Bank System Specification Document

Author	Linshu Yang
Group	Team 5
Date	2022/5/30

Catalog

1. System Architecture

1.1 System Composition

The Bank System consists of independent frontend and backend, controlled by two different controllers, using the Model-View-ViewModel structure.

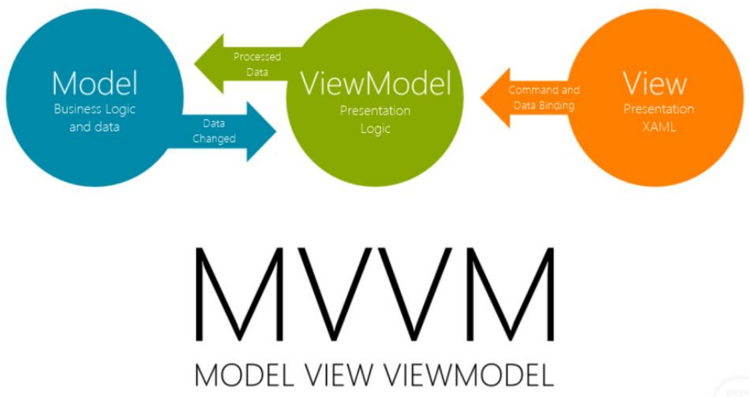


Fig. 4.1. MVVM Structure

Modules

1. Model
   The backend of the Bank System, receive requests from the frontend and maintain the database accordingly.
2. ViewModel
   The frontend controller of the Bank System, making communication with the backend and giving responses to users’ requests.
3. View
   Present information and interfaces to users, make directly interactions with them.

1.2 Client-Server Architecture

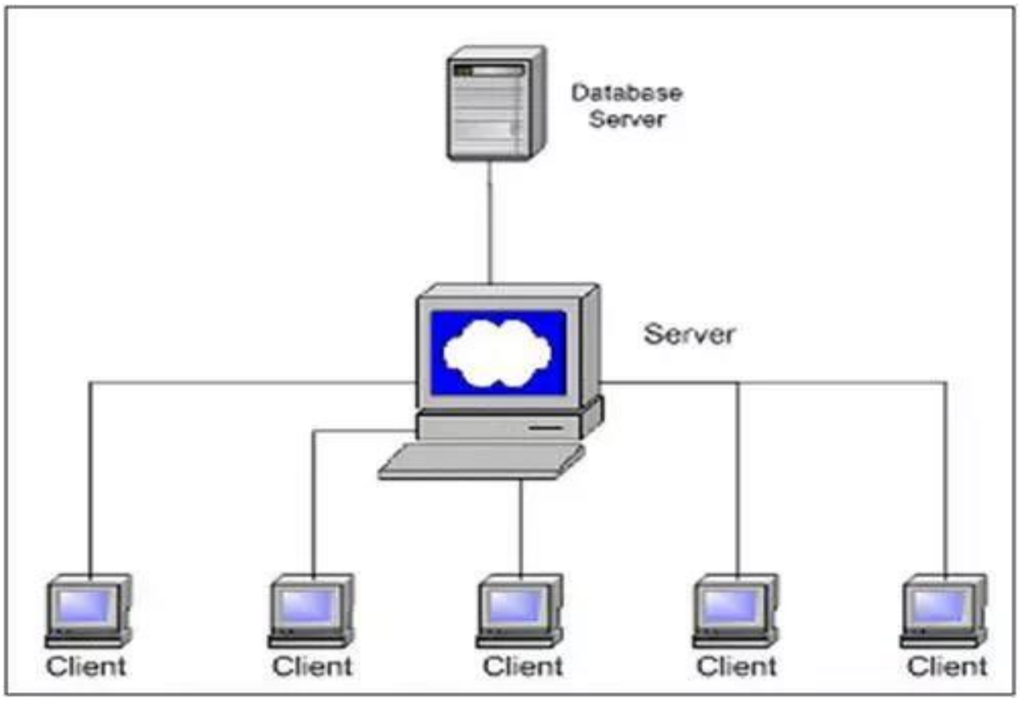


Fig. 4.2. Client-Server Architecture

The client-server model of the Bank System is a distributed application structure that partitions tasks or workloads between server, the providers of the resources and clients, the service requesters. In MATLAB environment, the client finishes the workload of user interactions and get resources from the server. And the server covers tasks related to the database, directly communicate to the database server (SQLite3 drivers) provided by MATLAB to maintain the database.

1.3 Class Diagram Design

Note that all the types mentioned below follow to the JavaScript type system.

1.3.1 Backend Class Design

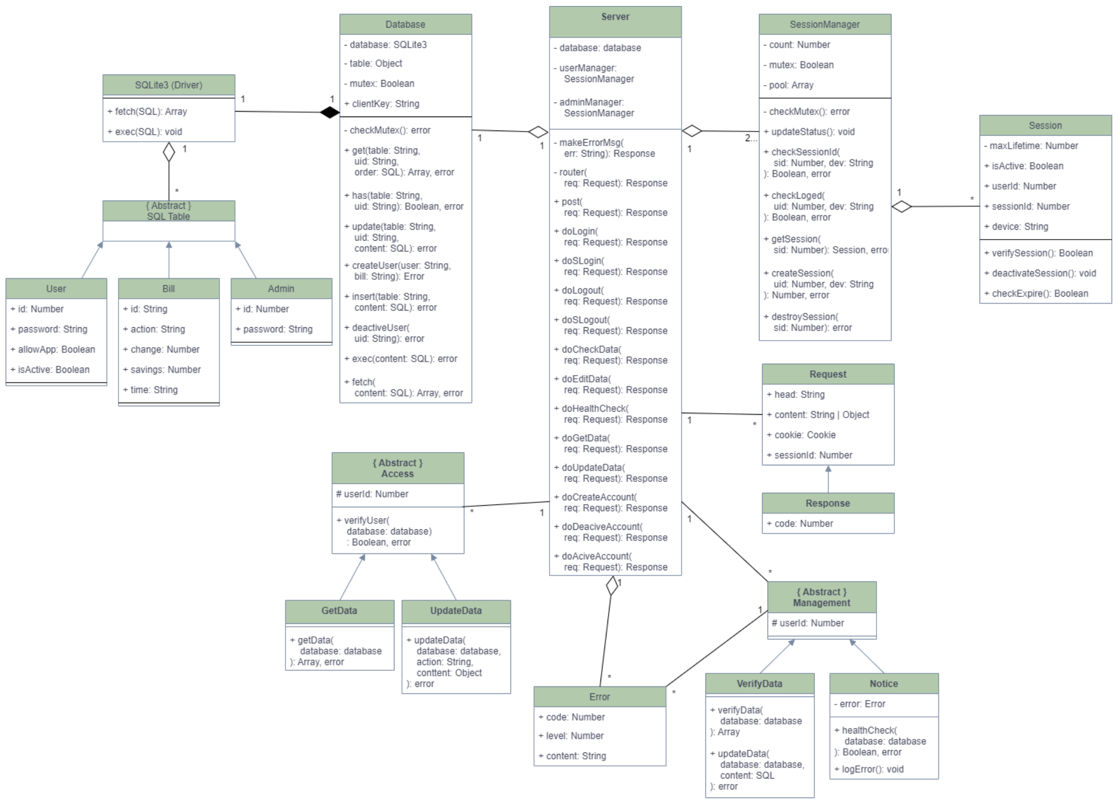

SQLite3 (Driver)
+ fetch(SQL): Array
+ exec(SQL): void
{ Abstract } SQL Table
User
+ id: Number
+ password: String
+ allowApp: Boolean
+ isActive: Boolean
Bill
+ id: String
+ action: String
+ change: Number
+ savings: Number
+ time: String
Admin
+ id: Number
+ password: String
Database
- database: SQLite3
- table: Object
- mutex: Boolean
+ clientKey: String
- checkMutex(): error
+ get(table: String, uid: String, order: SQL): Array, error
+ has(table: String, uid: String): Boolean, error
+ update(table: String, uid: String, content: SQL): error
+ createUser(user: String, bill: String): Error
+ insert(table: String, content: SQL): error
+ deactiveUser(uid: String): error
+ exec(content: SQL): error
+ fetch(content: SQL): Array, error
Server
- database: database
- userManager: SessionManager
- adminManager: SessionManager
- makeErrorMsg(err: String): Response
- router(req: Request): Response
+ post(req: Request): Response
+ doLogin(req: Request): Response
+ doSLogin(req: Request): Response
+ doLogout(req: Request): Response
+ doSLogout(req: Request): Response
+ doCheckData(req: Request): Response
+ doEditData(req: Request): Response
+ doHealthCheck(req: Request): Response
+ doGetData(req: Request): Response
+ doUpdateData(req: Request): Response
+ doCreateAccount(req: Request): Response
+ doDeaciveAccount(req: Request): Response
+ doAciveAccount(req: Request): Response
SessionManager
- count: Number
- mutex: Boolean
- pool: Array
- checkMutex(): error
+ updateStatus(): void
+ checkSessionId(sid: Number, dev: String): Boolean, error
+ checkLoged(uid: Number, dev: String): Boolean, error
+ getSession(sid: Number): Session, erro
+ createSession(uid: Number, dev: String): Number, error
+ destroySession(sid: Number): error
Session
- maxLifetime: Number
+ isActive: Boolean
+ userId: Number
+ sessionId: Number
+ device: String
+ verifySession(): Boolean
+ deactivateSession(): void
+ checkExpire(): Boolean
Request
+ head: String
+ content: String | Object
+ cookie: Cookie
+ sessionId: Number
Response
+ code: Number
{ Abstract } Access
# userId: Number
+ verifyUser(database: database) : Boolean, error
GetData
+ getData(database: database): Array, error
UpdateData
+ updateData(database: database, action: String, conttent: Object): error
Error
+ code: Number
+ level: Number
+ content: String
{ Abstract } Management
# userId: Number
VerifyData
+ verifyData(database: database): Array
+ updateData(database: database, content: SQL): error
Notice
- error: Error
+ healthCheck(database: database): Boolean, error
+ logError(): void
1
1
1
*
1
1
1
2...
1
*
1
*
1
*
1
*
1
*
1
*

Fig. 1.1. Backend Class Diagram

Modules

1. User
   Profiles of users stored in SQLite3. Field of this table includes id, password, allowApp and isActive.
2. Bill
   Bills of users stored in SQLite3. Field of this table includes id, action, change, savings, time.
3. Admin
   Profiles of admins stored in SQLite3. Field of this table includes id and password.
4. SQL Table
   Abstract class as the base class for specific tables, component of SQLite3.
5. SQLite3 (Driver)
   SQLite3 driver object provided by MATLAB Database Extension. Methods include fetch and exec.
6. Database
   Database class working as a proxy between the server and SQLite3, built to avoid direct use of SQL statement. Properties include database, table, mutex and clientKey. Methods include checkMutex, get, has, update, createUser, insert,

deactiveUser, exec and fetch.

7. GetData
   Handler for users to get data, with getData method.
8. UpdateData
   Handler for users to update data, with updateData method.
9. Access
   Abstract class as the base of specific user-access handlers with userId property and verifyUser method.
10. VerifyData
    Handler set for admins to check and edit data. Methods include verifyData and updataData.
11. Notice
    Handler set for server error management with error property. Methods include healthCheck and logError.
12. Management
    Abstract class as the base of specific admin-management handlers with property userId.
13. Error
    Class used as a uniformed interface for errors. Properties include code, level and content.
14. Session
    Class used to record users' or admins' login status. Properties include maxLifetime, isActive, userId, sessionId and device. Methods include verifySession, deactivateSession and checkExpire.
15. SessionManager
    Class built to manage sessions. Properties include count, mutex and pool. Methods include checkMutex, updateStatus, checkSessionId, checkLoged, getSession, createSession and destroySession.
16. Server
    Main controller of the backend. Properties include database, userManager, adminManager. Methods include makeErrorMsg, router, post, doLogin, doSlogin, doLogout, doSLogout, doCheckData, doEditData, doHealthCheck, doGetData, doUpdateData, doCreateAccount, doDeactiveAccount, doActiveAccount.

1.3.2 ATM Class Design

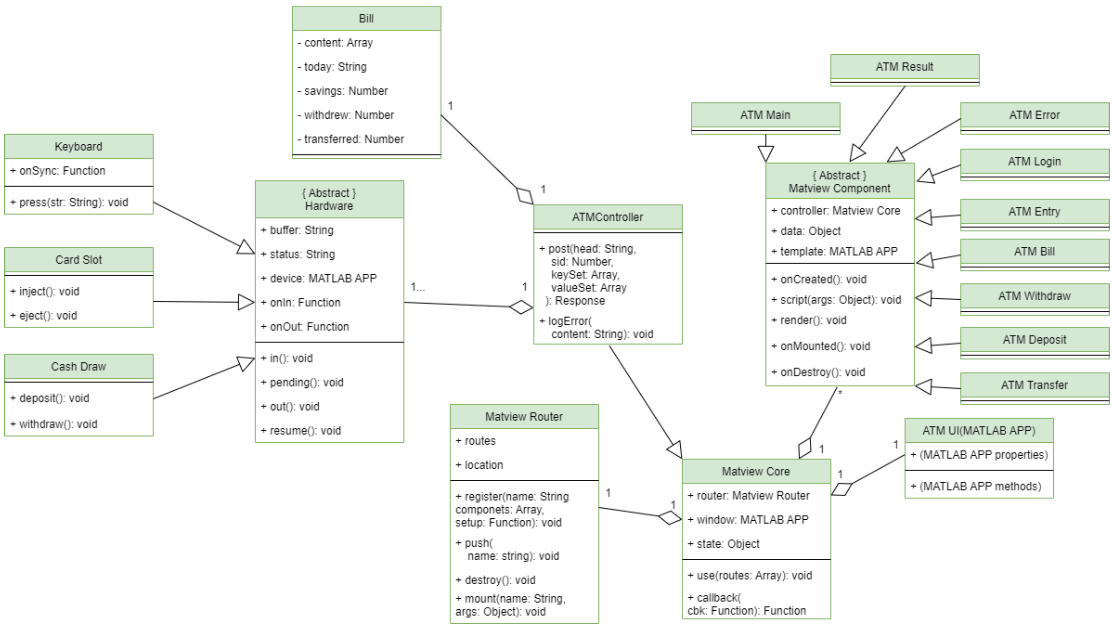


Fig. 1.2. ATM Class Diagram

Modules

1. Hardware
   Abstract class, as the base class of all virtual hardware used to simulate hardware in real life. Properties include buffer, status, device, onIn and onOut. Methods include in, pending, out and resume.
2. Keyboard
   Subclass of Hardware with extra property onSync and extra method press, used to simulate keyboards.
3. Card Slot
   Subclass of Hardware, used to simulate card slots. Extra methods include inject and eject.
4. Cash Draw
   Subclass of Hardware, used to simulate cash draws. Extra methods include deposit and withdraw.
5. Bill
   Bill class used to process bill table. Properties include bill, today, savings, withdrawed and transferred.
6. ATM UI (MATLAB APP)
   UI components based on MATLAB APP.
7. Matview Core
   Core class as the controller of Matview Router and other components. Properties include router, window and state. Methods include use and callback.
8. Matview Router

Router class built to change the visibility of the UI components, enable us to build a multi-page application in one page. Properties include routes and location. Methods include register, push, destroy and mount.

9. Matview Component

   Abstract class as a uniform component interface for customized components to interact with Matview Core. Properties include controller, data and template; Methods include onCreated, script, render, onMounted and onDestroy. In ATM there are nine components: ATM Main, ATM Result, ATM Error, ATM Login, ATM Entry, ATM Bill, ATM Withdraw, ATM Deposit and ATM Transfer.

10. ATMController

    Subclass of Matview core and main control of the ATM. Methods include post and logError.

1.3.3 APP Class Design

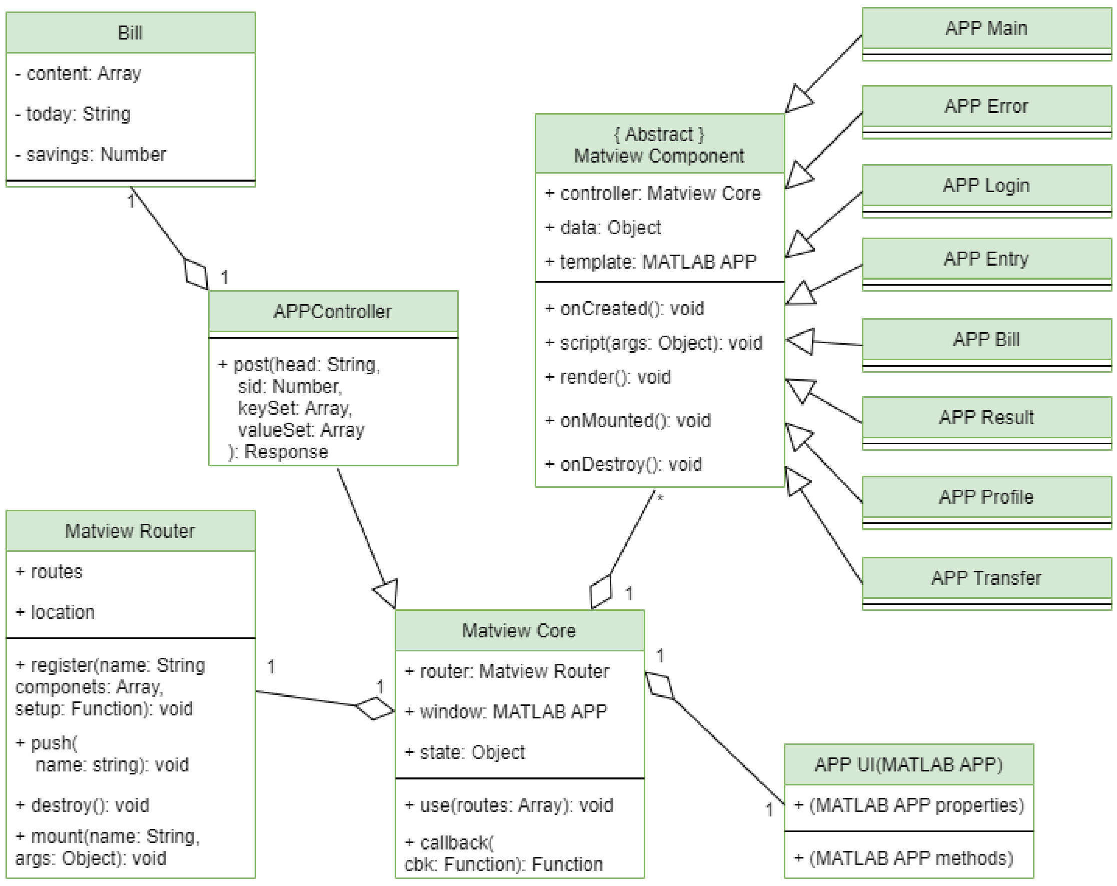


Fig. 1.3. APP Class Diagram

Modules

1. ATM UI (MATLAB APP)

   UI components based on MATLAB APP.

2. Router

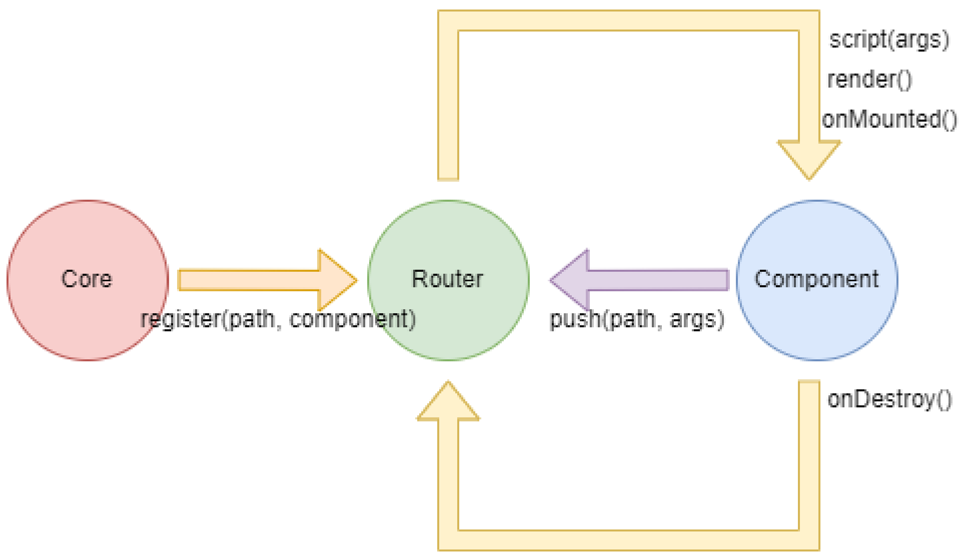
Router class built to change the visibility of the UI components, enable us to build a multi-page application in one page. Properties include routes and path. Methods include register and push.

3. APPController
   Main control of the ATM. Properties include sessionId and status. Methods include post, doLogin, doLogout, doTransfer and doSaveBill.

1.3 Matview Framework

Matview is a vue-inspired framework created by us using MATLAB. It uses uniformed interfaces for developers to write components, connect pages and reuse their codes, providing an easier way in developing a multi-page application in MATLAB.

- Matview Component
  Template of matview components. To add your own page to Matview, you need to bind MATLAB APP instance to the component's template property, and link callbacks if needed in the hook onCreated, which be invoked in the register process. Hooks include script and render hooks are used to manually initialize and bind data to MATLAB APP, and the component's data property will help you to store your data. Hooks like onMounted and onDestroy are two life-cycle hooks invoked after the page is mounted and after the page is destroyed. Developers should not change the page until a page is mounted, in other words, developers must not change the page in script and render hooks.
- Matview Router
  Scheduler of each page's mount and destroy, invoke hooks automatically in the process.
- Matview Core
  Main controller of the framework. It provides state property for components to store status and make communications. Developers can add your components to the system by invoke method use with a cell-like router table. And it is easy for them to extend it with virtual hardware or other customized widgets.



1.4 Virtual Hardware

Virtual Hardware are MATLAB APP simulated hardware created by us to simulate real-life ATMs and other devices. In our design, every hardware instance has three modes: in, out and pending

- In: virtual hardware instance will receive input from users and its indicator will turn green in this mode. Users can click corresponding buttons and give input to the system.
- Out: virtual hardware instance will give output to users and its indicator will turn red in this mode. Users will have to click corresponding buttons to receive the output, otherwise they might face errors in their further interactions.
- Pending: virtual hardware instance will remain silence in this mode, its indicator will stay grey, and no input or output is allowed. It will make no effect for users to click corresponding buttons.

Two life-cycle hooks are provided for asynchronous controls. The first one, onIn (onBlur), will be invoked when an input is finished. And the other one, onOut, will be invoked after an output is finished. Both hooks can be rebind publicly, and developers should make sure no cyclic invoking is involved in the bound callbacks.

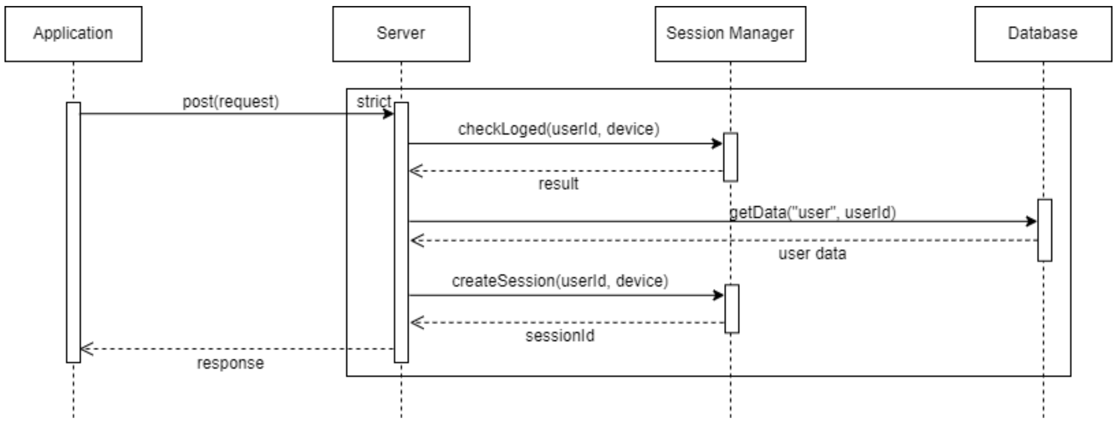
2. System Specifications

2.1 Backend Implementation

The backend serves independently with and without the frontend working. It provides interfaces for users to login, logout, get data and update data, and for admins to login, logout, check data, edit data, run health check, create account, activate account and deactivate account.

2.1.1 User Interaction

2.1.1.1 Login



Request Example

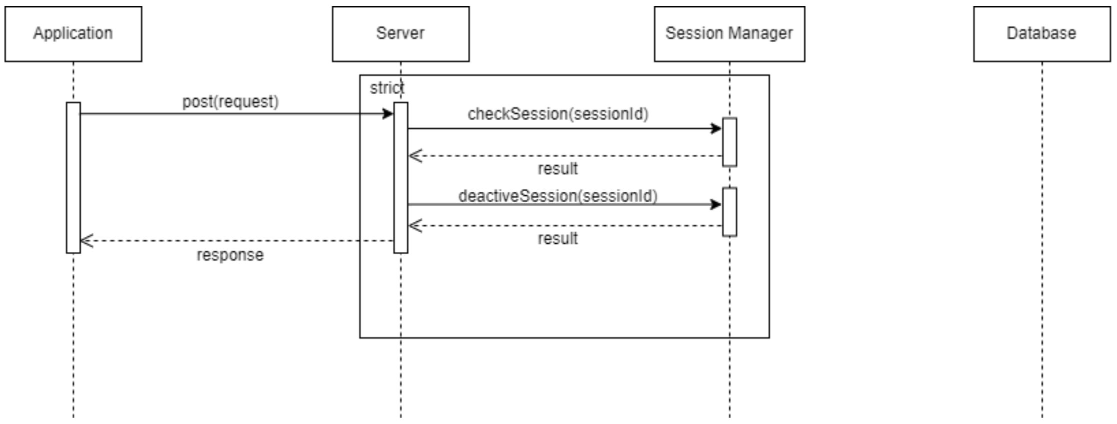
```
{
    "head": "/login",
    "body": {
        "adminId": "7894556355661236",
        "password": "123456",
        "clientKey": "123456"(if not allowApp)
    },
    "device": "atm"(from atm) | "app"(from app),
    "sessionId": 0
}
```

Response Example

```
{
"head": "/",
    "body": {
        "msg": "success"(success) | error message(failure),
    },
    "device": "server",
    "sessionId": 1,
    "code": 200(success) | 400(failure)
}
```

1. Receive a request from the frontend applications.
2. Verify the request.
    a) If it is invalid, then return a response with error messages.
    b) Otherwise, continue the procedure.
3. Check if user has logged in in the user session manager
    a) If an error occurs in the user session manager, then return a response with error messages.
    b) If user has logged in, then return a response with error messages.
    c) Otherwise, continue the procedure.
4. Get data from the database, then verify the parameters from the request.
    a) If an error occurs in the database, then return a response with error messages.
    b) If request's device is app, while user's app login is not activated and the request does not contain or contains wrong client key, then return then return a response with error messages.
    c) Otherwise, continue the procedure.
5. Verify the password.
    a) If it is wrong, then return then return a response with error messages.
    b) Otherwise, continue the procedure.
6. Create a new session in the user session manager.
    a) If an error occurs in the user session manager, then return then return a response with error messages.
    b) Otherwise, continue the procedure.
7. Send a success response with id of the newly created session.

2.1.1.2 Logout



Request Example

```
{
    "head": "/logout",
    "body": {

    },
    "device": "atm"(from atm) | "app"(from app),
    "sessionId": 1
}
```

Response Example

```
{
"head": "/",
    "body": {
        "msg": "success"(success) | error-message(failure),
    },
    "device": "server",
    "sessionId": 0,
    "code": 200(success) | 400(failure)
}
```

1. Receive a request from the frontend applications.
2. Verify the request.
    a) If it is invalid, then return then return a response with error messages.
    b) Otherwise continue the procedure.
3. Verify session id in the user session manager.
    a) If an error occurs in the user session manager, then return a response with

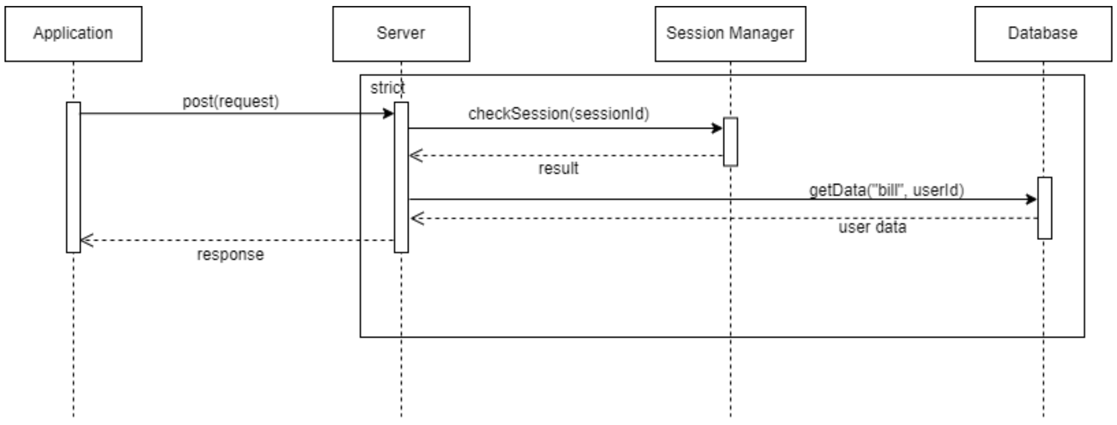
   error messages.

   b) If it is invalid, then return a response with error messages.

   c) Otherwise, continue the procedure.

4. Deactivate the session in the user session manager.
   a) If an error occurs in the user session manager, then return a response with error messages.
   b) Otherwise, continue the procedure.
5. Send a success response.

2.1.1.3 Get Data



Request Example

```
{
    "head": "/getData",
    "body": {

    },
    "device": "atm"(from atm) | "app"(from app),
    "sessionId": 1
}
```

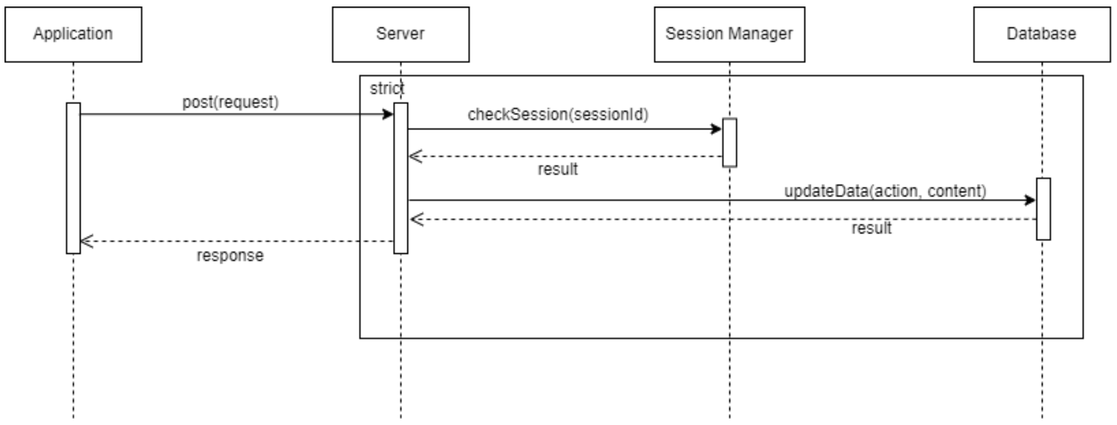
Response Example

```
{
"head": "/",
    "body": {
        "msg": "success"(success) | error-message(failure),
        "bill": `1236171647798361|deposit|12350|12350|2022-04-19 00:36:47
                 1236171647798361|deposit|12350|12700|2022-04-19 00:38:27
```

```
            1236171647798361|deposit|12350|24850|2022-04-19 14:30:54
            1236171647798361|deposit|12350|37200|2022-04-21 15:05:18`
    },
    "device": "server",
    "sessionId": 0,
    "code": 200(success) | 400(failure)
}
```

1. Receive a request from the frontend applications.
2. Verify the request.
   a) If it is invalid, then return then return a response with error messages.
   b) Otherwise continue the procedure.
3. Verify the session id in the user session manager.
   a) If an error occurs in the user session manager, then return a response with error messages.
   b) If it is invalid, then return a response with error messages.
   c) Otherwise, continue the procedure.
4. Verify parameters from the request.
   a) If they are invalid, then return a response with error messages.
   b) Otherwise, continue the procedure.
5. Get data from the database.
   a) If an error occurs in the database, then return a response with error messages.
   b) Otherwise, continue the procedure.
6. Send a success response with user data.

2.1.1.4 Update Data



Request Example

```
{
    "head": "/updateData",
    "body": {
    "action": "deposit" | "transfer" | "withdraw",
        "content": {
            "value": 123,
            "target": "1234567887456123"(if transfer)
        }
    },
    "device": "atm"(from atm) | "app"(from app),
    "sessionId": 1
}
```

Response Example

```
{
"head": "/",
    "body": {
        "msg": "success"(success) | error-message(failure),
    },
    "device": "server",
    "sessionId": 0,
    "code": 200(success) | 400(failure)
}
```

1. Receive a request from the frontend applications.
2. Verify the request.
    a) If it is invalid, then return then return a response with error messages.
    b) Otherwise continue the procedure.
3. Verify the session id in the user session manager.
    a) If an error occurs in the user session manager, then return a response with error messages.
    b) If it is invalid, then return a response with error messages.
    c) Otherwise, continue the procedure.
4. Verify the parameters from the request.
    a) Make validations according to the action.
        i. If action is withdrawal, check if the target value is less than user's savings.
        ii. If action is deposition, nothing.
        iii. If action is transference, check if the target account exists and the target value is less than user's savings.
    b) If the parameters are invalid, then return a response with error messages.
    c) Otherwise, continue the procedure.
5. Update the database.
    a) If an error occurs in the database, then return a response with error messages.

b) Otherwise, continue the procedure.

6. Send a success response.

2.1.2 Admin Interaction

2.1,2,1 Login

Request Example

```
{
    "head": "/slogin",
    "body": {
        "adminId": "7894556355661236",
        "password": "123456"
    },
    "device": "admin",
    "sessionId": 0
}
```

Response Example

```
{
"head": "/",
    "body": {
        "msg": "success"(success) | error-message(failure),
    },
    "device": "server",
    "sessionId": 1,
    "code": 200(success) | 400(failure)
}
```

The same as user's login, but the request comes from console, and it is the admin session manager instead of the user session manager to manage sessions.

2.1.2.2 Logout

Request Example

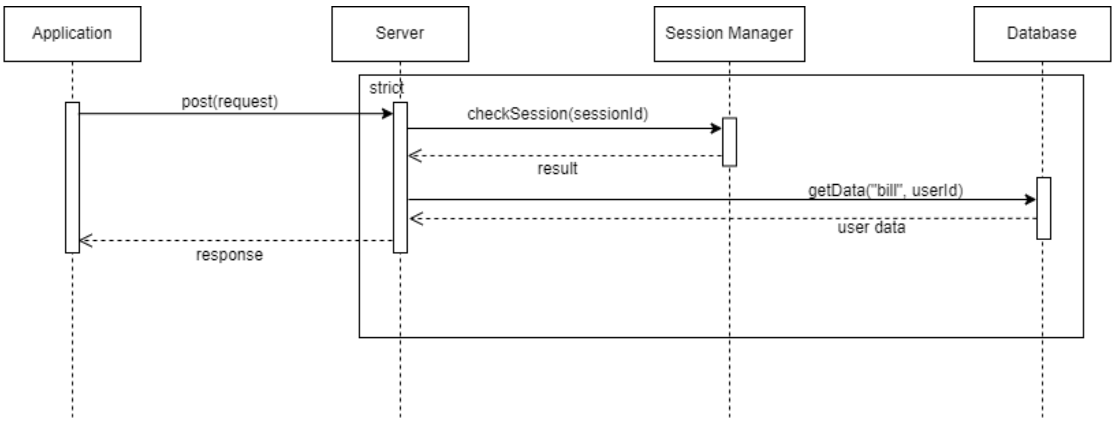
```
{
    "head": "/slogout",
    "body": {

    },
    "device": "admin",
    "sessionId": 1
}
```

Response Example

```
{
"head": "/",
    "body": {
        "msg": "success"(success) | error-message(failure),
    },
    "device": "server",
    "sessionId": 0,
    "code": 200(success) | 400(failure)
}
```

The same as user's logout, but the request comes from console, and it is the admin session manager instead of the user session manager to manage sessions.

2.1.2.3 Check Data



Request Example

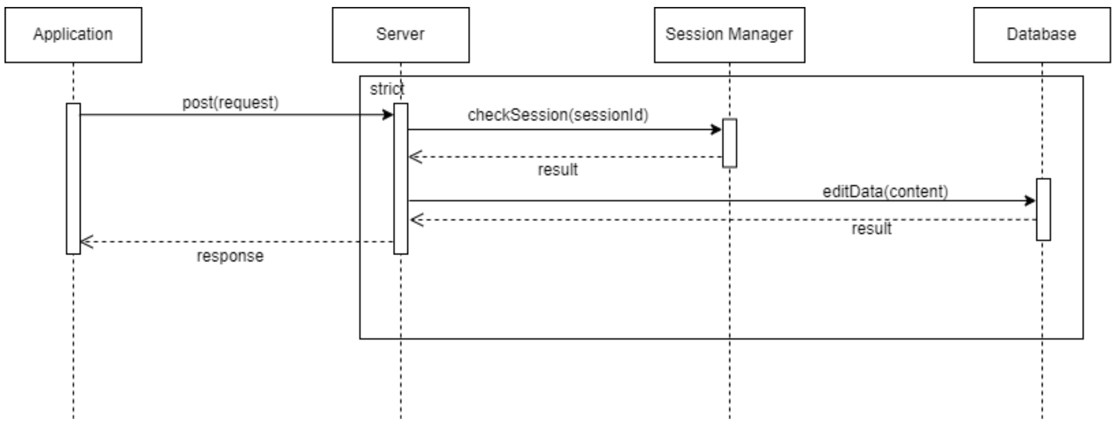
```
{
    "head": "/checkData",
    "body": {
        "userId": "1145141919810123",
    },
    "device": "admin",
    "sessionId": 1
}
```

Response Example

```
{
"head": "/",
    "body": {
        "msg": "success"(success) | error-message(failure),
        "bill": `1236171647798361|deposit|12350|12350|2022-04-19 00:36:47
                1236171647798361|deposit|12350|12700|2022-04-19 00:38:27
                1236171647798361|deposit|12350|24850|2022-04-19 14:30:54
                1236171647798361|deposit|12350|37200|2022-04-21 15:05:18`

    },
    "device": "server",
    "sessionId": 0,
    "code": 200(success) | 400(failure)
}
```

1. Receive a request from admins.
2. Verify the request.
   a) If it is invalid, then return then return a response with error messages.
   b) Otherwise continue the procedure.
3. Verify the session id in the user session manager.
   a) If an error occurs in the user session manager, then return a response with error messages.
   b) If it is invalid, then return a response with error messages.
   c) Otherwise, continue the procedure.
4. Verify parameters from the request.
   a) If they are invalid, then return a response with error messages.
   b) Otherwise, continue the procedure.
5. Get data from the database.
   a) If an error occurs in the database, then return a response with error messages.
   b) Otherwise, continue the procedure.
6. Send a success response with user data.

2.1.2.4 Edit Data



Request Example

```
{
    "head": "/editData",
    "body": {
        "userId": "1145141919810123",
        "content": {
            "condition": "action=='transfer'"(STL like),
             "result": "action='receive'"(STL like)
        }
    },
    "device": "admin",
    "sessionId": 1
}
```

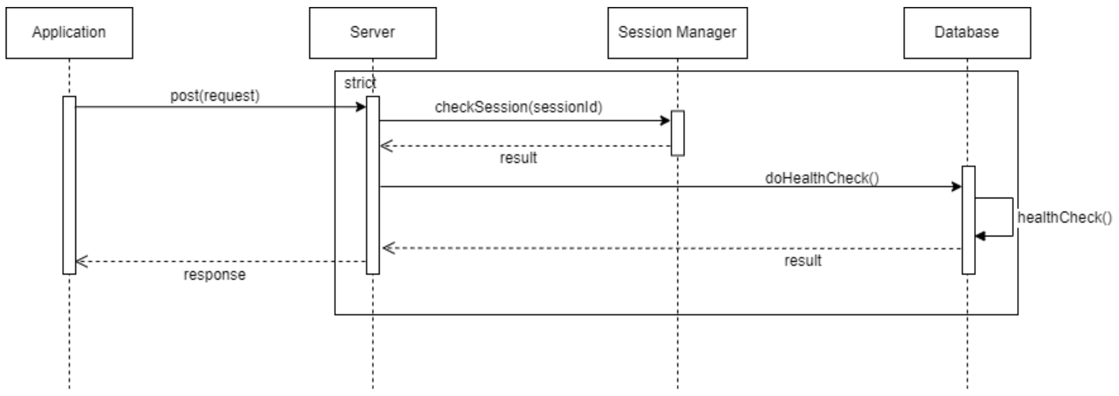
Response Example

```
{
    "head": "/",
    "body": {
        "msg": "success"(success) | error-message(failure),
    },
    "device": "server",
    "sessionId": 0,
    "code": 200(success) | 400(failure)
}
```

1. Receive a request from admins.
2. Verify the request.

   a) If it is invalid, then return then return a response with error messages.
   b) Otherwise continue the procedure.
3. Verify the session id in the user session manager.
   a) If an error occurs in the user session manager, then return a response with error messages.
   b) If it is invalid, then return a response with error messages.
   c) Otherwise, continue the procedure.
4. Verify parameters from the request.
   a) If they are invalid, then return a response with error messages.
   b) Otherwise, continue the procedure.
5. Update the database.
   a) If an error occurs in the database, then return a response with error messages.
   b) Otherwise, continue the procedure.
6. Send a success response.

2.1.2.5 Health Check



Request Example

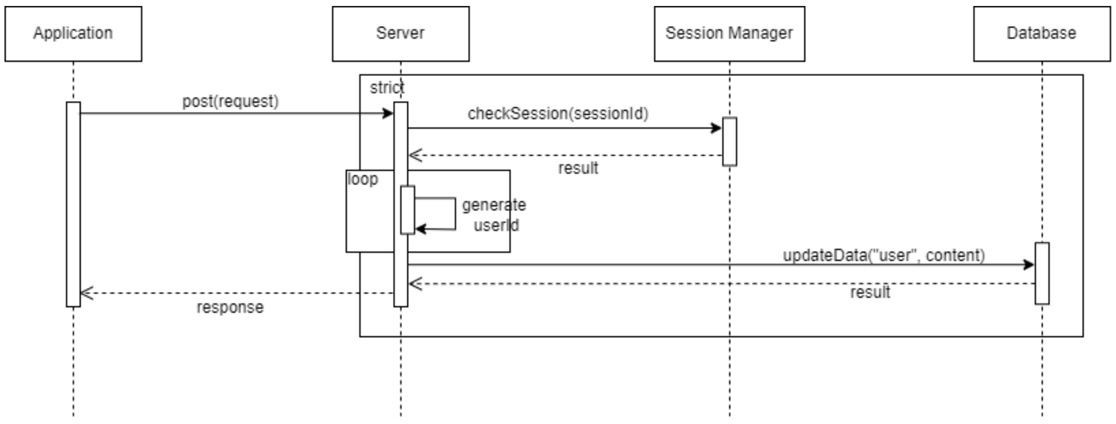
```
{
    "head": "/healthCheck",
    "body": {

    },
    "device": "admin",
    "sessionId": 1
}
```

Response Example

```
{
"head": "/",
    "body": {
        "msg": "success"(success) | error-message(failure),
    },
    "device": "server",
    "sessionId": 0,
    "code": 200(success) | 400(failure)
}
```

1. Receive a request from admins.
2. Verify the request.
   a) If it is invalid, then return then return a response with error messages.
   b) Otherwise continue the procedure.
3. Verify the session id in the user session manager.
   a) If an error occurs in the user session manager, then return a response with error messages.
   b) If it is invalid, then return a response with error messages.
   c) Otherwise, continue the procedure.
4. Database run health check sequences.
   a) If an error occurs in the database, then return a response with error messages.
   b) Otherwise, continue the procedure.
5. Send a success response.

2.1.2.5 Create Account



Request Example

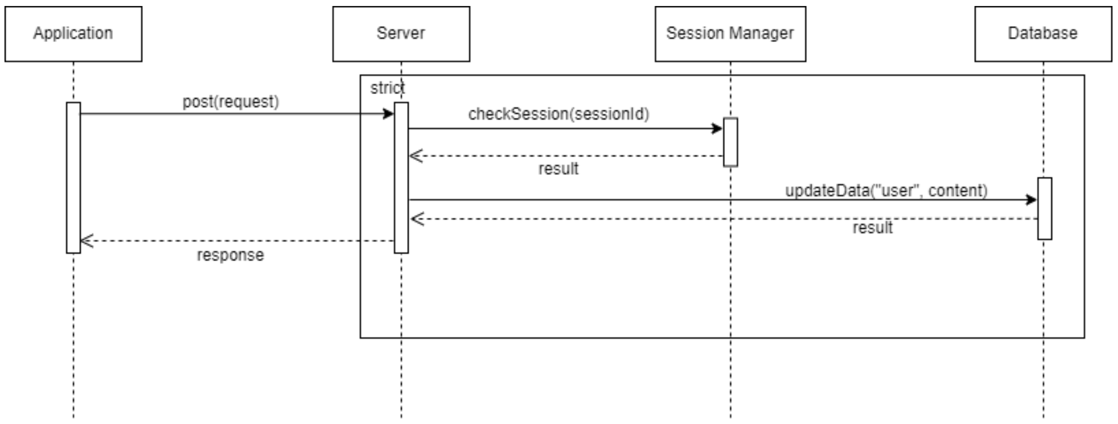
```
{
    "head": "/createAccount",
    "body": {
        "password": "123456"
    },
    "device": "admin",
    "sessionId": 1
}
```

Response Example

```
{
"head": "/",
    "body": {
        "msg": "success"(success) | error-message(failure),
        "userId": "1234567891234567"
    },
    "device": "server",
    "sessionId": 0,
    "code": 200(success) | 400(failure)
}
```

1. Receive a request from admins.
2. Verify the request.
    a) If it is invalid, then return then return a response with error messages.
    b) Otherwise continue the procedure.
3. Verify the session id in the user session manager.
    a) If an error occurs in the user session manager, then return a response with error messages.
    b) If it is invalid, then return a response with error messages.
    c) Otherwise, continue the procedure.
4. Verify parameters from the request.
    a) If they are invalid, then return a response with error messages.
    b) Otherwise, continue the procedure.
5. Generate user id by random.
6. Update the database.
    a) If an error occurs in the database, then return a response with error messages.
    b) Otherwise, continue the procedure.
7. Send a success response with the newly generated user id.

2.1.2.6 Activate Account



Request Example

```
{
    "head": "/activeAccount",
    "body": {
        "userId": "1234567891234567"
    },
    "device": "admin",
    "sessionId": 1
}
```

Response Example

```
{
"head": "/",
    "body": {
        "msg": "success"(success) | error-message(failure),
    },
    "device": "server",
    "sessionId": 0,
    "code": 200(success) | 400(failure)
}
```

1. Receive a request from admins.
2. Verify the request.
    a) If it is invalid, then return then return a response with error messages.
    b) Otherwise continue the procedure.
3. Verify the session id in the user session manager.
    a) If an error occurs in the user session manager, then return a response with

error messages.

   b) If it is invalid, then return a response with error messages.
   c) Otherwise, continue the procedure.

4. Verify parameters from the request.
   a) If they are invalid, then return a response with error messages.
   b) Otherwise, continue the procedure.
5. Update the database.
   a) If an error occurs in the database, then return a response with error messages.
   b) Otherwise, continue the procedure.
6. Send a success response.

2.1.2.7 Deactivate Account

Same as activate account, by the content to be updated in the database is the opposite.

Request Example

```
{
    "head": "/deactiveAccount",
    "body": {
        "userId": "1234567891234567"
    },
    "device": "admin",
    "sessionId": 1
}
```

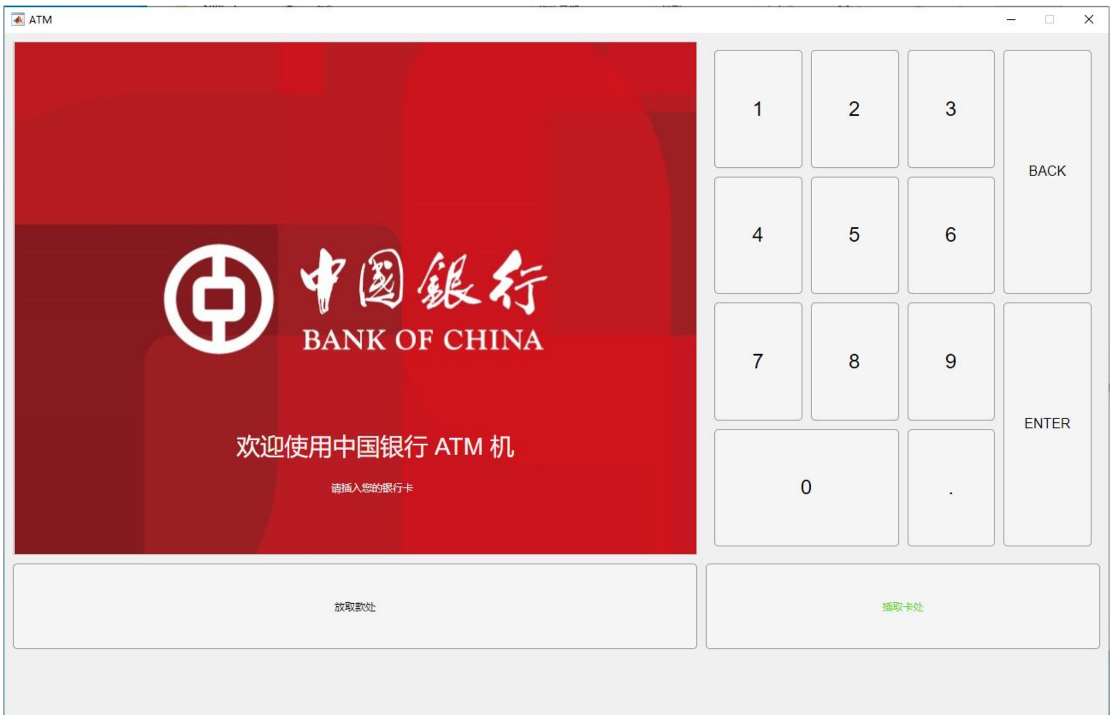
Response Example

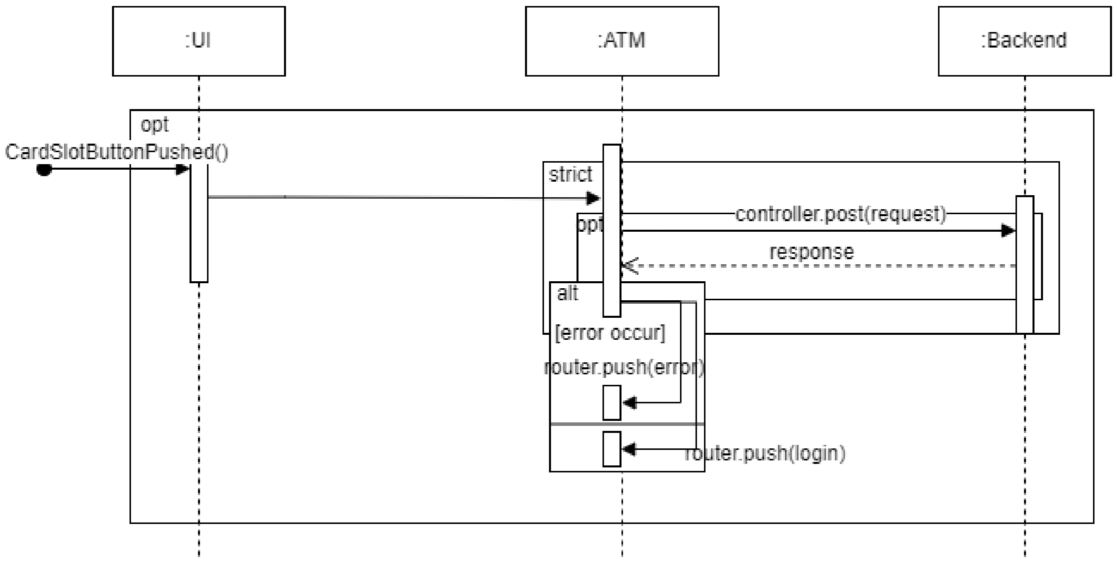
```
{
    "head": "/",
    "body": {
        "msg": "success"(success) | error-message(failure),
    },
    "device": "server",
    "sessionId": 0,
    "code": 200(success) | 400(failure)
}
```

2.2 Frontend Implement

2.2.1 ATM Implement

2.2.1.1 Entry

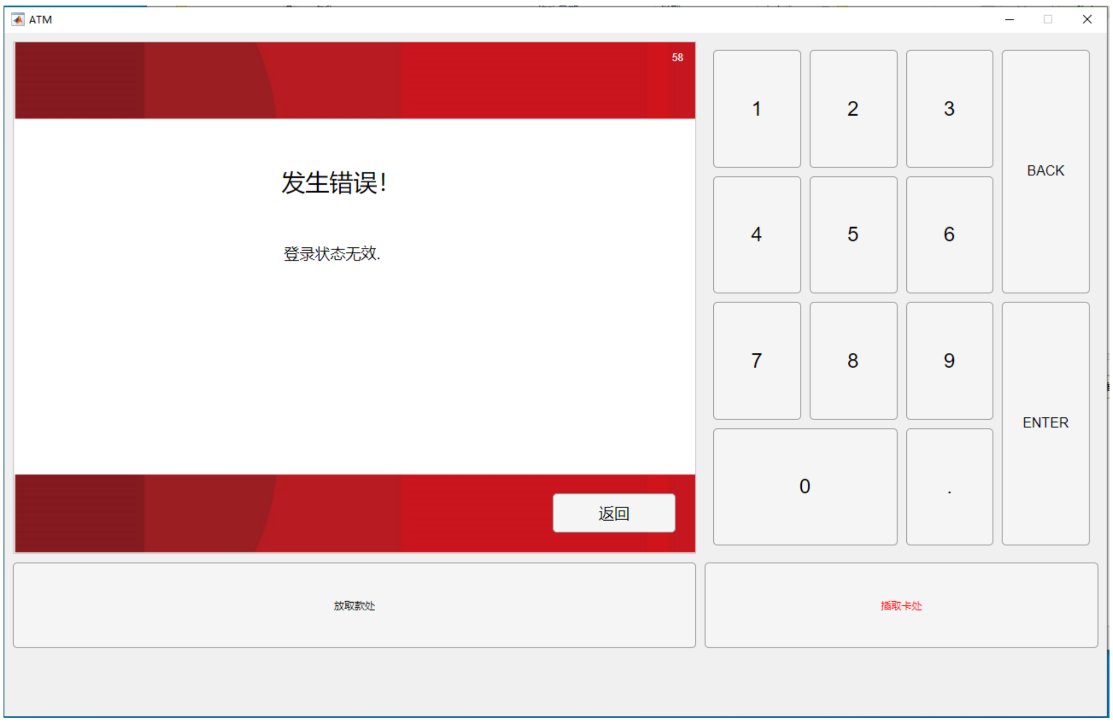




- Press the Card Slot button.
  1. Check health status

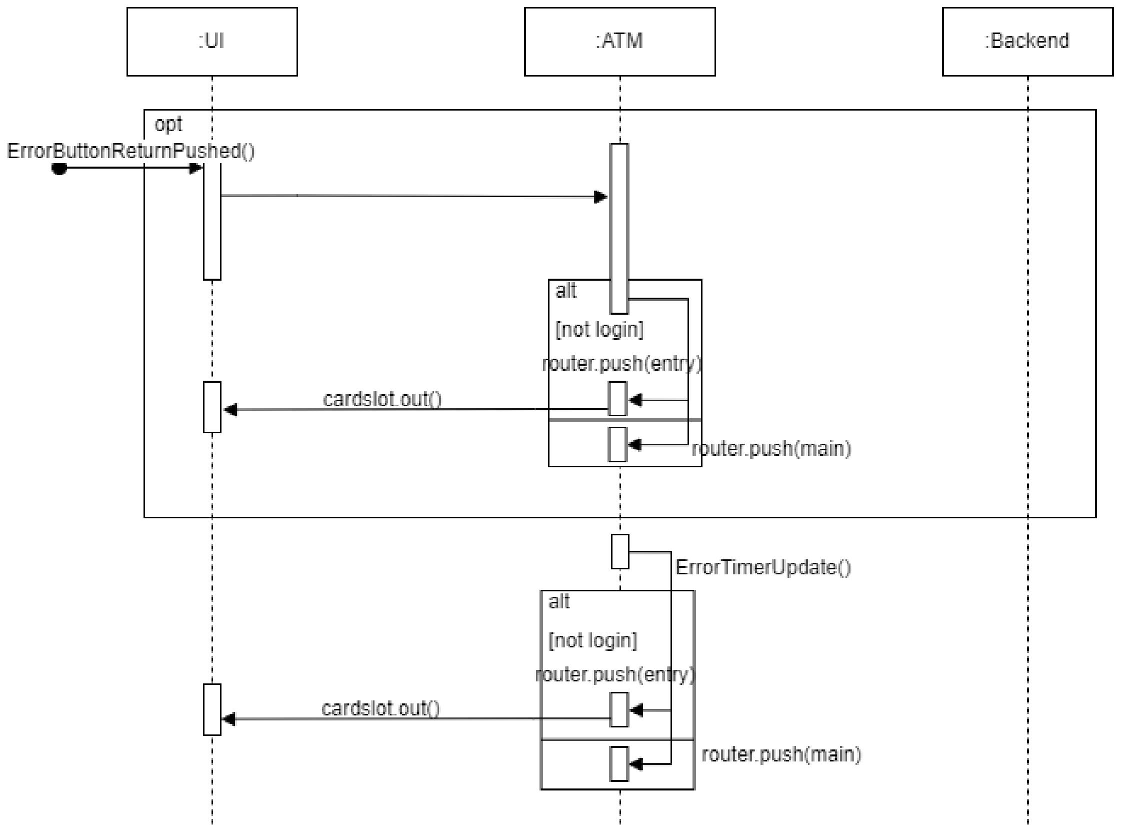
   a) If an error occurs, go to Error page with error messages.
   b) Otherwise, continue the procedure.
2. Verify the card.
   a) If it is invalid, go to Error page with error messages.
   b) Otherwise, continue the procedure.
3. The ATM will send request to the backend to get user data.
   a) If an error occurs in the backend, go to Error page with error messages.
   b) If the user does not exist or is not active, go to Error page with error messages.
   c) Otherwise, continue the procedure.
4. Go to Login page.

2.2.1.2 Error



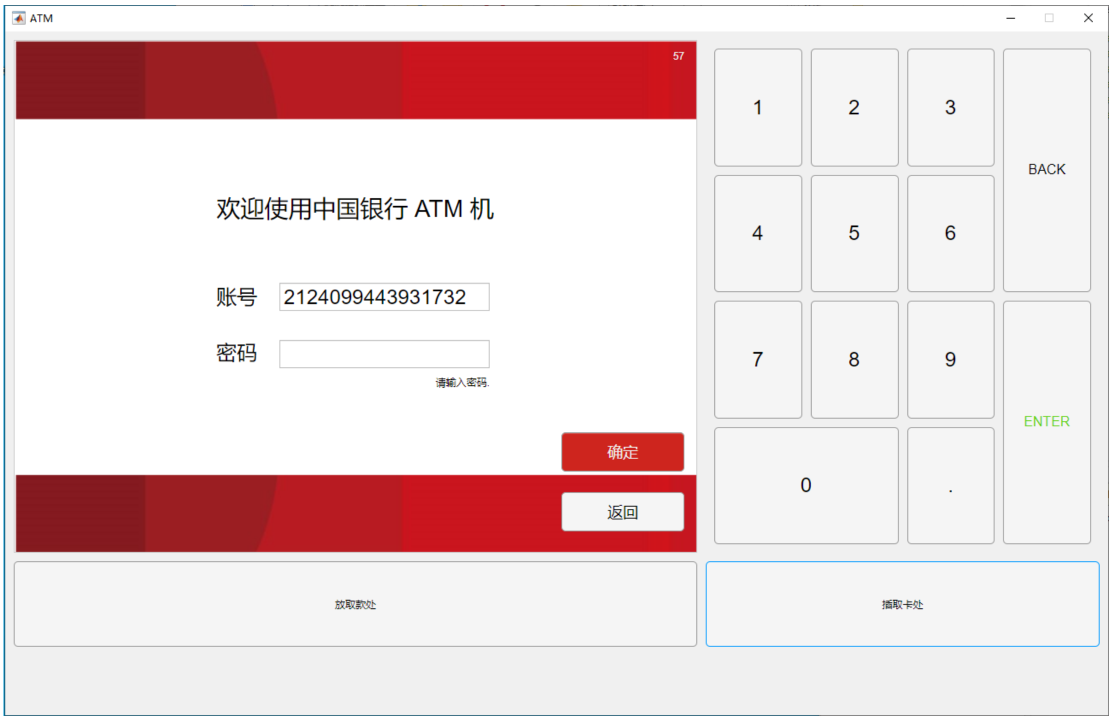
发生错误!

登录状态无效.



- Press the Return button.
  1. Check user’s log in status.
     a) If user has logged in, then go to Main page.
     b) Otherwise, continue the procedure.
  2. Go to Entry page and eject the card.
- Time runs out.
  1. The same as pressing the Return button.

2.2.1.3 Login


ATM

57

欢迎使用中国银行 ATM 机

账号 2124099443931732

密码

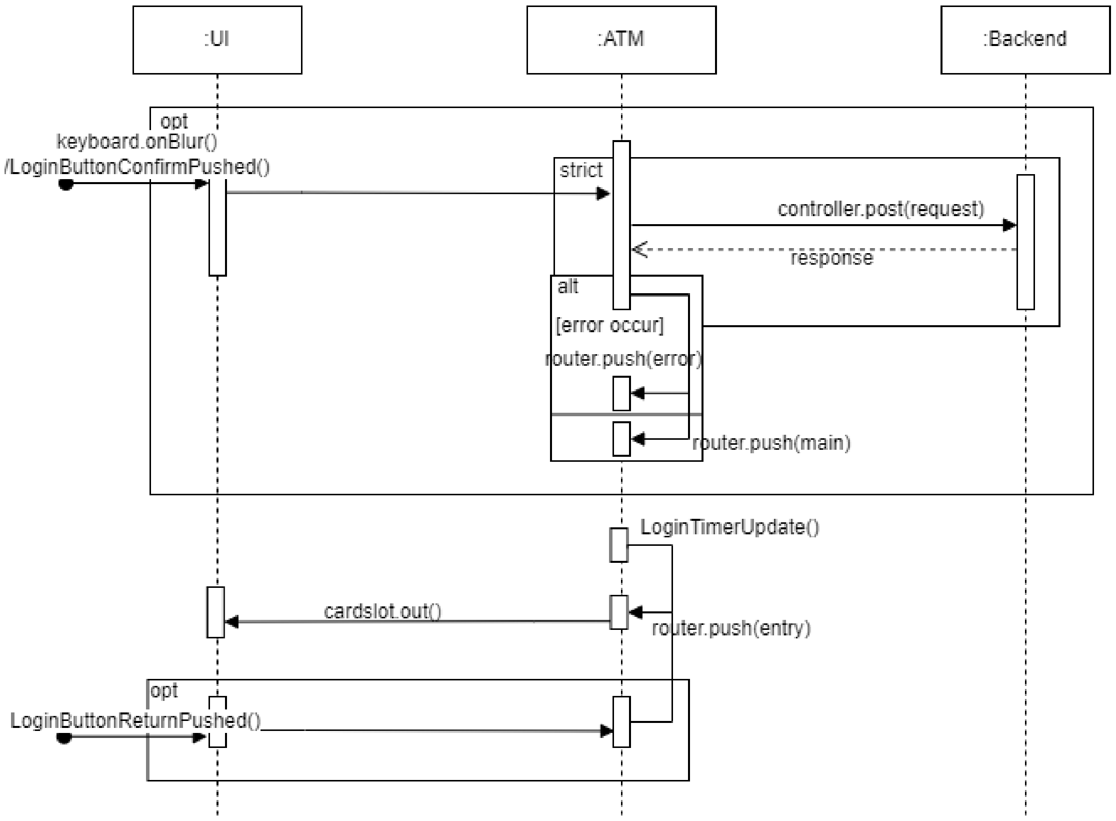
请输入密码.

确定

返回

放取款处

插取卡处

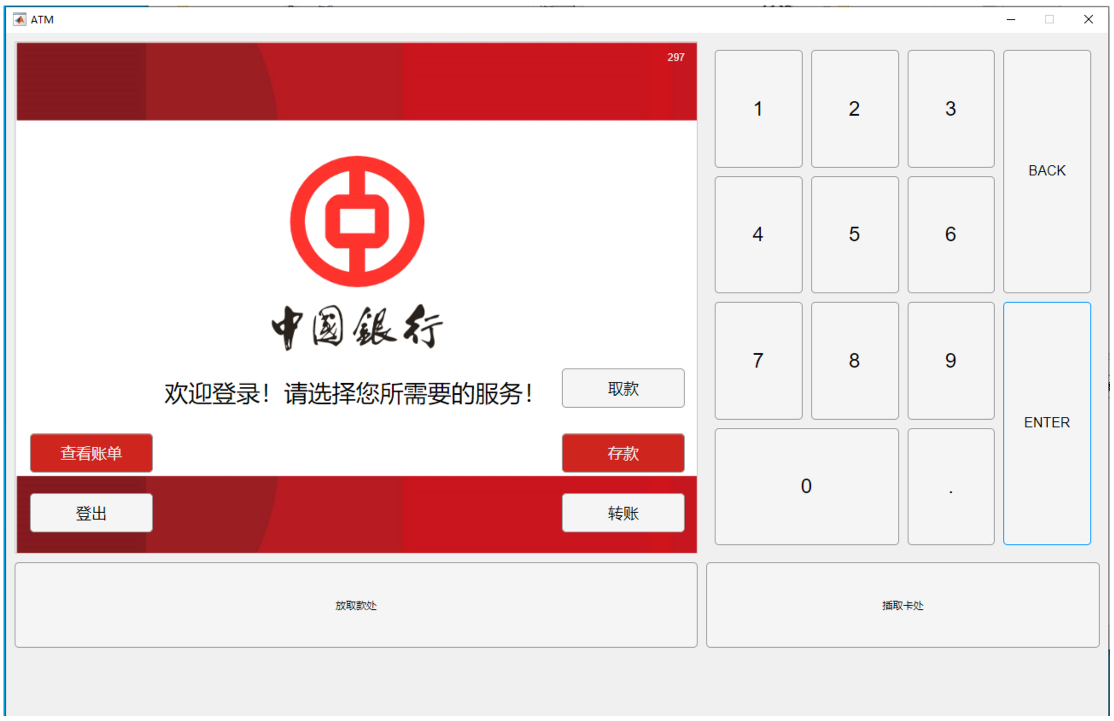


- Press '#' on the virtual keyboard or press the Confirm button.

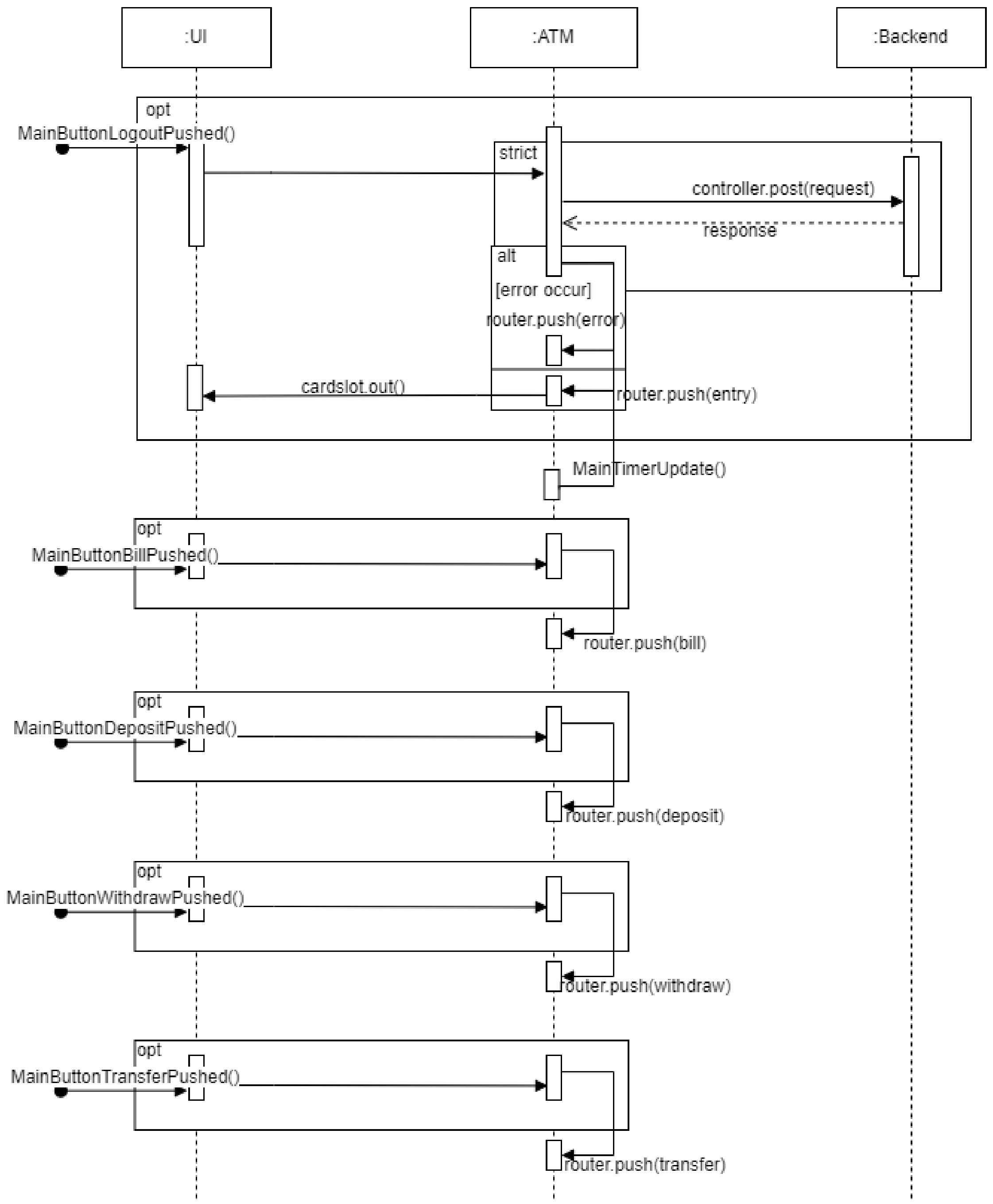
1. Verify the password.
    a) If it is invalid, give a hint message and return.
    b) Otherwise, continue the procedure.
2. Send post to the backend.
    a) If an error occurs in the backend, go to Error page with error messages.
    b) If login failed in the backend, give a hint message and return.
    c) Otherwise, continue the procedure.
3. Store session id from the response and go to Main page.

- Press the Return button.
    1. Go to Entry page and eject the card.
- Time runs out.
    1. The same as pressing the Return button.

2.2.1.4 Main



欢迎登录！请选择您所需要的服务！



- Press the Bill button.
  1. Go to Bill page.
- Press the Deposit button.
  1. Go to Deposit page.
- Press the Withdraw button.
  1. Go to Withdraw page.
- Press the Transfer button.
  1. Go to Transfer page.
- Press the Logout button.
  1. Send post to the backend.
     a) If an error occurs in the backend, go to Error page with error

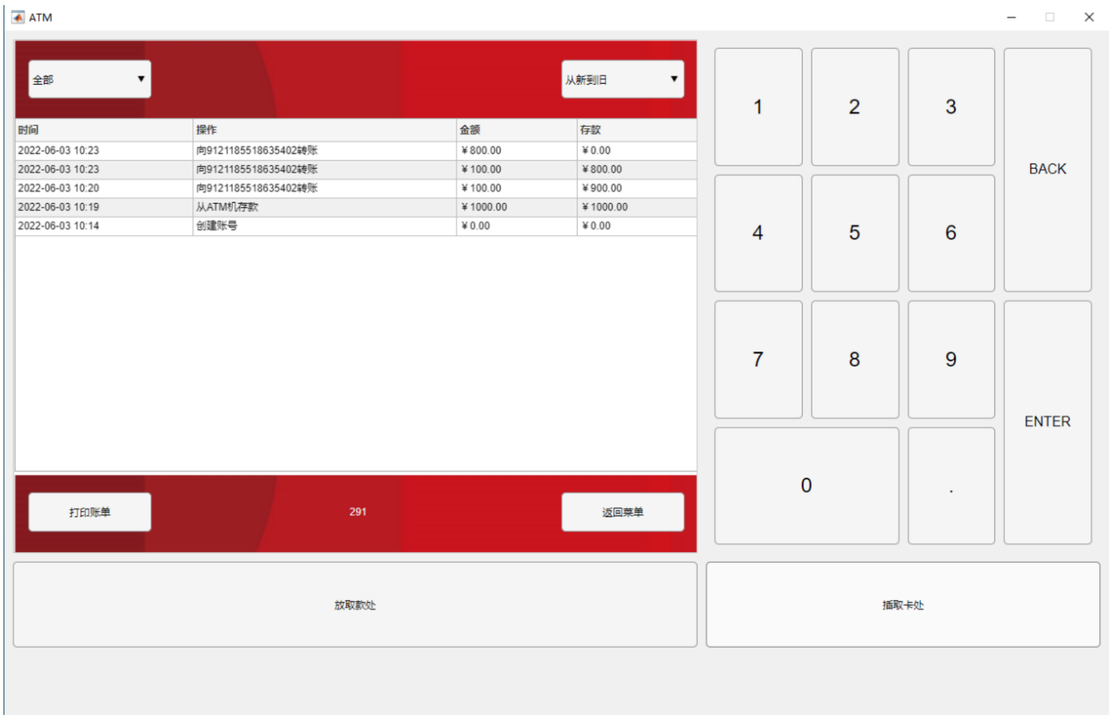
messages.

b) Otherwise, continue the procedure.

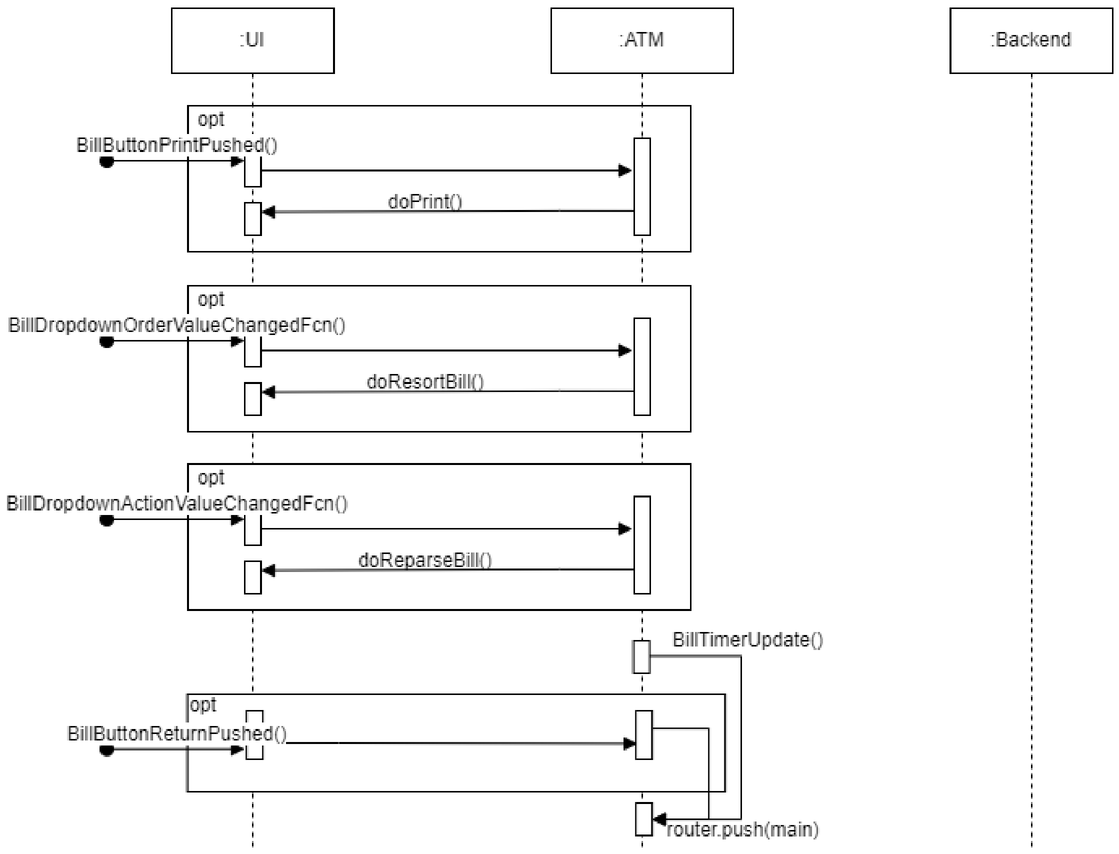
2. Clear session id and continue the procedure.
3. Go to Entry page and eject the card.

- Time runs out.
  1. The same as pressing the Logout button.

2.2.1.5 Bill

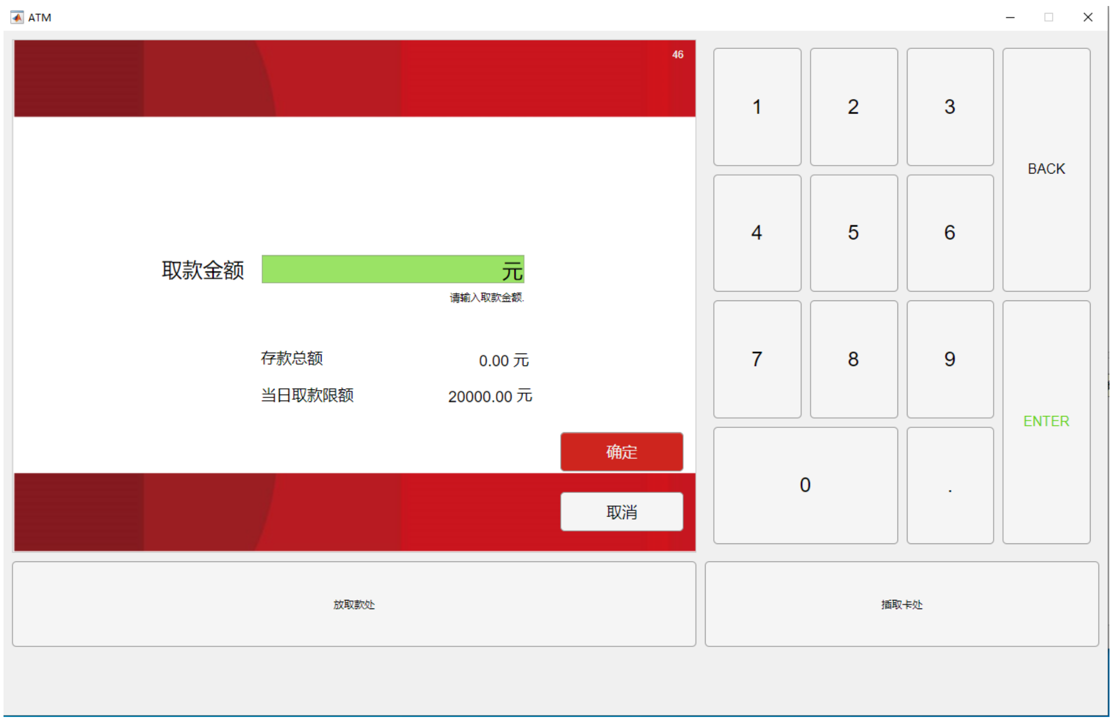


时间	操作	金额	存款
2022-06-03 10:23	向9121185518635402转账	¥800.00	¥0.00
2022-06-03 10:23	向9121185518635402转账	¥100.00	¥800.00
2022-06-03 10:20	向9121185518635402转账	¥100.00	¥900.00
2022-06-03 10:19	从ATM机存款	¥1000.00	¥1000.00
2022-06-03 10:14	创建账号	¥0.00	¥0.00



- Press the Print button.
  1. Pop out a figure with the content of bill, simulating the bill printing on real ATMs.
- Press the Return button.
  1. Go to Main page.
- Select the action.
  1. Reparse and update the bill.
- Select the order.
  1. Resort and update the bill.
- Time runs out.
  1. The same as pressing the Return button.

2.2.1.6 Withdraw

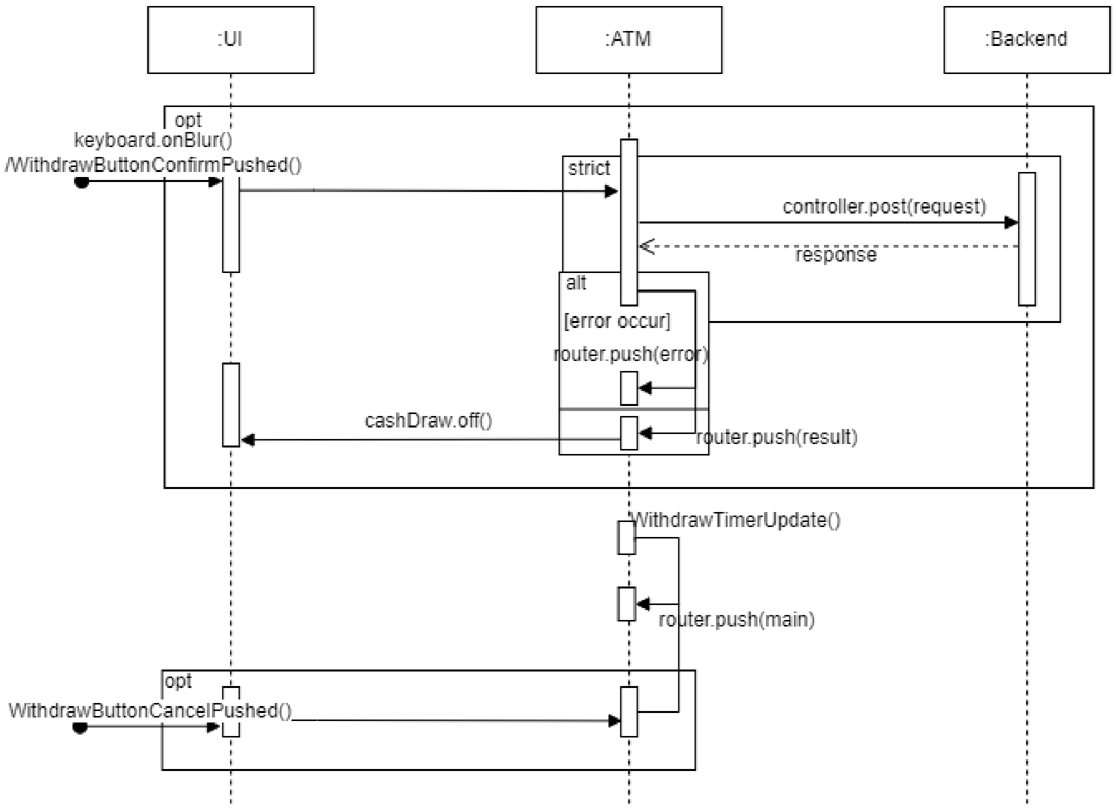


取款金额 元

请输入取款金额.

存款总额 0.00 元

当日取款限额 20000.00 元

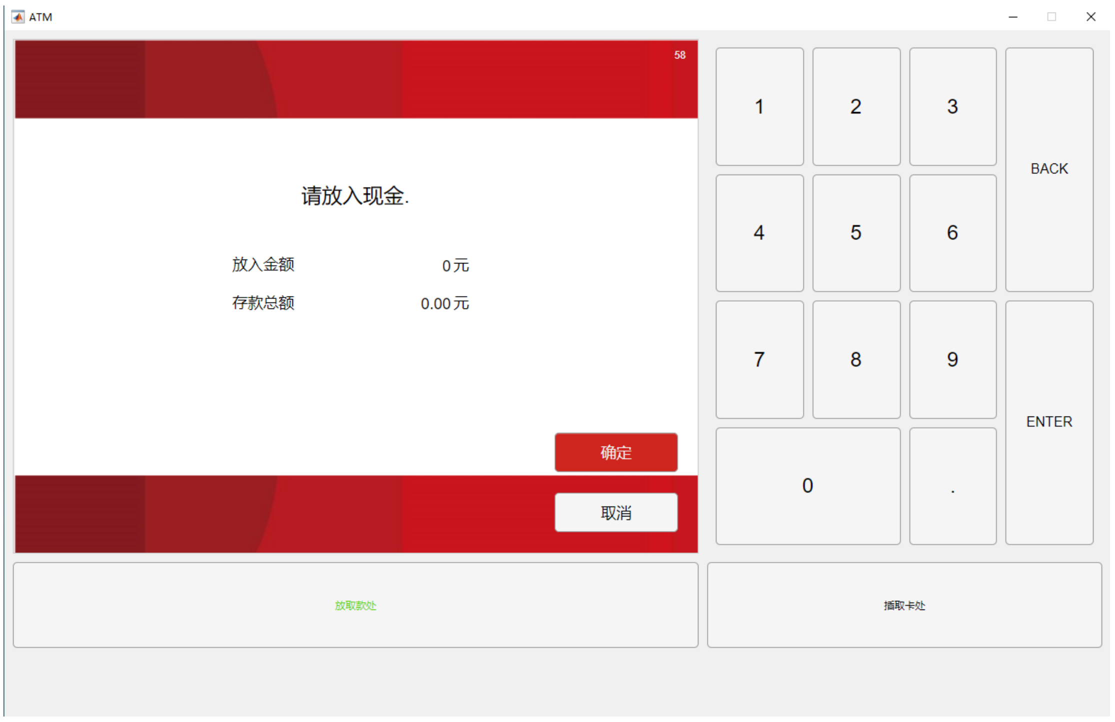


- Press ‘#’ on the virtual keyboard or press the Confirm button.

1. Verify the withdraw value.
   a) If it is invalid, give a hint message and return.
   b) Otherwise, continue the procedure.
2. Send post to the backend.
   a) If an error occurs in the backend, go to Error page with error messages.
   b) If withdraw failed in the backend, go to Error page with error messages.
   c) Otherwise, continue the procedure.
3. Go to Result page with action withdraw and open the cash draw with withdrew cash.

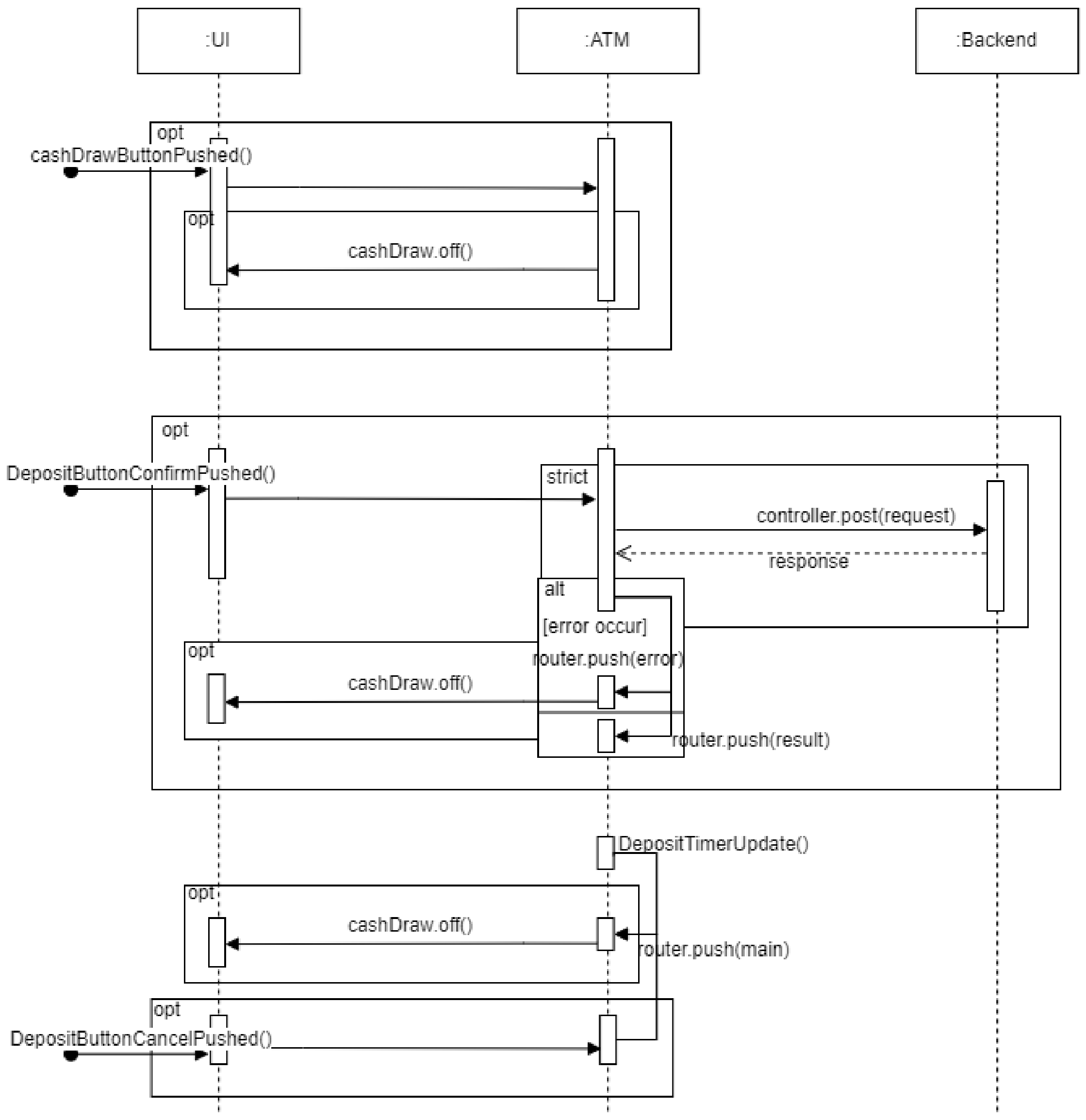
- Press the Cancel button.
  1. Go to Main page.
- Time runs out.
  1. The same as pressing the Cancel button.

2.2.1.7 Deposit



请放入现金.

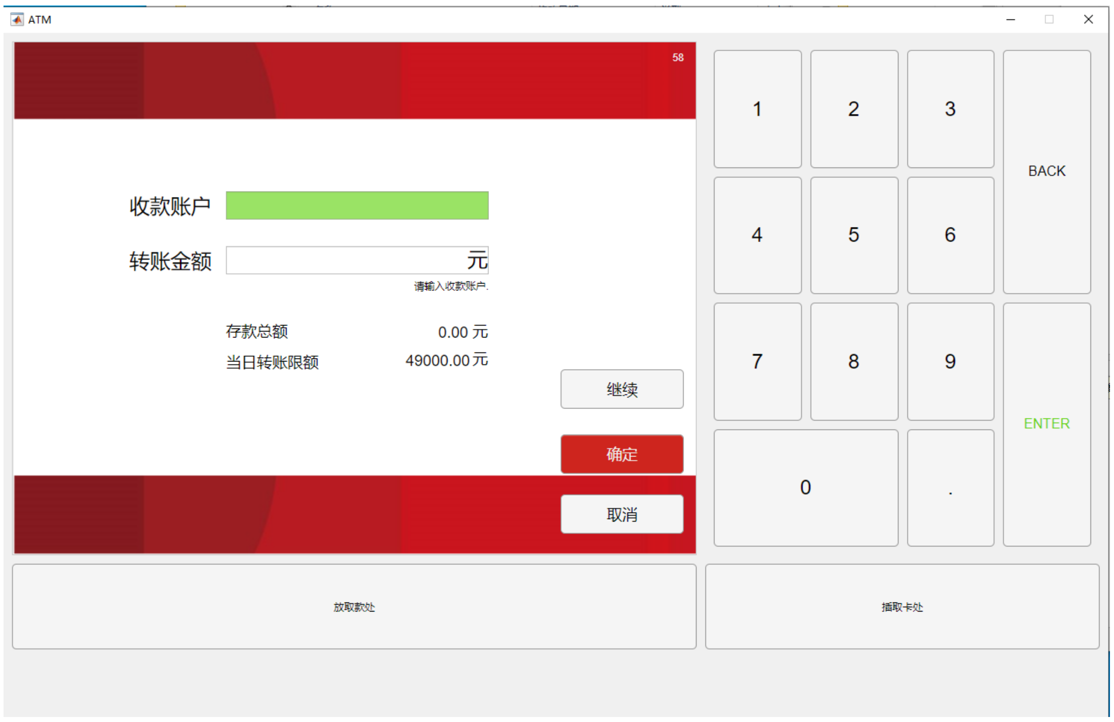
放入金额	0 元
存款总额	0.00 元

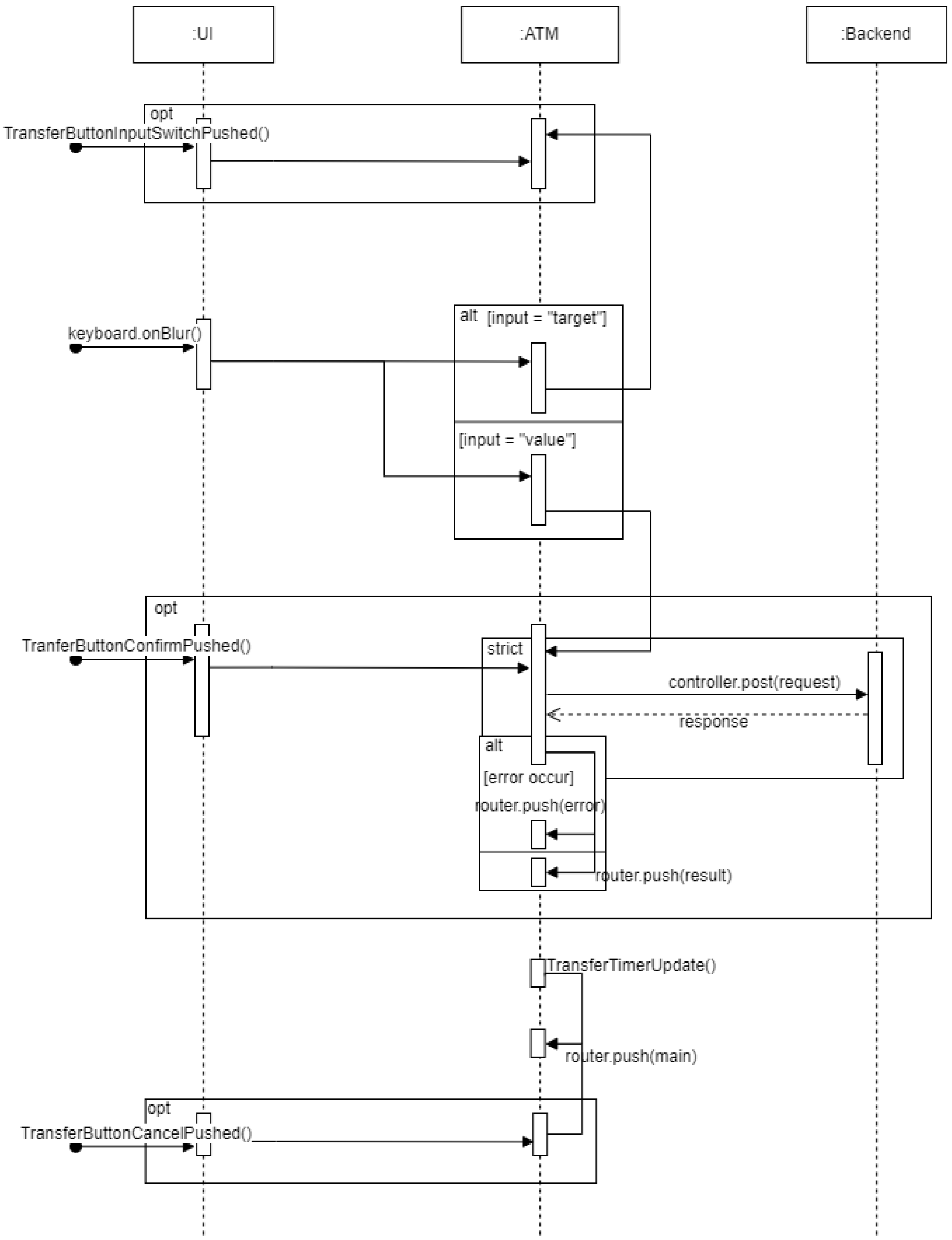


- Press the Cash Draw button.
  1. Verify the cashes.
     a) If they are invalid, open the cash draw with the invalid cash.
     b) Otherwise continue the procedure.
  2. Add the money of the cashes into the total deposited money.
- Press the Confirm button.
  1. Verify the deposit value.
     a) If it is invalid, give a hint message, and if the cash draw is not empty, open the cash draw with the deposited money, then return.
     b) Otherwise, continue the procedure.
  2. Send post to the backend.
     a) If an error occurs in the backend, go to Error page with error messages, and if the cash draw is not empty, open the cash draw with the deposited money.

    b) If deposit failed in the backend, go to Error page with error messages, and if the cash draw is not empty, open the cash draw with the deposited money.
    c) Otherwise, continue the procedure.
  3. Go to Result page with action deposit.
- Press the Cancel button.
  1. Go to Main page, and if the cash draw is not empty, open the cash draw with the deposited money.
- Time runs out.
  1. The same as pressing the Cancel button.

2.2.1.8 Transfer


收款账户
转账金额 元
请输入收款账户.
存款总额 0.00 元
当日转账限额 49000.00 元



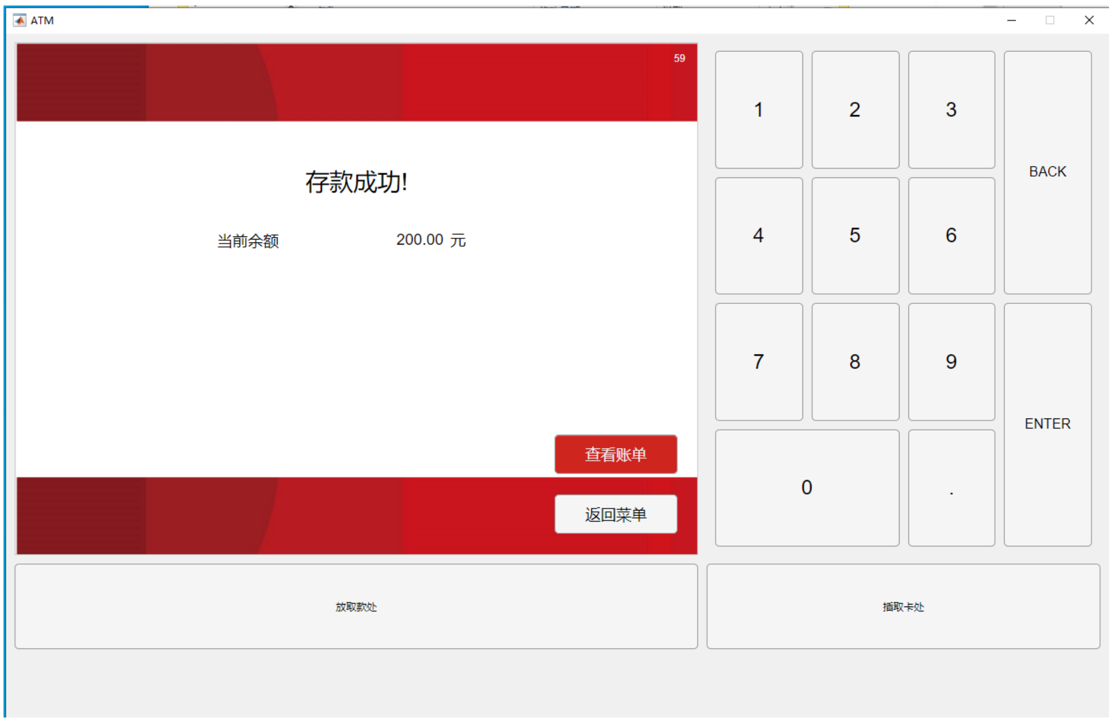
- Press the Switch button.
  1. Switch the input focus.
- Press '#' on the virtual keyboard
  1. Check the input focus.
     a) If the input focus is 'target', then switch the input focus.
     b) If the input focus is 'value', then do the transfer sequences.
- Press the Confirm button.
  1. Verify the transfer target and value.

a) If it is invalid, give a hint message, and return.
b) Otherwise, continue the procedure.

2. Send post to the backend.
   a) If an error occurs in the backend, go to Error page with error messages.
   b) If transfer failed in the backend, go to Error page with error messages.
   c) Otherwise, continue the procedure.
3. Go to Result page with action transfer.

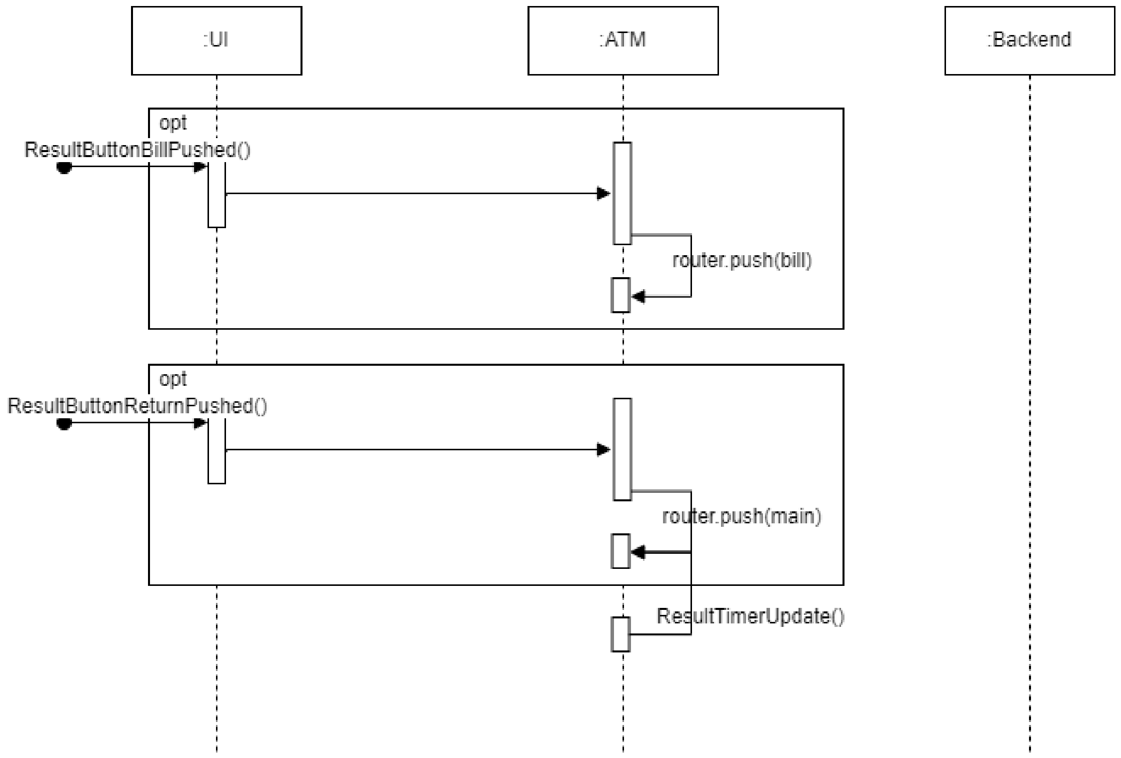
- Press the Cancel button.
  1. Go to Main page.
- Time runs out.
  1. The same as pressing the Cancel button.

2.2.1.9 Result



存款成功!

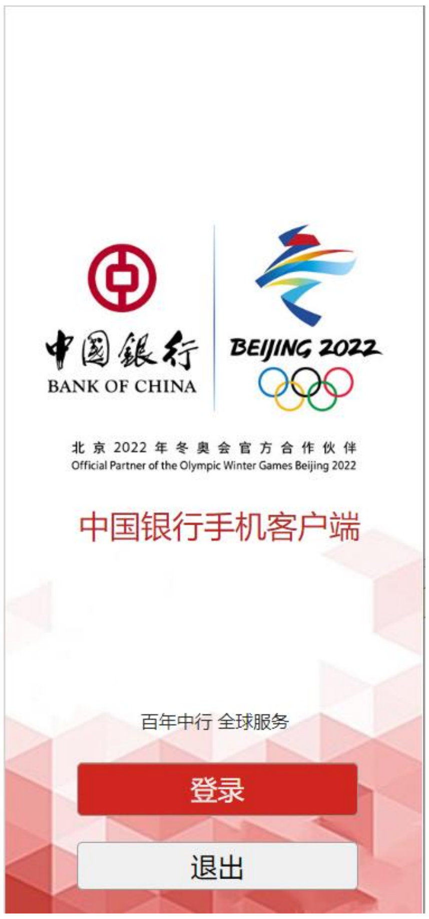
当前余额 200.00 元

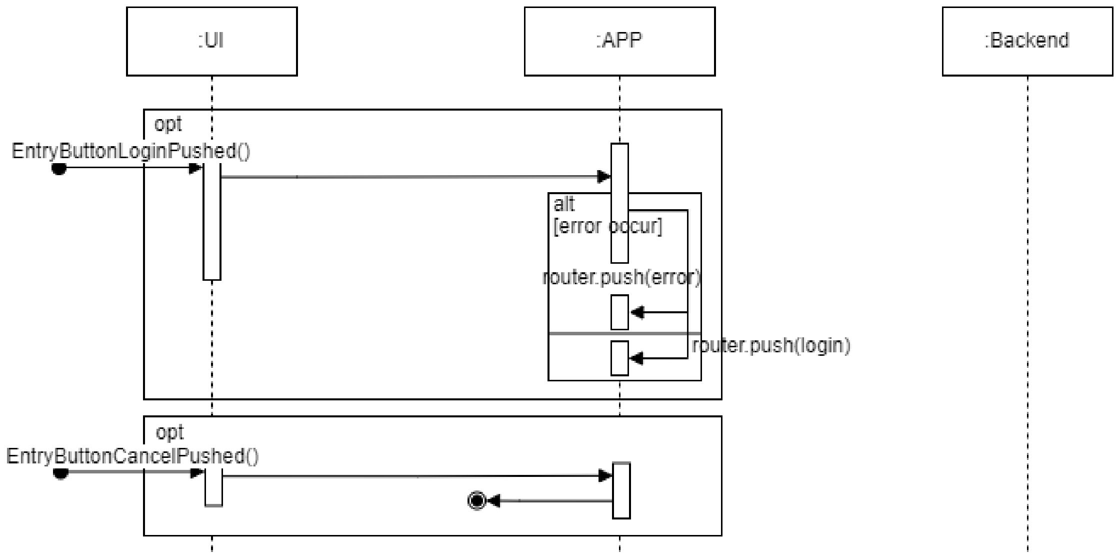


- Press the Bill button.
  1. Go to Bill page.
- Press the Return button.
  1. Go to Main page.
- Time runs out.
  1. The same as pressing the Return button.

2.2.2 APP Implement

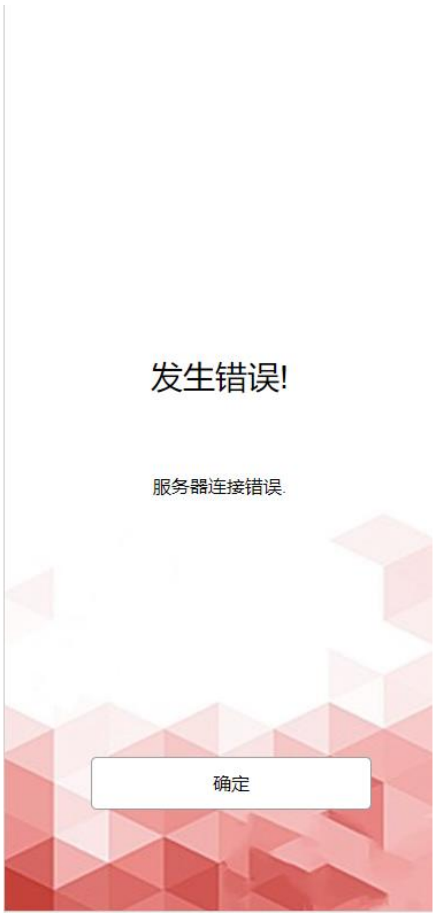
2.2.2.1 Entry

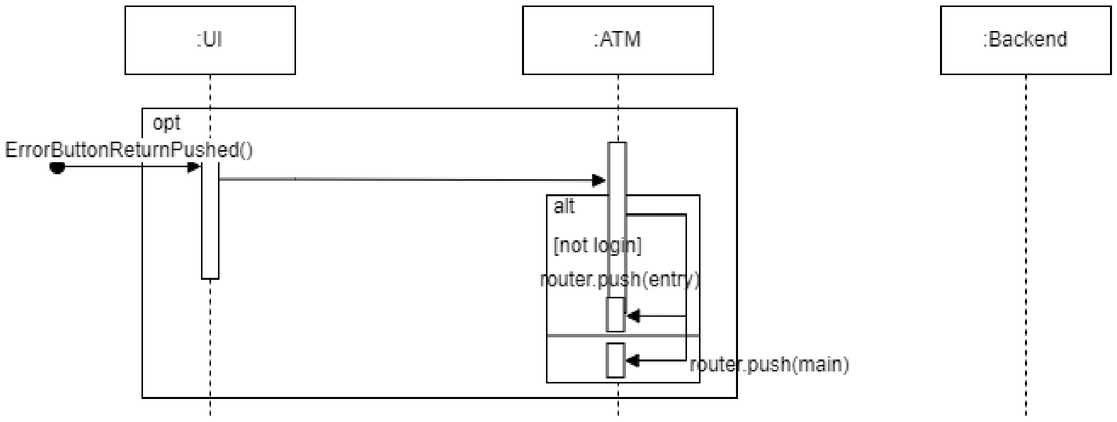

中国银行
BANK OF CHINA
BEIJING 2022
北京2022年冬奥会官方合作伙伴
Official Partner of the Olympic Winter Games Beijing 2022
中国银行手机客户端
百年中行 全球服务
登录
退出



- Press the Login button.
  1. Check health status.
     a) If an error occurs, go to Error page with error messages.
     b) Otherwise, continue the procedure.
  2. Go Login page.
- Press the Exit button.
  1. Close the APP.

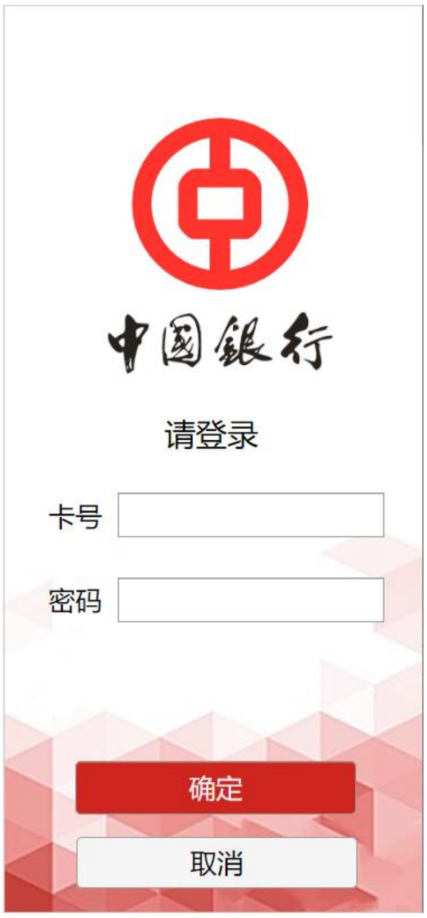
2.2.2.2 Error

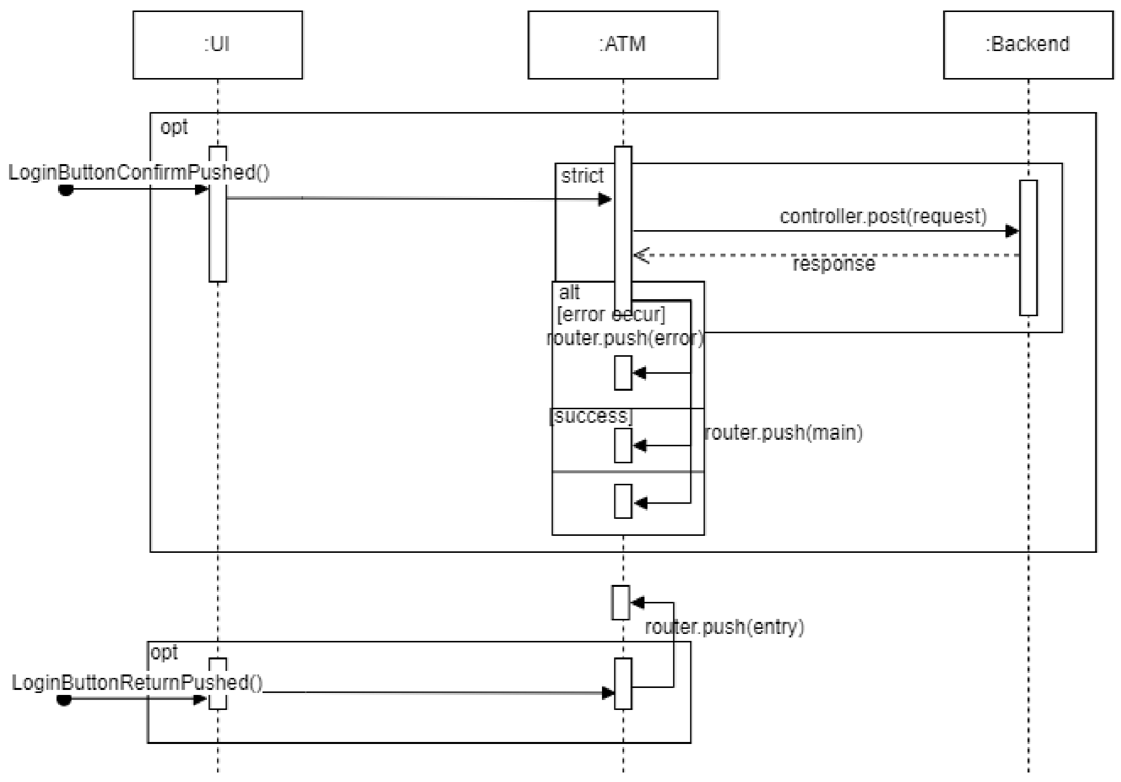




- Press the Return button.
    1. Check if user's login status.
        a) If user has logged in, go to Main page.
        b) Otherwise, go to Entry page.

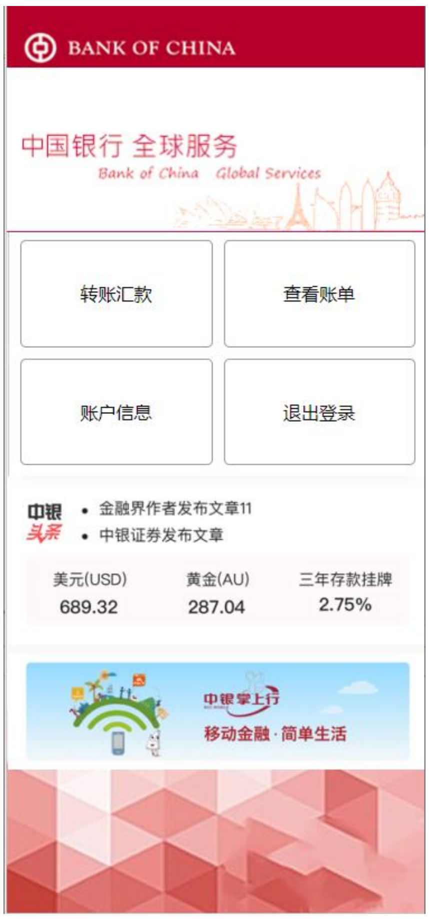
2.2.2.3 Login





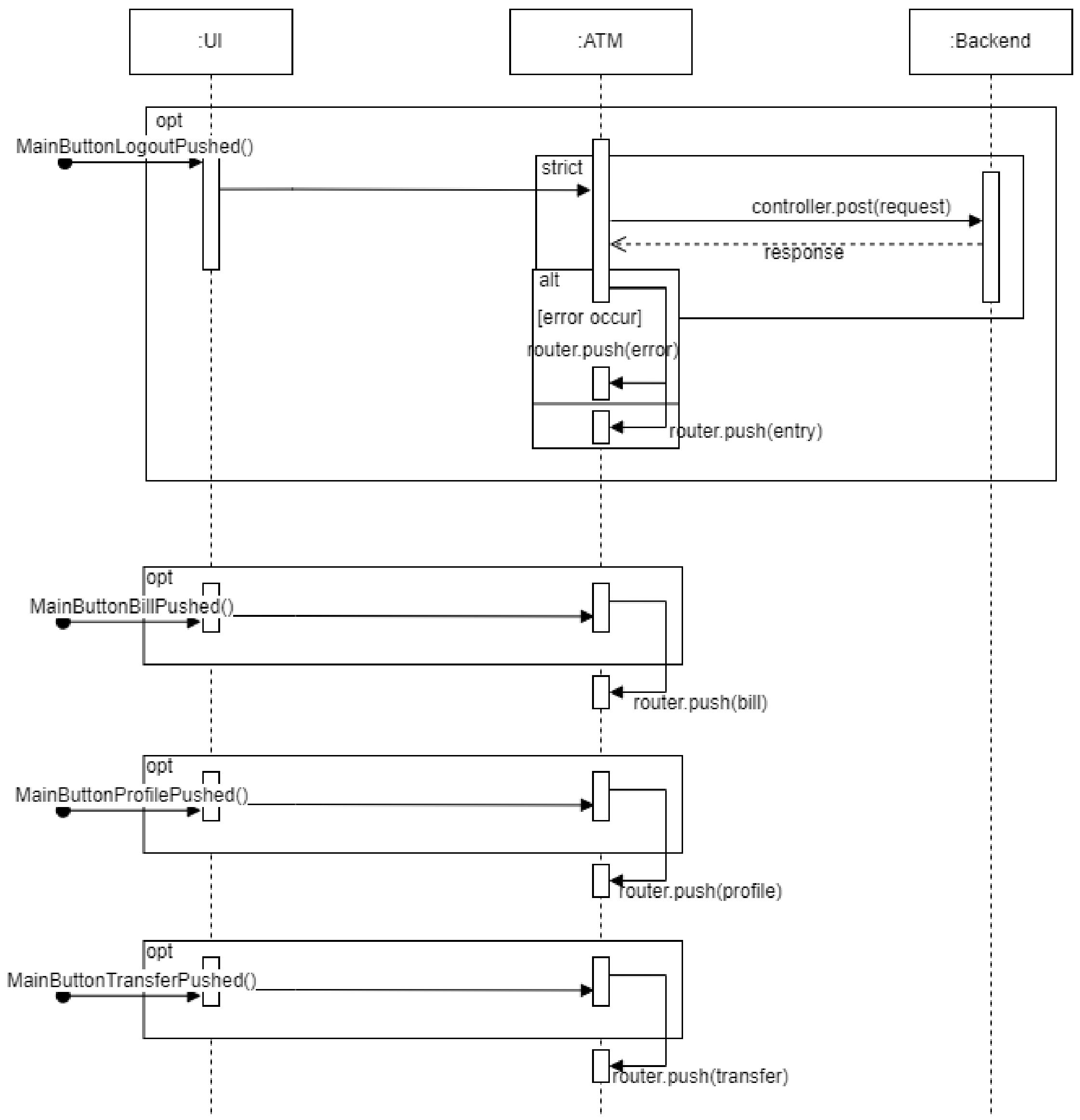
- Press the Confirm button.
  1. Verify the password and account.
     a) If they are invalid, give a hint message and return.
     b) Otherwise, continue the procedure.
  2. Send post to the backend.
     a) If an error occurs in the backend, go to Error page with error messages.
     b) If login failed in the backend, give a hint message and return.
     c) If user's APP login is not activated, then show captcha input and give hint to notice the user.
     d) Otherwise, continue the procedure.
  3. Store session id from the response and go to Main page.
- Press the Return button.
  1. Go to Entry page.

2.2.2.4 Main



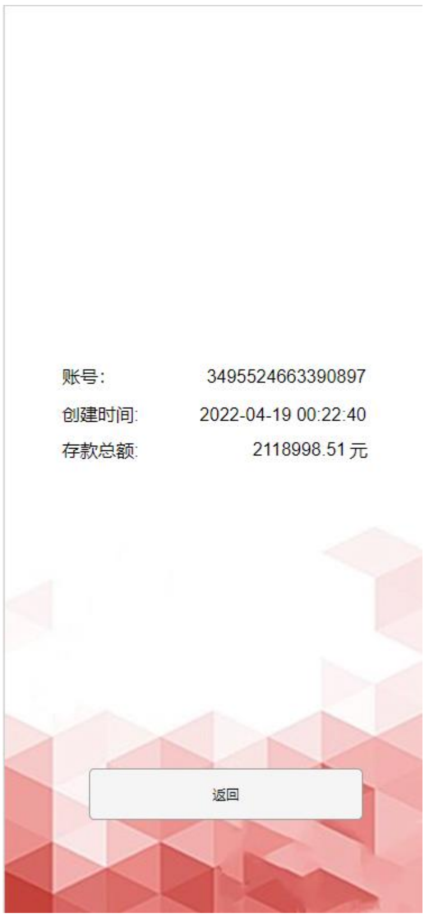
- 金融界作者发布文章11
- 中银证券发布文章

美元(USD)	黄金(AU)	三年存款挂牌
689.32	287.04	2.75%



- Press the Bill button.
  1. Go to Bill page.
- Press the Profile button.
  1. Go to Profile page.
- Press the Transfer button.
  1. Go to Transfer page.
- Press the Logout button.
  1. Send post to the backend.
     a) If an error occurs in the backend, go to Error page with error messages.
     b) Otherwise, continue the procedure.
  2. Clear session id and continue the procedure.
  3. Go to Entry page.

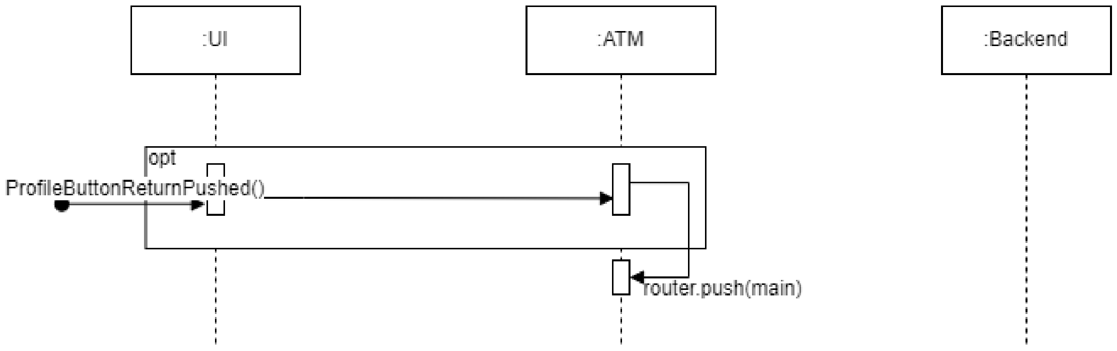
2.2.2.5 Profile



账号: 3495524663390897

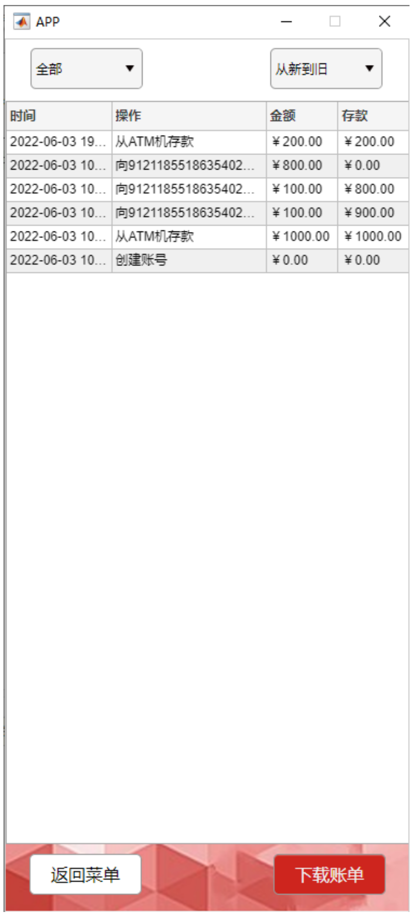
创建时间: 2022-04-19 00:22:40

存款总额: 2118998.51 元

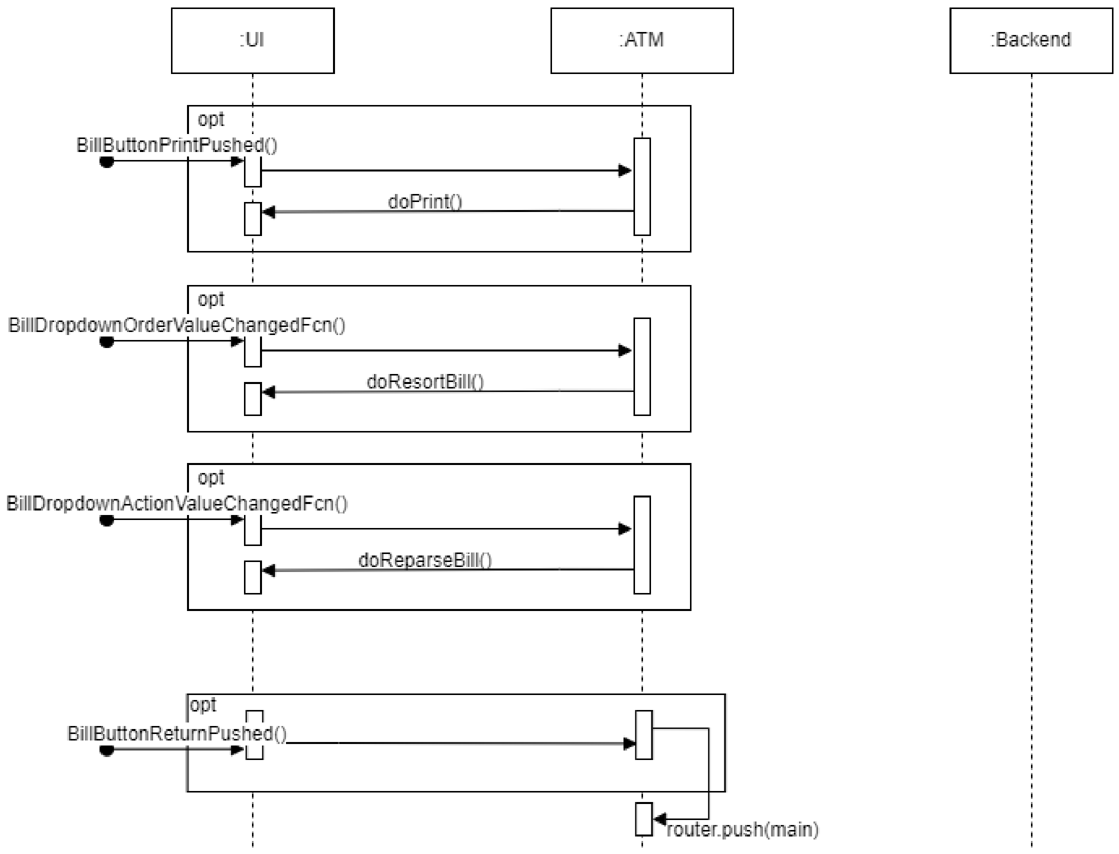


- Press the Return button.
  1. Go to Main page.

2.2.2.6 Bill

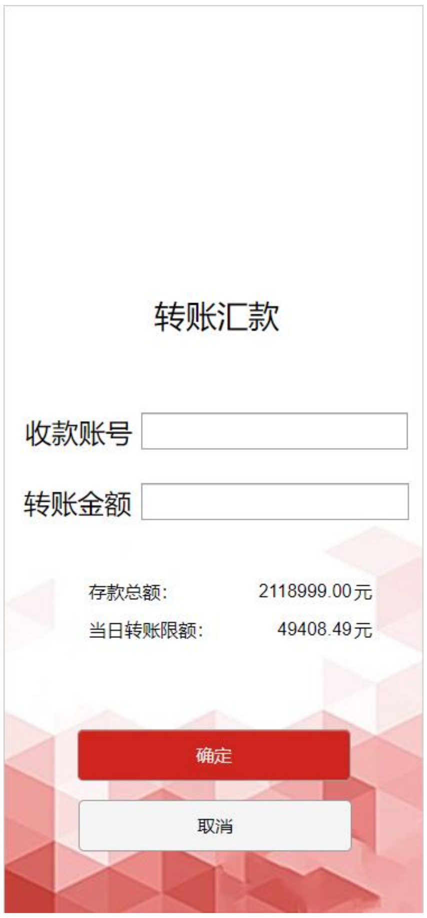


时间	操作	金额	存款
2022-06-03 19...	从ATM机存款	¥200.00	¥200.00
2022-06-03 10...	向9121185518635402...	¥800.00	¥0.00
2022-06-03 10...	向9121185518635402...	¥100.00	¥800.00
2022-06-03 10...	向9121185518635402...	¥100.00	¥900.00
2022-06-03 10...	从ATM机存款	¥1000.00	¥1000.00
2022-06-03 10...	创建账号	¥0.00	¥0.00



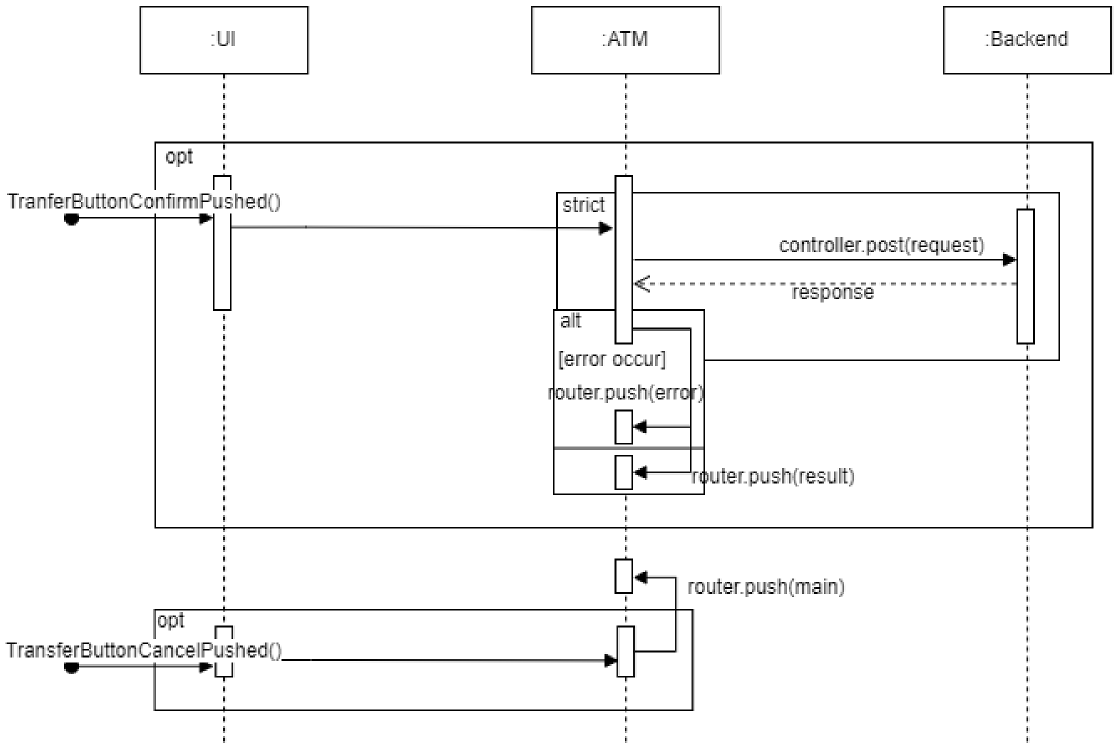
- Press the Save button.
  1. Try to save the bill into a file named bill.xlsx on the local file system.
  2. Pop out a msgbox to inform the user.
- Press the Return button.
  1. Go to Main page.
- Select the action.
  1. Reparse and update the bill.
- Select the order.
  1. Resort and update the bill.

2.2.2.7 Transfer



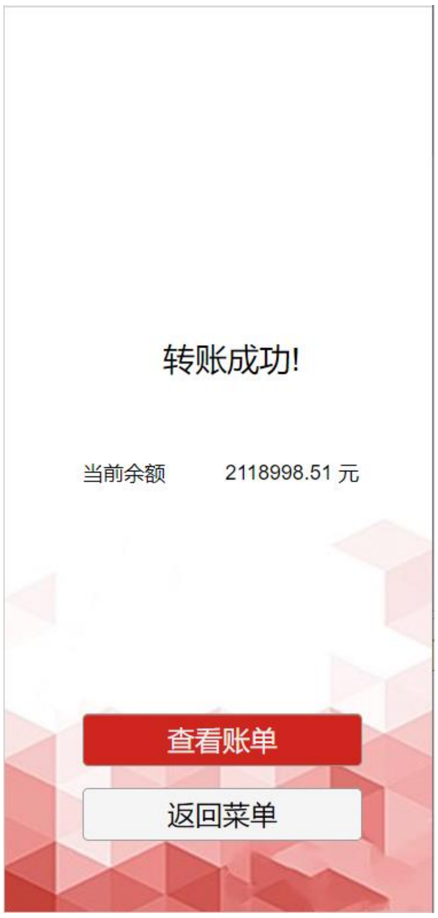
存款总额: 2118999.00元

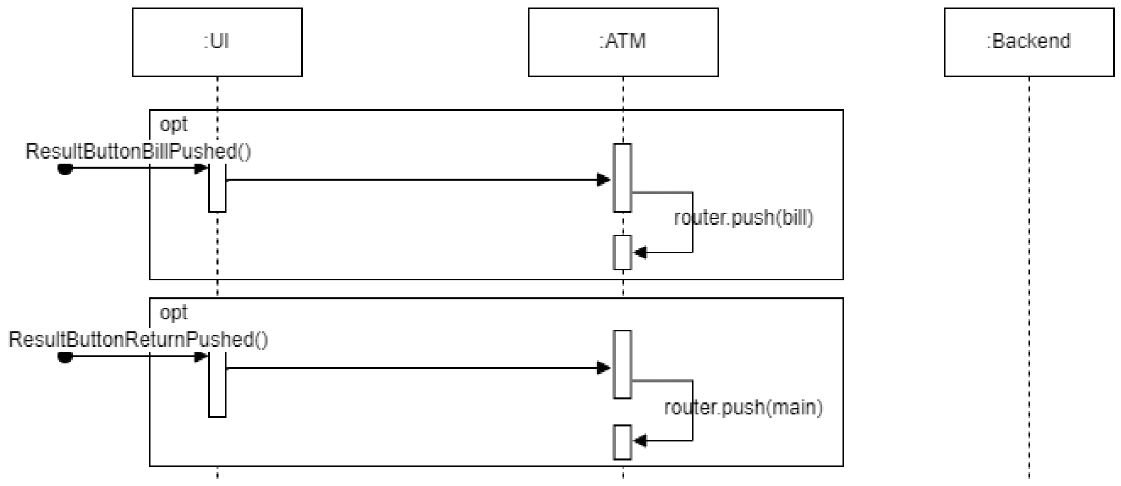
当日转账限额: 49408.49元



- Press the Confirm button.
  1. Verify the transfer target and value.
     a) If it is invalid, give a hint message, and return.
     b) Otherwise, continue the procedure.
  2. Send post to the backend.
     a) If an error occurs in the backend, go to Error page with error messages.
     b) If transfer failed in the backend, go to Error page with error messages.
     c) Otherwise, continue the procedure.
  3. Go to Result page with action transfer.
- Press the Cancel button.
  1. Go to Main page.

2.2.2.8 Result





- Press the Bill button.
  1. Go to Bill page.
- Press the Return button.
  1. Go to Main page.