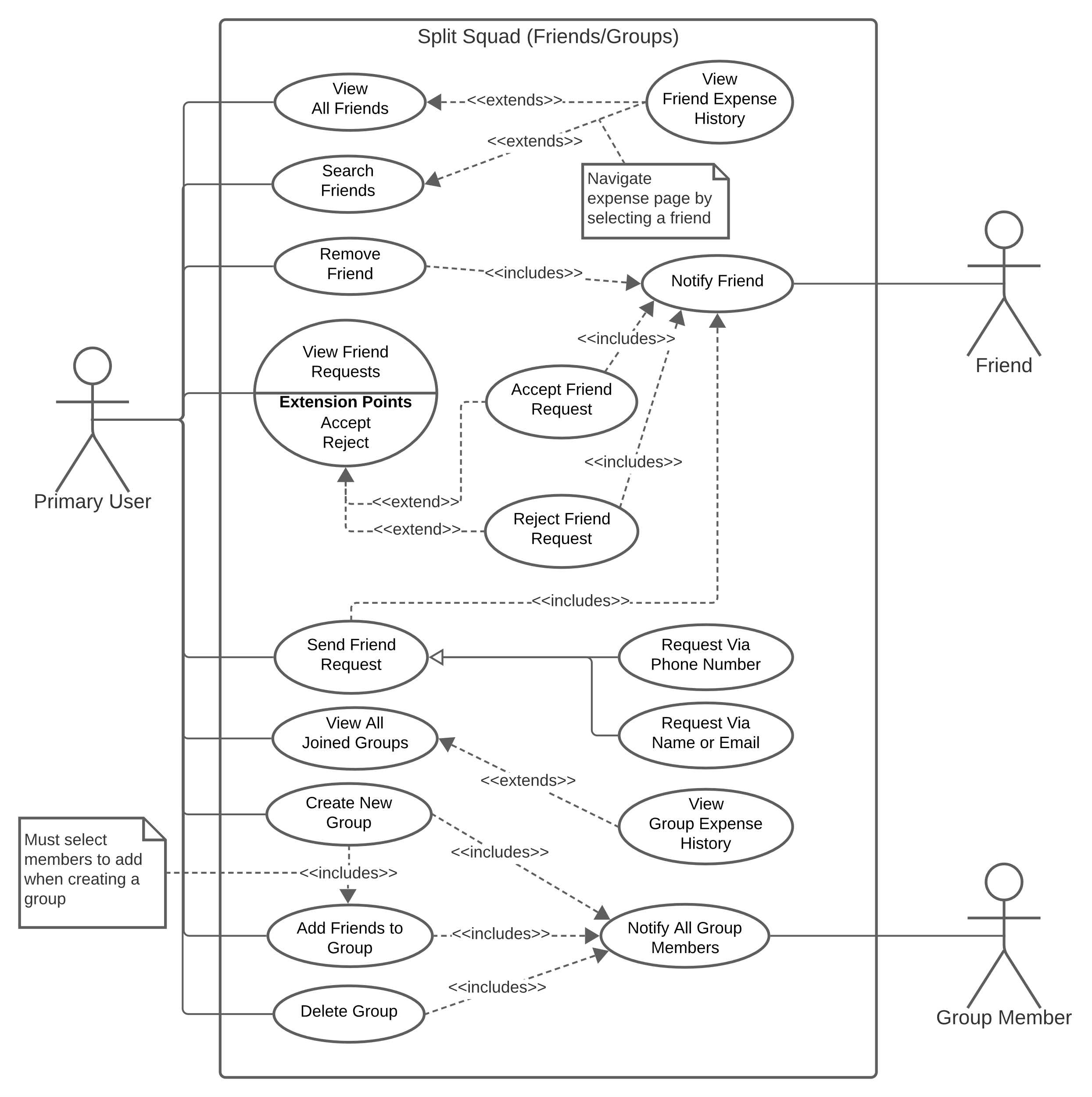
|  |
| --- |
| 1. The user shall be able to create a split request among one or more other users with fields to input total expense, to split total expense in the split request equally, by percentage, by custom amount, or by a mix of percentages and custom amount with options to add periodicity (weekly/monthly) or categories (food, rent, trips, other) to the split request. 2. The user shall be able to see and accept or deny a pending split request and, if accepted, pay the expense back partially or in full via a selected payment integration such as debit card, credit card, or a supported Peer-to-Peer payment system such as Paypal or Venmo. 3. The user shall be able to view expense organization such as history of all incoming/outgoing expenses (“I owe you”/”You owe me”) and current unpaid incoming/outgoing expenses or statistics such as total incoming/outgoing expenses, net balance of expenses, graph of net balance over time, and expenses by category (food, rent, trips, other). 4. The user shall be able to forgive or partially forgive an outgoing expense with an optional note for adding a reason such as cash payment or favor. 5. The user shall be able to add/view/search for/remove friends and create/view/search for/delete groups and see all related expense information for the friend or group. 6. The user shall be able to view and change account settings such as name, email, phone, profile picture, payment information, password, language, dark/light view mode and notification settings such as toggles for text/email/mobile push notifications, friend requests notifications, pending split request notifications, and incoming payment notifications. 7. The system shall store all user information, friend/group information, expense records in server database(s) belonging to the app. |

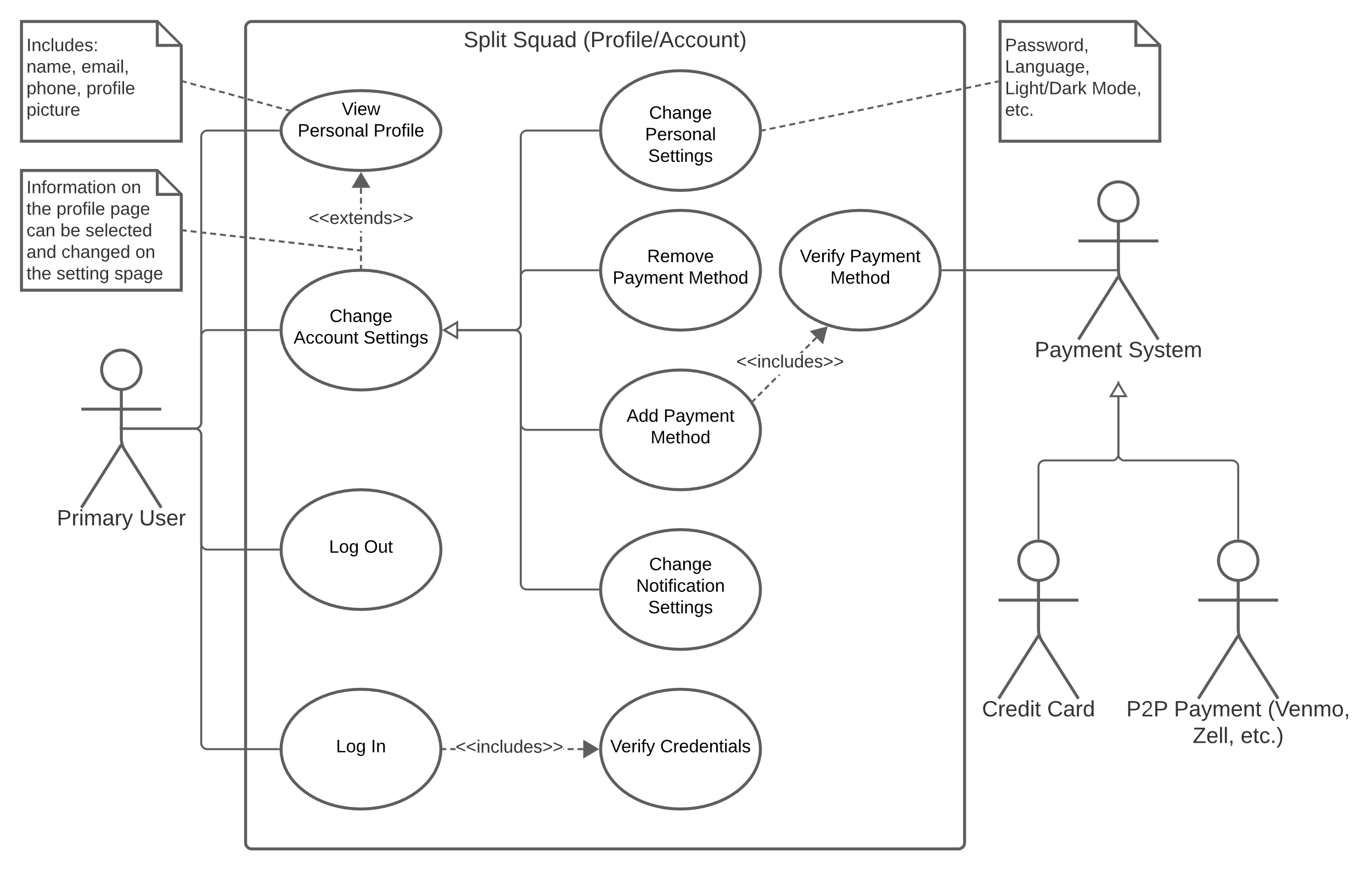
**5.b.) [10 POINTS]** Non-functional requirements (use all non-functional requirement types listed in Figure 4.3 - Ch 4. This means provide one non-functional requirement for each of the leaves of Figure 4.3. You can certainly make assumptions, even make up government/country-based rules, requirements to be able to provide one for each. Please explicitly specify if you are considering such assumptions.)

|  |
| --- |
|  |

**6. [15 POINTS]** Use case diagram – Provide a use case diagram (like Figure 5.5) for your project. Please note that there can be more than one use case diagram as your project might be very comprehensive. (Ch 5 and Ch 7)







**7. [15 POINTS]** Sequence diagram – Provide sequence diagrams (like Figure 5.6 and Figure 5.7) for each use case of your project. Please note that there should be an individual sequence diagram for each use case of your project. (Ch 5 and Ch 7)

|  |
| --- |
|  |

**8. [15 POINTS]** Class diagram – Provide a class diagram (like Figure 5.9) of your project. The class diagram should be unique (only one) and should include all classes of your project. Please make sure to include cardinalities, and relationship types (such as generalization and aggregation) between classes in your class diagram. Also make sure that each class has a class name, attributes, and methods named (Ch 5).

|  |
| --- |
|  |

**9. [15 POINTS]** Architectural design – Provide an architectural design of your project. Based on the characteristics of your project, choose and apply only one appropriate architectural pattern from the following list:

(Ch 6 section 6.3)

9.1. Model-View-Controller (MVC) pattern (like Figure 6.6)

9.2. Layered architecture pattern (like Figure 6.9)

9.3. Repository architecture pattern (like Figure 6.11)

9.4. Client-server architecture pattern (like Figure 6.13)

9.5. Pipe and filter architecture pattern (like Figure 6.15)

|  |
| --- |
|  |