Bill Share app

Yuning Xie, Yizhuang Peng, Baiwei Sun

1 introduction:

This document is showing a “Bill share app”. It shows all the basic requirements for the app and it also shows some design details for the app.
The Bill share app can help people manage their bills better. Anyone with the app can simply upload the bill and select their friends from their contact list and then, the app will do all the rest. The app is competitive because it introduces new features in bill confirming steps that can protect the owner and the debitor better. The Bill share app also supports users to create chat groups and send text and pictures in those chat groups. This feature helps users communicate better about their bill.

- The application runs as a human readable interface on Android Mobile phones.
- The application provides some kind of basic GUI for easy use of users. The GUI must provide basic utilities like menus, buttons, panes, and provide function of keyboard for easy control on a touch screen
- The application runs online and requires a stable server to store some data
- The application runs on some version of android device, the source code for this application is in JAVA

2 Features:

Create account:

This application supports users to create their own account with email or phone number.

Login:

This application requires users to sign in to authenticate his/her accounts to grant their legal access to the service of the application.

Friend List:

This application supports users to have their own friend list. They can see friends’ names and avatars in the friend list menu.

Add Friend through email or phone number:

Users can send friend requests to other users by entering their email address or phone number. After the requests is confirmed, they will see the new friend in their friend list

Friend QR code:

Each user has their own unique friend QR code, other user can send friend request by scanning QR code

Add friend through QR code:

Users can send friend requests by scanning other people’s QR code. After the requests is confirmed, they will see the new friend in their friend list

Group List:

This application supports users to have their own Group chat list. They can see the Group’s name and small avatar of some members in the group

Create New Group:

This application supports users to create new group chats by selecting some friends in their friend list. The user that created the group will be the groups’ administrator.

Add new member to the group:

Users can add new members to the group by sending invite requests to them. Administrators can control who can send invitations to other users in the group.

Delete member to the group:

Administrators can kick existing users from the group. This feature requires no requests step

Leave a group:

Every member in a group can choose to leave the group at any time. After they leave the group, the group administrator will receive a message.

Bill form:

The bill will be exhibited as “total amount” the owner has paid and several debts that each debitor should pay.

Create a Bill request:

Every member in a group can create a bill request. They can enter a short description to the bill and enter the total amount of this bill request. They can also select which member in the group to be included in the bill request as the debtor and enter the amount they should pay as well. The debts will be sent to these debtors as notifications and they will be set to “unaccepted” status on their pages.

Bill request confirming stage:

After the creation of a bill request. The App will calculate how much money each debitor should pay and send them a request to confirm. For those who accept the bill, new indebts will be established, and the debt will become “unpaid” status. For those who decline the bill, no debts will be brought up and they will appear as “declined” status.

After the confirmation, the request creators will receive a message and then they can monitor the bill. After some time, some debtors still haven’t confirmed their request, the creator will also receive a message to notify them.

Bill Paying step:

After the debitor accept the bill, he/her will be able to see the amount needed to pay and the status of his debt as “unpaid”. If the debtor pays the bill, he/she will push the button “Pay the bill” to change the status into “checking Payment” and notify the owner of the bill.

Bill monitoring step:

After the bill request is confirmed, the bill will be created. The owner of this bill can monitor the process of this bill by checking which debtor has paid the bill and their payment time and who hasn’t paid the bill yet.

Bill Settlement step:

If one of the debts in the bill has been paid, the status of this debt in the bill will become “waiting for checking” and the owner can push “confirm” to settle the debt after confirming receiving the payment from the debitor. After the confirmation of the owner is detected, the debt in the debtor's page will appear as “paid” status.

Once all debts have been settled for a bill, the status of the bill will become “accomplished”.

Send notification to debtor:

The bill owner can select all debtors or some of them that haven't paid the bill and send some simple message to them directly to notify them.

Send message in group:

This application allows user to send simple messages in group chat and every member in the group can see the message

Send pictures in group:

This application allows user to send some pictures in group chat and every member in the group cae see the picture

Edit Bill description:

The application allows users to enter some text description about the bill when creating one. Every selected bill debtor can see the description when they see this bill

Upload Bill receipt:

The application allows users to upload some picture recipes about the bill when they create one. Every selected bill debtor can see the picture recipes when they see the bill

Change Password:

The application allows users to change their password.

Update user’s profile:

The application allows users to change their user’s profile by uploading a new avatar picture, editing their nickname or changing their registered email address or phone number.

View transaction history:

The application allows users to check their transaction history including amount of money sent or received, the debtor or owner their money goes and the bill starting date or ending date.

3 Detail Utility Interface:

Login Page(figure 1):

login():

Input: The user can enter their username and password.

Output: If the input matches a user in the database, they can enter the main page of the application. If the input mismatch, they can see an error message like “incorrect password”.

gotosignup():

Input: select Create One button.

Output: they will enter account sign up page

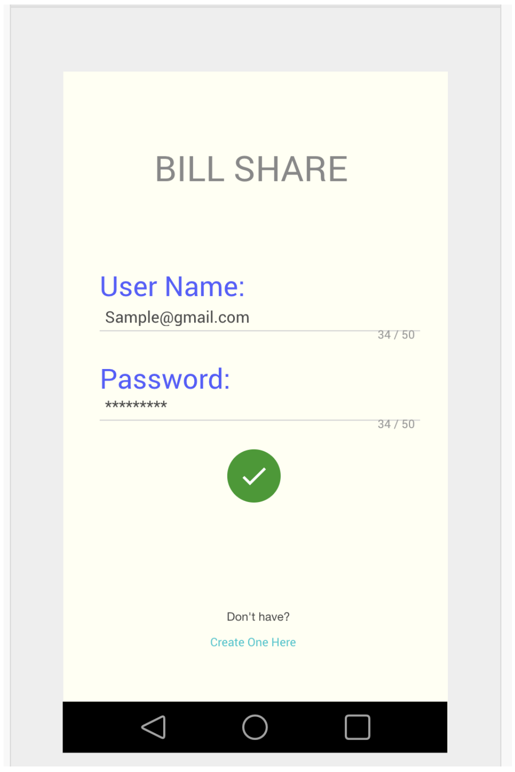


figure1: Login page

SignUp Page(figure 2):

signup():

Input: The user can enter their email or phone number and the password they want.

Output: If the email or phone number doesn’t match a user in the database, and the password matches the password rule. A new account will be created and the user will be redirected to the main UI page. If the input match a user in the database of the password doesn’t pass the password rule, the user will a error message

gotologin():

Input: user select “X” button.

Output: user will be redirected to login page

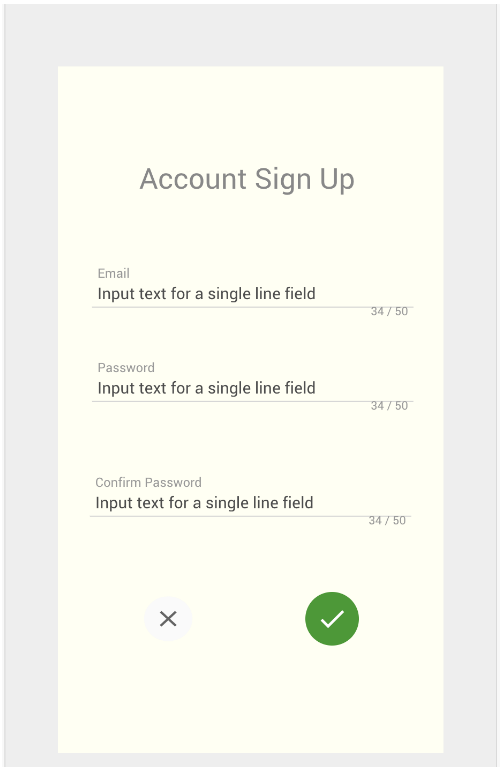

Account Sign Up
Email
Input text for a single line field
34 / 50
Password
Input text for a single line field
34 / 50
Confirm Password
Input text for a single line field
34 / 50

figure2: Sign up page

Group List Page(figure 3):

gotoFriend():

Input:user click the friend button in the bellow

Output: user will be redirected to friend list page

gotoSetting():

Input: user click the setting button in the bellow

Output: user will be redirected to setting page

createNewGroup():

Input: user click the new group button on the right bottom

Output: users can create a new group by selecting some friends or create an empty group and invite members later.

deleteGroup():

Input: L User click the … button on the right top corner and click delete group in drop down menu

OutputL user can select which group to delete in their group chat list

enterChat():

Input: user select any one chat button in the middle

Output: user will be redirected to that selected group chat page

searchChat():

Input: user click the search button on the top right corner

Output: user can enter some input and search for any matching in group name, chat message.

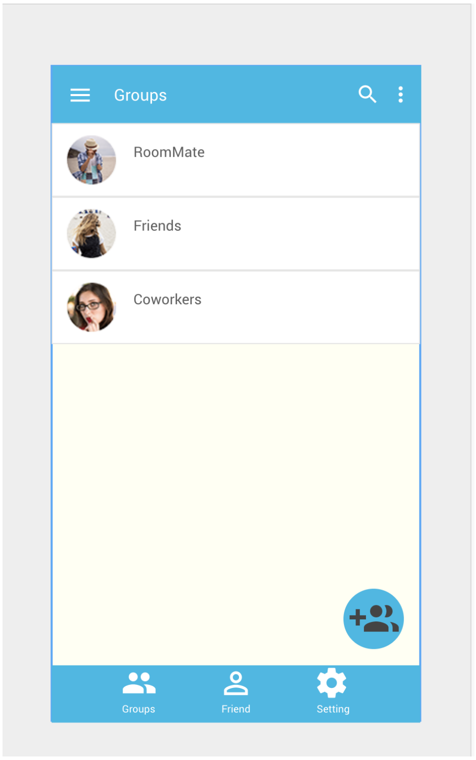


figure3: Group List page

Friend List Page(figure 4):

gotoGroup():

Input: user clock the group button in the bellow

Output: user will be redirected to Group list page

gotoSetting():

Input: user click the setting button in the bellow

Output: user will be redirected to setting page

addNewFrined():

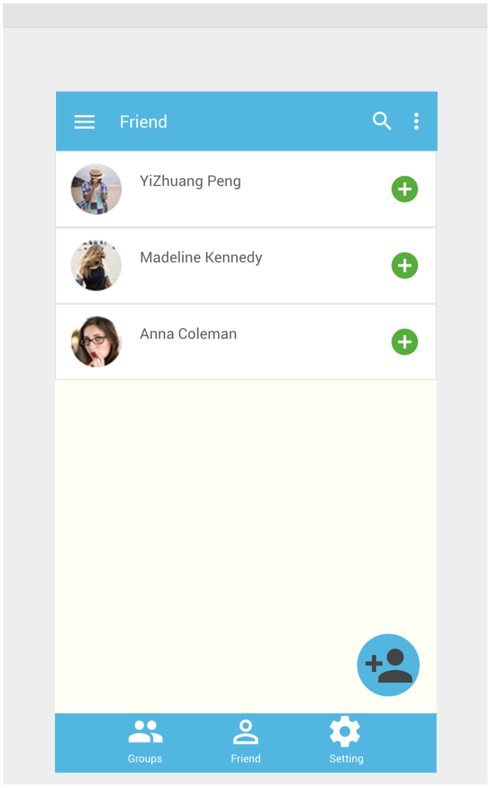


figure4: Friend List page

Input: User click the new friend button on the right bottom
Output: User can add new friend by enter their email or phone number to send requests to them.

addNewFriendwithQRcode():
Input: User click the … button on the right top corner and click add friend through QR code in drop down menu
Output: Application will pop up camera to scan QR code, if the QR code is a valid friend QR code, send friend request to them.

deleteFriend():
Input: User click the … button on the right top corner and click delete friend button in drop down menu
Output: user can select which friend to delete from their frind list:

viewProfile():
Input: User click any friend button:
Output: User will be redirected to their friends profile page

searchFriend():
Input: user click the search button on the top right corner
Output: user can enter some input and search for any matching in friend's name, email address or phone number.

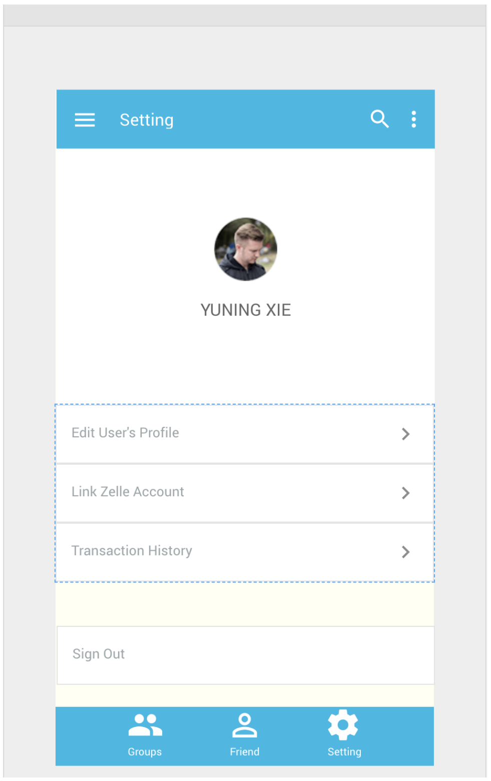


figure5: Setting page

Setting Page(figure 5):

gotoGroup():
Input: user clock the group button in the bellow
Output: user will be redirected to Group list page

gotoFrined():
Input:user click the friend button in the bellow
Output: user will be redirected to friend list page

editUserProfile():
Input: user click the Edit user's profile button
Output: user will be redirected to Edit user's profile page.

linkZelleAccount():
Input: user click the Link Zelle Account
Output: user will be redirected to Zelle's Page

viewTransactionHistory():
Input: user click the Transaction History button

Output: user will see lists of their transactions including the amount of money they paid or received depending on whether they are a bill owner or debtor. The creation data of the bill and the finishing date of the transaction

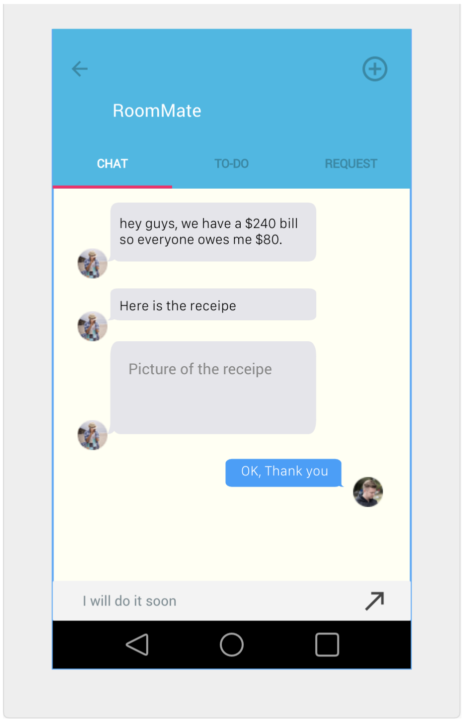
signout():
Input: user click the signout button
Output: user will be signed out and will be redirected to login page.

Chat Detail page(figure 6):

gotoTodo():
Input: user click the To-do button on the up middle.
Output: user will be redirected to To-do list page of this group

gotoRequest():
Input: user click the request button on the up right
Output: user will be redirected to bill creation page to create a bill inside the group

sendMessage():
Input: user click the send button on the right bottom
Output: the message in the sending text box will be sent to the group chat.



hey guys, we have a $240 bill so everyone owes me $80.

Here is the receipe

Picture of the receipe

OK, Thank you

I will do it soon

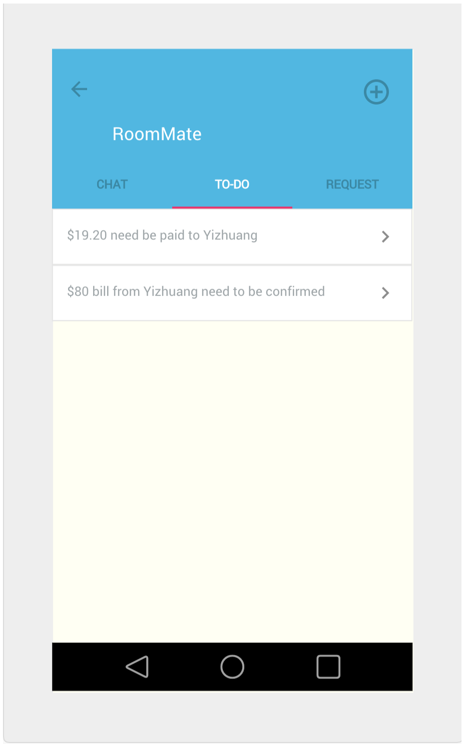
figure6: Chat detail page

To-do list page(figure 7):

gotoChat():
Input: user click the chat button on the up left
Output: user will be redirected to the chat detail page

gotoRequest():
Input: user click the request button on the up right
Output: user will be redirected to bill creation page to create a bill inside the group

enterbill():
Input: user click any one bill in the middle of the page
Output: user will be able to see the bill details including (owner, creation date, amount of money) and they can decide to accept or reject.



$19.20 need be paid to Yizhuang

$80 bill from Yizhuang need to be confirmed

figure7: To-Do list page

Bill creation page(figure 8):

gotoChat():
Input: user click the chat button on the up left
Output: user will be redirected to the chat detail page

gotoTodo():
Input: user click the To-do button on the up middle.
Output: user will be redirected to To-do list page of this group

sendBillRequest():
Input: user clock the request equally split bill button in middle
Output: application will collect the information user entered and let user to select bill debtor in the group, then application will send bill creation request to all the selected member

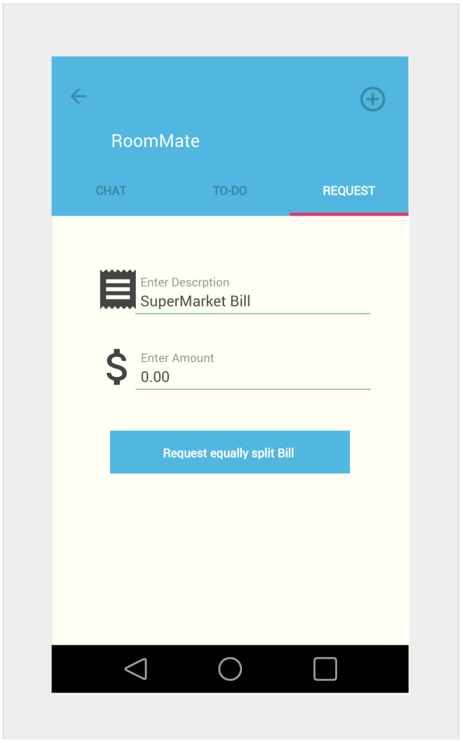


figure8: bill creation page

Friend profile page(figure 9):

ViewTransactionHistory():
Input: user click the view transaction history with this friend button in the middle
Output: user will be able to see all the transactions with this friend in the before

deleteFriend():
Input: user click the delete friend button in the below():
Output: the friend will be deleted from the user's friend list

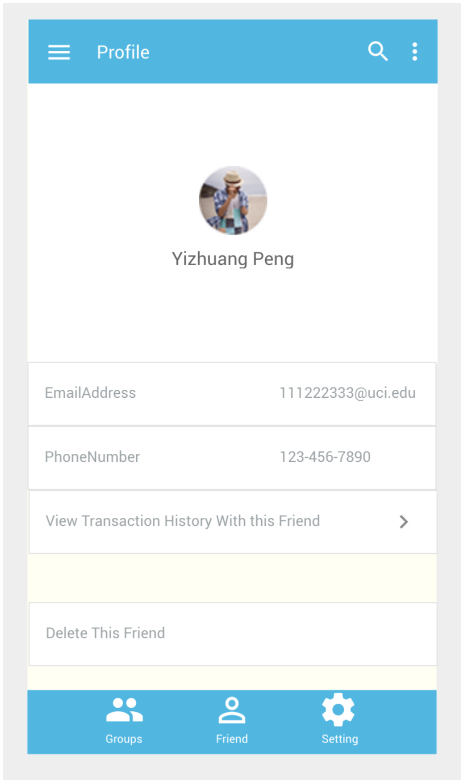


figure9:friend profile page

Edit Profile page(figure 10):

ChangeName():
Input: user click the change name button in the middle
Output: user can enter their new first name and last name and application will save them into database

uploadNewAvatar():
Input: user click the upload new avatar button in the middle
Output: user can upload a new avatar and application will save the picture into database

updateEmailAddress():
input: user click the update email address button in the middle
Output: user can enter a new email address and application will save it into database

updatePhoneNumber():
input: user click the update phone number button in the middle
Output: user can enter a new phone number and application will save it into database

ChangePassword():
input: user click the change password button in the middle
Output: user can enter a new password and application will save it into database

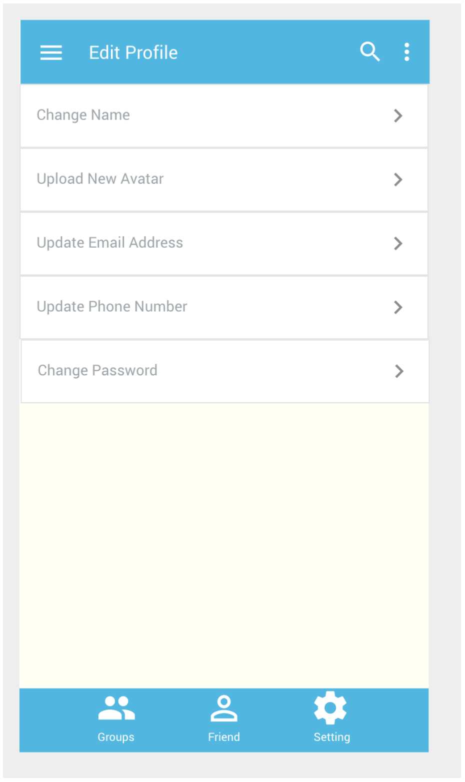
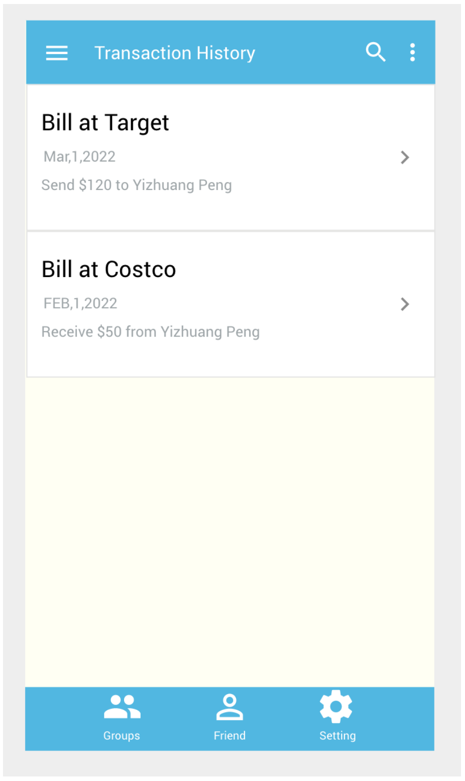


figure10: edit profile page

Transaction history page(figure 11):

viewHistoryDetial():
Input: user click any one bill button in the middle
Output: user can view the selected bill details



Bill at Target
Mar,1,2022
Send $120 to Yizhuang Peng

Bill at Costco
FEB,1,2022
Receive $50 from Yizhuang Peng

figure11: transaction history page

4 Architecture:

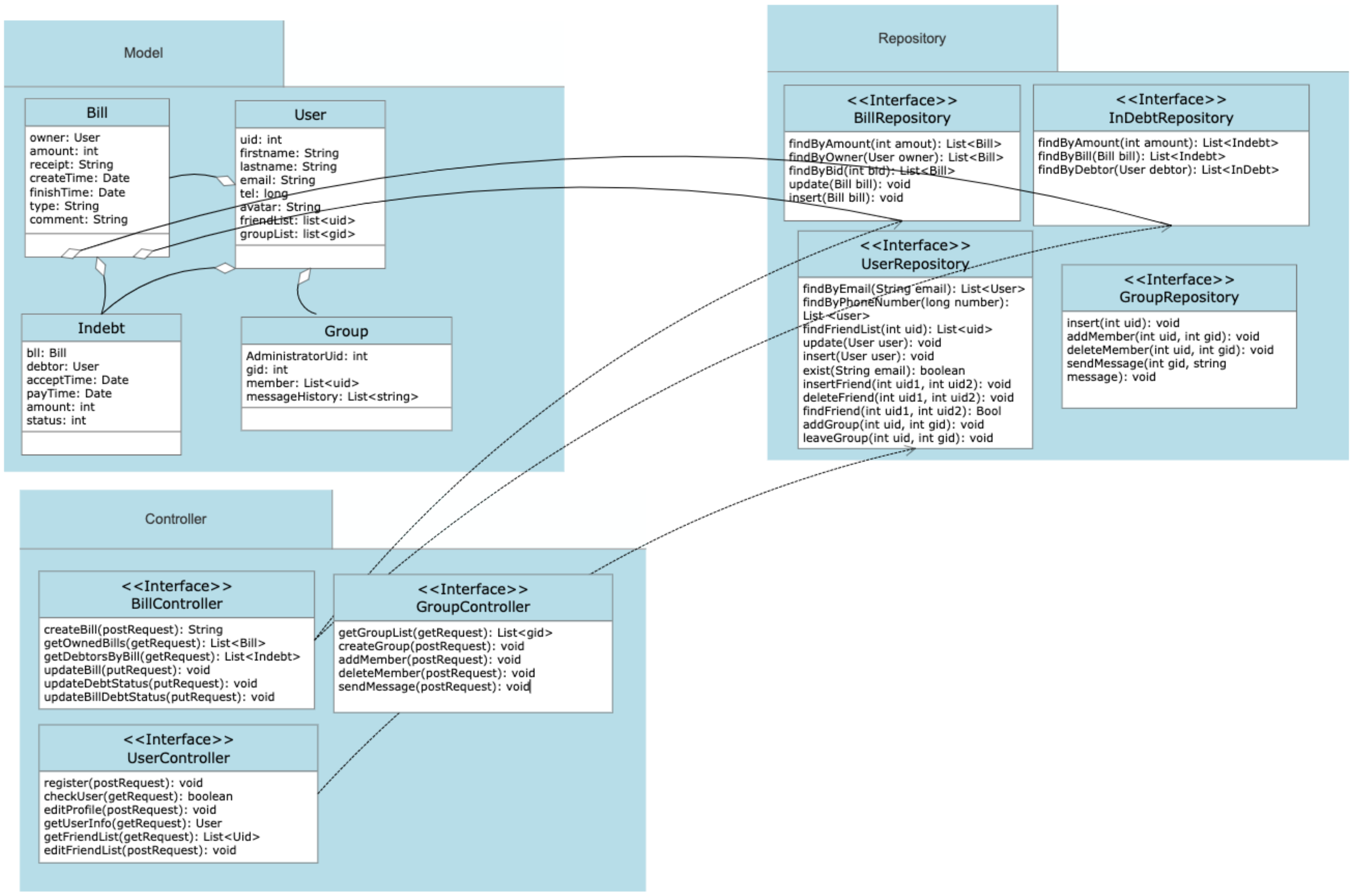

Model
Bill
owner: User
amount: int
receipt: String
createTime: Date
finishTime: Date
type: String
comment: String
User
uid: int
firstname: String
lastname: String
email: String
tel: long
avatar: String
friendList: list<uid>
groupList: list<gid>
Indebt
bll: Bill
debtor: User
acceptTime: Date
payTime: Date
amount: int
status: int
Group
AdministratorUid: int
gid: int
member: List<uid>
messageHistory: List<string>
Repository
<<Interface>>
BillRepository
findByAmount(int amout): List<Bill>
findByOwner(User owner): List<Bill>
findByBid(int bid): List<Bill>
update(Bill bill): void
insert(Bill bill): void
<<Interface>>
InDebtRepository
findByAmount(int amount): List<Indebt>
findByBill(Bill bill): List<Indebt>
findByDebtor(User debtor): List<InDebt>
<<Interface>>
UserRepository
findByEmail(String email): List<User>
findByPhoneNumber(long number): List<user>
findFriendList(int uid): List<uid>
update(User user): void
insert(User user): void
exist(String email): boolean
insertFriend(int uid1, int uid2): void
deleteFriend(int uid1, int uid2): void
findFriend(int uid1, int uid2): Bool
addGroup(int uid, int gid): void
leaveGroup(int uid, int gid): void
<<Interface>>
GroupRepository
insert(int uid): void
addMember(int uid, int gid): void
deleteMember(int uid, int gid): void
sendMessage(int gid, string message): void
Controller
<<Interface>>
BillController
createBill(postRequest): String
getOwnedBills(getRequest): List<Bill>
getDebtorsByBill(getRequest): List<Indebt>
updateBill(putRequest): void
updateDebtStatus(putRequest): void
updateBillDebtStatus(putRequest): void
<<Interface>>
GroupController
getGroupList(getRequest): List<gid>
createGroup(postRequest): void
addMember(postRequest): void
deleteMember(postRequest): void
sendMessage(postRequest): void
<<Interface>>
UserController
register(postRequest): void
checkUser(getRequest): boolean
editProfile(postRequest): void
getUserInfo(getRequest): User
getFriendList(getRequest): List<Uid>
editFriendList(postRequest): void

figure12: Architectural Diagram

Model:

Model is data that needs to be stored in the server. It contains four parts. Bill, User and Indebt.

Bill:

Bill model is created when a bill is created.

It will store owner(the owner to the bill), amount(the total amount of the bill), receipt (the path to the receipt picture), createTime(create time of the bill), finishTime(finishTime of the bill), type(the type of the bill like living expense or shopping), commnet(some description about this bill)

It will reference the User model to track who is the owner of the bill

Indebt:

Indebt is created when a bill is created. The difference between Indebt model and bill model is When a bill includes more than one debtor, Only one bill but several Indebt model will be created.

It will store bill(the Bill of the Indebt), debtor(Who is the debtor of the Indebt), acceptTime(the time the user confirm the bill request), Paytime(the time when the debtor pay the bill), amount(the amount that the debtor should pay) and status (an int indicate the status of the indebt model).

Status follow the following convention:

-1: the debtor declined to claim he/she owes the bill owner;
0: the debt is created, waiting for decline/accept of the debtor;
1: the debt is accepted by the debtor;
2: debtor paid the debt;
3: bill owner confirmed the payment from debtor

It will reference to Bill model to track which bill the debtor is indebt for. It will reference to User model to track who is the debtor of the indebt model.

User:

The User model is created when a user finishes a signup function.

It will store uid(an user id that is unique for every user model) first name, last name(the first name and last name of the user), email(the entered email for this user), phone number(the entered phone number for this user), avatar(the path to the avatar picture), friendList(store the uid of all the added friends), groupList(store the gid of all the group that the user is a member.)

Group:

The Group model is created when a user creates a new group, the user that creates the group will be the administrator of the group.

It will store AdministratorUid(the uid of the administrator), gid(an group id that is unique for every group model), member(a list of uid that the user is a member of the group), MessageHistory(a list of string or path to pictures that was sent in the group)

repository:

Repository is the interface of the database. Any creation, lookup and modification must call these functions to make changes to the database. It includes Four parts and all control the corresponding model.

BillRepository:

BillRepository contains all the access to Bill model. Including different kind of look up and different kind of modification

FindByAmout(int amount): it will look up the bill database that match the given amount and return the list of all the matching bill

FindByOwner(User Owner):: it will look up the bill database that match the given Owner and return the list of all the matching owner

FindByBid(int bid): it will look up the bill database that match the given bid and return the Bill

Update(Bill bill): it will look up the bid of the input bill and replace the found bill in the database with the new given bill

Insert(Bill bill): it will insert the given bill data into the database as a new data

IndebtRepository:

IndebtRepository contains all the access to Indebt model. Including different kind of look up and different kind of modification

FindByAmount(int amount): It will look up the bill database and find all the matching bill and then the list of all their inherent indebt

FindByBill(Bill bill): It will look up inherit database that match the given bill and return the list of all the matching indebt

FindByDebtor(User Debtor): It will look up inherit database that match the given debtor and return the list of all the matching indebt

UserRepository:

UserRepository contains all the access to User model. Including different kind of look up and different kind of modification

findByEmail(string email): it will look up the user database that match the given email address and return the list of all the matching user

findByPhoneNumber(long number): it will look up the user database that match the given phone number and return the list of all the matching user

findFriendList(int uid): it will look up the user database that match the given uid and return the corresponding friend list

update(User user): it will look up the uid of the input User and replace the found User in the database with the new given User

Insert(user user): it will insert a the given User data into the database as a new data

exist(User user): it will try to look up the uid of the input User and return a boolean of whether this user exists in the database.

insertFriend(int uid1, int uid2): it will look up the user database that matches the given uid1 and insert uid2 into the friendList of the uid1

deleteFriend(int uid1, int uid2): it will look up the user database that matches the given uid1 and delete uid2 from the friendList of the uid1

findFriend(int uid1, int uid2): it will look up the user database that matches the given uid1 and try to look up uid2 from the friendList of the uid1

addGroup(int uid, int gid): it will look up the user database that matches the given uid and the given gid to the groupList of the uid.

leaveGroup(int uid, int gid): it will look up the user database that matches the given uid and the given gid to the groupList of the uid.

GroupRespository:

insert(int uid): it will insert a new group data into the database and set the administratorUid as the given uid.

addMember(int uid,int gid): it will look up the group database that match the given gid and insert the given uid into the member

deleteMember(int uid, int gid): it will look up the group database that match the given gid and delete the given uid into the member

sendMessage(int gid, string message): it will look up the group database that match the given gid and add the given message to the message history

Controller:

Controller is the real connection between the UI and the backend data server, All interactions between front-end and server will call some member of the controller and those controller members will then use repositories to get access to the different models.

UserController:

UserController is aimed to deal with services related to users.

register(): it will collect all the information entered by user in the UI and call the Insert() function to create new User in the database

checkUser(): it will collect information entered by the user in the UI and call the exist() function to look up the user in the database.

editProfile(): It will collect information entered by the user in the UI and call update() to modify the found User in the database

getUserInfo(): it will collect information entered by the user in the UI and call findbyEmail() or findbyPhoneNumber() to look up the User in the database

getFriendList(getRequest): it will use the user’s Uid to get the all the uid in the user’s friend list and show them in the UI

editFriendList(postRequest): it will collect information entered by the user to call the corresponding insertFriend() or deleteFrined()

BillController:

BillController is the API that application will use when relating to the Bill or Indebt. It will mainly updating and checking the bill model and Indebte model by inserting new data or modifying them

createBill(): it will collect all the information entered by user in the UI and call insert function to create new Bill in the database

getOwnedBills(): it will try to look up all the bill that the users is the owner and show all the bills in the UI

getDebtorsByBill(): it will collect information entered by the user and look up all the debtors of bill and show all the debtors in the UI

updateBill(): it will collect information entered by the user and call update() to change the Bill in the database
updateDebtStatus(): when a user paid their bill, the application will automatically collect that and update the Indebt model status
updateBillDebtStatus(): when all user paid their bill, the application will automatically collect that and update the bill in the database to make it finished

GroupController:
getGroupList(): it will use the user’s Uid to get the all the gid in the user’s group list and show them in the UI
createGroup(): it will collect information entered by the user and call insert() to create a new group data in the database. The user that created the group will the be group administrator
addMember(): it will collect information entered by the user and call addmember() and addgroup() to change both the group data and the user data in the database
deleteMember(): it will collect information entered by the user and call deletemember() and leavegroup() to change both the group data and the user data in the database
sendMessage(): it will collect the text entered by the user and call sendMessage() to keep the message in the database.

5: Feature Map:

Below is the feature map that maps the feature to the API. It will show what function need to be called to realize the designed feature of the map

Feature: create account
Subcomponents: UserController.register()

Feature: login
SubComponents: JWT package

Feature: Friend List
SubComponents: UserController.getFrinedList()

Feature: Add Friend through email or phone number
SubComponents: UserController.CheckUser(), UserController.editFriendList()

Feature: Friend QR code
SubComponents:None

Feature: Add friend through QR code
SubComponents:UserController.CheckUser(), UserController.editFriendList()

Feature: Group List
SubComponents:GroupController.getGroupList()

Feature: Create New Group
SubComponents:GroupController.createGroup()

Feature: Add new member to the group
SubComponents:GroupController.addMember()

Feature: Delete member to the group
SubComponents:GroupController.deleteMember()

Feature: Leave a group
SubComponents:GroupController.deleteMember()

Feature: Create a Bill request
SubComponents: BillController.createBill()

Feature: Bill request confirming stage
SubComponents:

Feature: Bill monitoring step
SubComponents:BillController.getOwnedBills()

Feature: Send notification to debitor
SubComponents:BillController.getDebtorsByBill()

Feature: Send message in group
SubComponents:GroupController.SendMessage()

Feature: Send pictures in group
SubComponents:GroupController.SendMessage()

Feature: Edit Bill description
SubComponents:GroupController.updateBill()

Feature: Upload Bill receipt
SubComponents:GroupController.updateBill()

Feature: Linking Bank(or zelle) account
SubComponents:None

Feature: Change Password
SubComponents: UserController.editProfile()

Feature: Update user's profile
SubComponents: UserController.editProfile()

Feature: View transaction history
SubComponents: BillController.getOwnedBills(), BillController.getDebtorsByBill()

6 Reference:

SpringBoot
https://docs.spring.io/spring-boot/docs/current/reference/html/
https://docs.spring.io/spring-boot/docs/current/api/

SpringData – JPA
https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference

Spring Framework WebSocket Support
https://docs.spring.io/spring-framework/docs/4.3.x/spring-framework-reference/html/websocket.html

WebSocket Authorization
https://docs.spring.io/spring-security/reference/servlet/integrations/websocket.html#websocket-authorization

STOMP JS
https://stomp-js.github.io/stomp-websocket/codo/extra/docs-src/Usage.md.html

Ionic docs
https://ionicframework.com/docs

ReactJS
https://reactjs.org/docs/getting-started.html#learn-react

JsonWebToken.io
https://github.com/jwtk/jjwt