Smart-Split Requirements Specification

Cristian Balan,
Zane Richards,
Leo Martinez,
Nick Azzouz,
Xavier Evans

Introduction

Overview

The purpose of the Software Requirements Specification document is to provide the following:

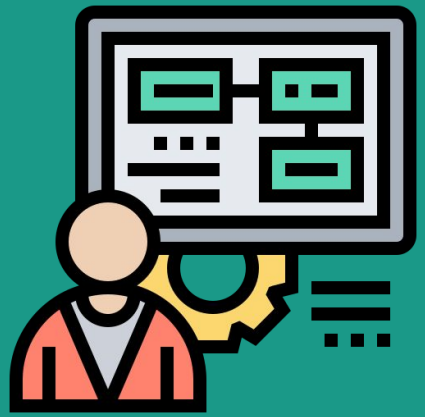
- Functional and non-functional requirements
- Context
- Use Cases
- Class Models

The document serves as a guide to direct the design and implementation of the final system.

System Purpose

Users	Location	Responsibilities	Need
• Group Travelers • Roommates • Landlord • Everyday users	• Web and Mobile platforms • Internet access from any location	• Provide users with reliable online payment tracking • Provide group transactions and services	• Benefits vast group of users as previously mentioned

Functional Requirements



High Priority

Software shall allow users to…

- Create a transaction for themselves or as a group. **(Expense Tracking)**
- Pay outstanding balances toward a group transaction. **(Contribution)**
- Add various payment methods to their account. **(Digital wallet)**
- Secure payment functions behind password verification. **(Validation)**

Medium Priority

Software shall allow users to…

- Upload a picture of the receipt and item(s) purchased.
- Create unique name IDs.
- Add a profile photo.

Low Priority

Software shall allow users to…

- Translate the names of items purchased from foreign merchants. (Translation)
- Increase the fonts and contrast of the interface. (Accessibility)
- Input their birthday and choose to have it publicly announced to friends.

Non-Functional Requirements

Non-Functional Requirements

Reliability

- System shall be completely operational with the expected traffic.
- System will schedule maintenance updates during low operational hours (night).
- Customer service department to help resolve any low-level issues.

Usability

- System will split bills between two or more parties.
- System will maintain a clean UI to allow for a simple user experience.

Non-Functional Requirements

Performance

- System should handle simultaneous connections among users ranging from 2 to 100,000.
- System will update to reflect the information in the database in a timely manner.
    - No more than 30 seconds.
- Reports generated by the system should take no more than 30 seconds to be available.

Security

- The user-facing services shall only be accessible via password for each user account.
- Transactional information shall only be sent to the involved parties via email or download.

Non-Functional Requirements

Supportability

- System should accommodate new payment methods as they arise and are approved.
- System's web-app must be operable on browsers such as: Google Chrome, FireFox, and Safari.

Online user Documentation and Help

- System shall provide an optional tutorial for first-time users.
- The tutorial will be condensed into a step-by-step startup guide.

Non-Functional Requirements

Interfaces

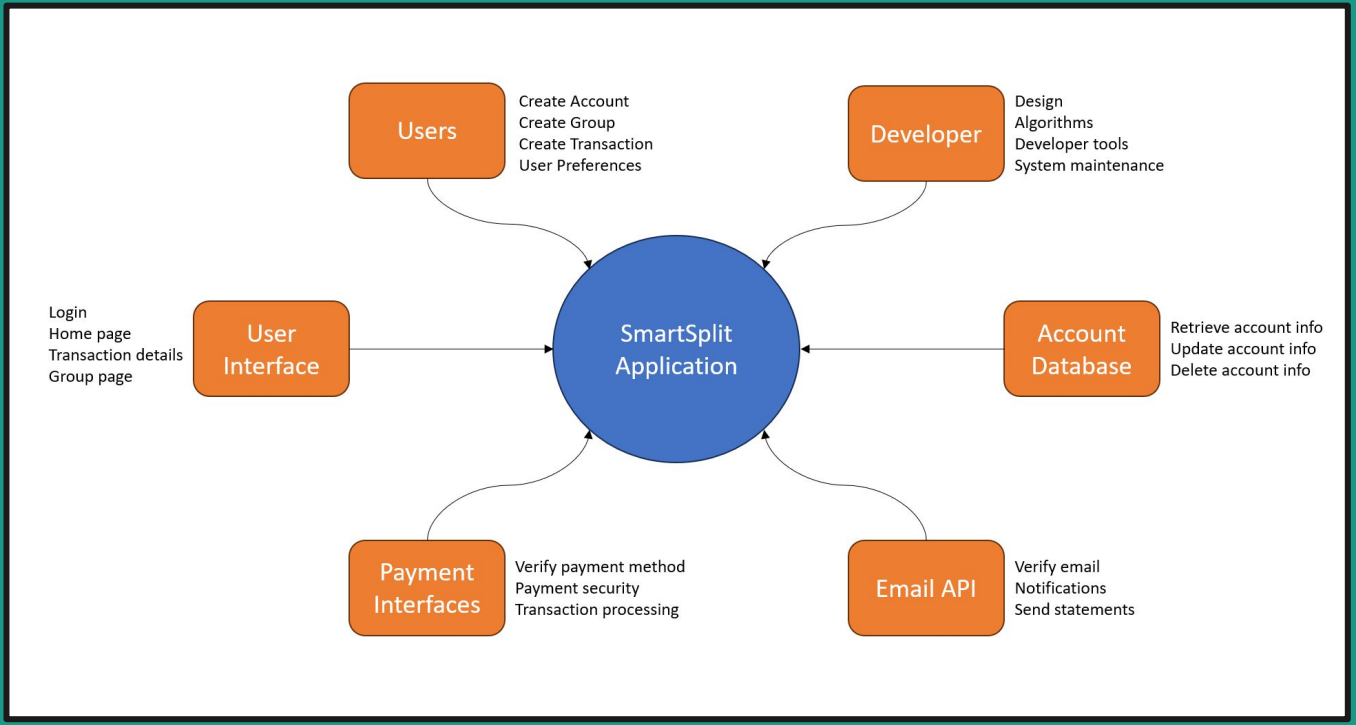
The system must interface with:

- The SQLite database for group and transactional information
- The financial records and transactions database (bank transactions)
- The language translation tool

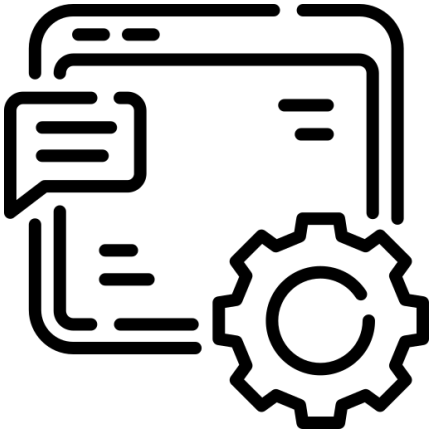
Scalability

- The servers should provide adequate resources for the amount of user accounts and simultaneous usage.
- No less than 30% of headroom for memory and storage resources.

Context Model



System Externals



Users

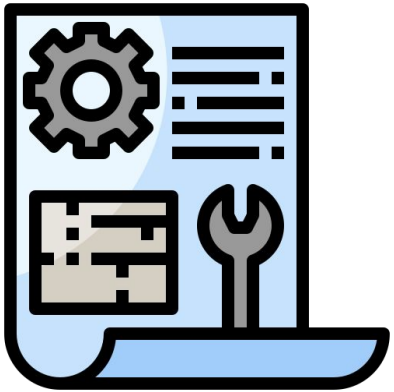
Developer

Account Database

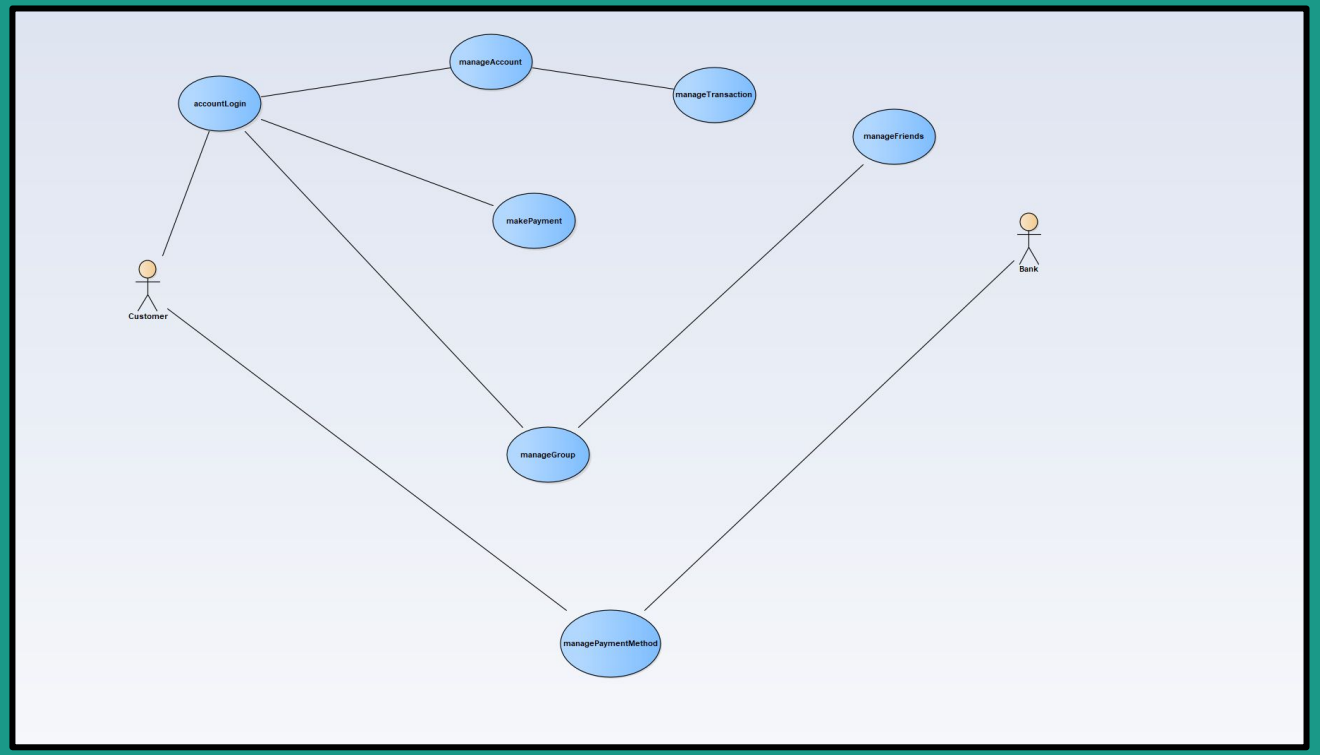
Email API

Payment Interfaces

User Interface



Use Case Model



Use Case Descriptions

Use Case Name:	**Create Account**
Participating actor:	Customer
Entry condition:	Valid email
Exit condition:	Customer has account
Event flow:	1. Customer selects create account option 2. Customer enters a valid email 3. Customer enters a sufficient password 4. Customer completes recaptcha 5. Customer receives confirmation via email

Use Case Name:	**Account Login**
Participating actor:	Customer
Entry condition:	The customer is registered.
Exit condition:	The customer can now obtain data and perform functions
Event flow:	1. The use case starts when a user indicates that he wants to login. 2. The system requests the username and password. 3. The user enters his/her username and password. 4. The system verifies the username and password against all registered users. 5. The system starts a login session and displays the home page.

Use Case Name:	**Manage account**
Participating actor:	Customer
Entry condition:	The customer has successfully logged in.
Exit condition:	The system updates account
Event flow:	1. The use case starts when a user selects manage account option 2. The user can have the option to delete their account 3. The user can customize their profile by adding a picture and a bio. 4. The system updates the account

Use Case Name:	**Manage payment method**
Participating actor:	Customer/ Bank
Entry condition:	The customer is logged in and has valid payment information.
Exit condition:	The customer's account will now have the desired payment methods set to their account.
Event flow:	1. The use case starts when a customer indicates that he/she wants to add, modify, or delete payment method(s). 2. The system displays all current payment methods associated with the customer's account. 3. The customer adds/updates/deletes payment method(s). 4. The system stores any change to the customer's payment method(s).

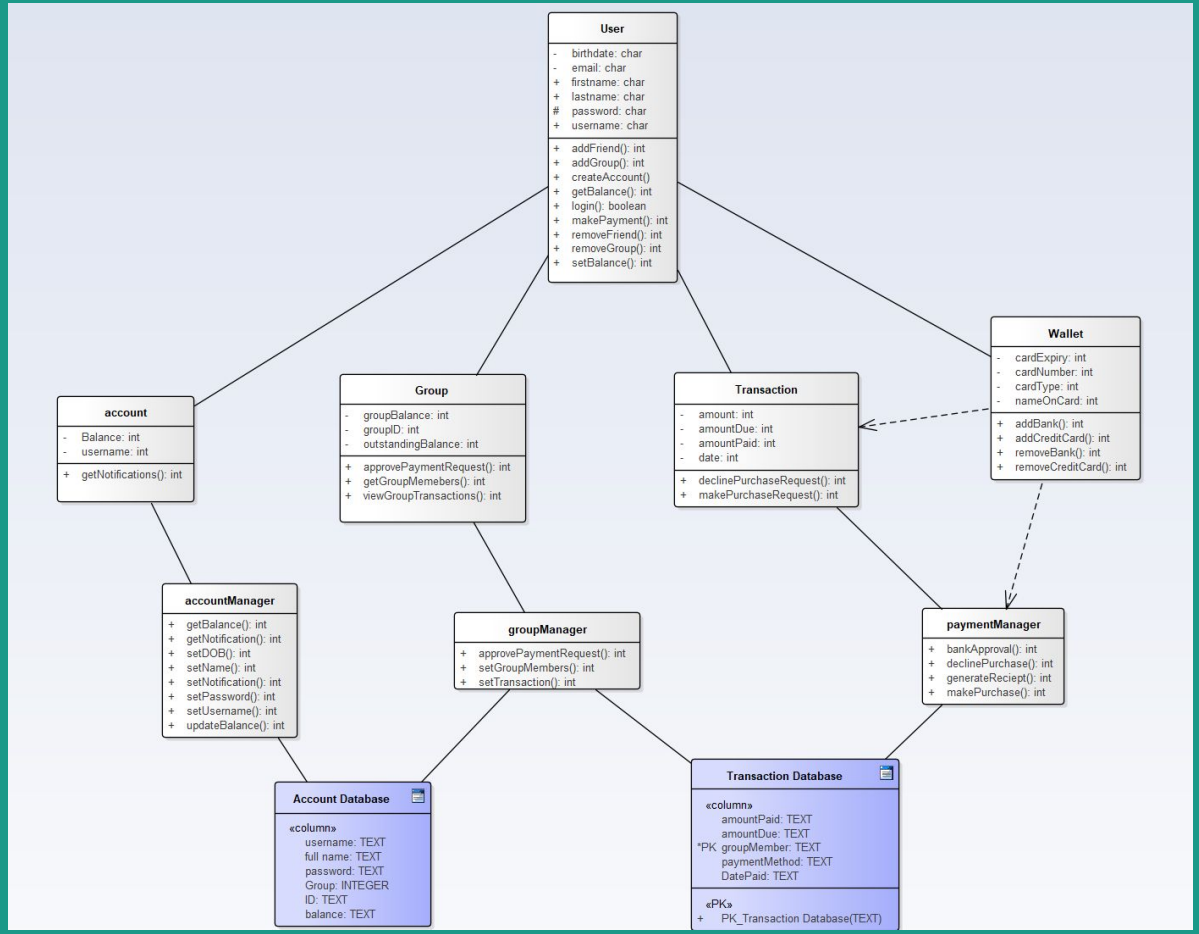
Use Case Name:	**Manage transaction**
Participating actor:	Customer
Entry condition:	Transaction has been created between 2 or more customers.
Exit condition:	Any changes to the transaction that the customer makes is applied and saved.
Event flow:	1. Customer clicks the manage transaction option 2. Customer makes changes to transaction (Cost, splitting method, etc.) 3. Customer saves changes made to transaction.

Use Case Name:	**Make payment**
Participating actor:	Customer/ Bank
Entry condition:	Transaction has been created between 2 or more customers.
Exit condition:	The customer's portion of a transaction is payed via their defined payment method
Event flow:	1. Customer selects transaction 2. Customer confirms the amount owed and pays 3. Money is taken via customers defined payment method 4. Transaction updates that the customer has paid their share.

Use Case Name:	**Manage group**
Participating actor:	Customer/ Users
Entry condition:	Create a group with friends
Exit condition:	Group successfully gets created
Event flow:	1. Select users from the friends list 2. Create a group 3. Create a name for the group 4. Group successfully gets created

Use Case Name:	**Manage friends**
Participating actor:	Customer/User
Entry condition:	Add a friend or delete a friend
Exit condition:	Friends list is updated
Event flow:	1. User will look up a friend by their Username 2. User will have the option to add a friend or delete a friend 3. Friend's list will be updated

Class Model


User
- birthdate: char
- email: char
+ firstname: char
+ lastname: char
# password: char
+ username: char
+ addFriend(): int
+ addGroup(): int
+ createAccount()
+ getBalance(): int
+ login(): boolean
+ makePayment(): int
+ removeFriend(): int
+ removeGroup(): int
+ setBalance(): int
Wallet
- cardExpiry: int
- cardNumber: int
- cardType: int
- nameOnCard: int
+ addBank(): int
+ addCreditCard(): int
+ removeBank(): int
+ removeCreditCard(): int
account
- Balance: int
- username: int
+ getNotifications(): int
Group
- groupBalance: int
- groupID: int
- outstandingBalance: int
+ approvePaymentRequest(): int
+ getGroupMemebers(): int
+ viewGroupTransactions(): int
Transaction
- amount: int
- amountDue: int
- amountPaid: int
- date: int
+ declinePurchaseRequest(): int
+ makePurchaseRequest(): int
accountManager
+ getBalance(): int
+ getNotification(): int
+ setDOB(): int
+ setName(): int
+ setNotification(): int
+ setPassword(): int
+ setUsername(): int
+ updateBalance(): int
groupManager
+ approvePaymentRequest(): int
+ setGroupMembers(): int
+ setTransaction(): int
paymentManager
+ bankApproval(): int
+ declinePurchase(): int
+ generateReciept(): int
+ makePurchase(): int
Account Database
«column»
username: TEXT
full name: TEXT
password: TEXT
Group: INTEGER
ID: TEXT
balance: TEXT
Transaction Database
«column»
amountPaid: TEXT
amountDue: TEXT
*PK groupMember: TEXT
paymentMethod: TEXT
DatePaid: TEXT
«PK»
+ PK_Transaction Database(TEXT)

Class Overview

- User Class (1)
  - Account Class (1)
  - Group Class (0..*)
  - Transaction Class (0..*)
  - Wallet Class (1)
- Management classes
- Databases