移动互联网技术与应用

大作业报告

姓名：董岩
班级：2016211310
学号：2016211225

姓名：倪理涵
班级：2016211310
学号：2016211156

Contents

1. 系统开发的创意与背景

当下是移动互联网的时代，手机已是生活的必需品。合理地使用手机可以让生活更加轻松、便捷。 这学期的《移动互联网技术与应用》课程紧跟时代潮流，由来自企业，具有实际工程经验的老师 为我们讲授移动平台软件的开发，以及与服务端的数据交互技术。

这一学期，我们通过IOS平台富文本编辑器软件的开发实践，熟悉了IOS平台的软件开发流程； 之前几周进行Android平台的记账软件的开发，是一个比富文本编辑器规模稍大的软件，更使我们熟悉了 移动平台客户端软件的整体结构。

这一次的期末大作业，在上一次Android作业的基础上，增加了服务器数据同步的功能，给了我们一个 了解服务器后端框架的机会。

2. 相关技术

2.1. 开发环境

- 操作系统：Windows & Ubuntu

- 工具：Android Studio，VS Code，命令行终端
- 语言：Javascript，React（JSX），Golang

2.2. 所使用的技术

我们的应用主要使用React-Native框架开发，开发过程需要借助`React-Native-CLI`，`Android Studio`等工具，在其中涉及到的技术有：

- Javascript（ES2016版本）
- NodeJS，基于V8引擎的JS运行环境，通常用于后端和界面的开发
- React，一套用于编写UI的JS类库
- React-Native，一套使用React开发移动端本地应用的框架
- Redux，一个提供可预测程序状态的JS容器
- React-Redux，一套React和Redux相结合的框架
- Gin，一个高性能go语言后端框架

3. 系统功能需求

这个软件希望做到：

- 能详尽地记录账目信息
- 能统计账目信息
- 能以图表的方式查看统计结果
- 账目信息能够稳定保存，能在远端保存备份

我们将主要的功能需求划分为三个部分：记账功能、统计功能和同步功能。

3.1. 记账功能

记账功能需要保存具体的账目信息。一条账目具体需要保存：

- 交易时间（年月日时分）
- 交易类型（收入/支出）
- 交易项目（购物、餐饮、教育……）
- 交易金额
- 对这条交易的描述（具体消费了什么……）
- 与交易相关的图片（可以有多张）
- 交易地点（可以自动获取当前位置）

记账功能要有以下用例：

- 增添新的账单条目
    - 新添的账目初始化为默认值
- 修改已有账单条目
    - 修改交易时间
    - 修改交易类型
    - 修改交易项目
    - 修改交易金额
    - 修改对交易的描述
    - 添加或删除交易相关的图片
    - 修改交易发生的地点

- 删除一条账单条目

3.2. 统计功能

统计功能要求能够对一段时间内的交易信息加以整理，然后以较为直观的形式呈现出来。

呈现的形式分为两部分，一部分是文字信息，以列表和数字的形式呈现；另一部分是可视化信息， 以图表的形式呈现。

统计功能有以下用例:

- 查看指定月份的收入和支出总额
    - 统计一个月的收入总额
    - 统计一个月的支出总额
- 查看某一日的具体交易信息
    - 列表显示当日的所有账单
- 查看指定月份的支出和收入类别占比
    - 图表显示当月支出中各类别的占比
    - 图表显示当月收入中各类别的占比

3.3. 同步功能

同步功能要求做到账目数据的可持久化保存，数据要在软件停止运行后稳定的存储在手机中。

唯一需要保存的数据是所有的账目，因为统计信息可以依据账目信息动态生成，不需要可持久地 存储在手机上。

为了支持数据的迁移和备份，软件应有将数据上传至远端服务器和从服务器下载的功能。

4. 系统设计与实现

4.1. 系统总体设计

我们将记账软件在功能上分为四个模块：记账模块、列表统计模块、图表统计模块、和云端同步模块。

在软件结构上分为容器和视图两个部分，利用redux框架，视图部分负责显示和交互，容器部分存储软件 数据，通过接收视图传来的信号，对数据进行同步更新。

4.2. 系统物理分布

系统总体目录结构:

```
OurAccounts
├── android
│   ├── app
│   ├── build
│   ├── gradle
│   ├── keystores
│   ├── build.gradle
```

```
│   ├── gradle.properties
│   ├── gradlew
│   ├── gradlew.bat
│   ├── local.properties
│   ├── OurAccounts.iml
│   └── settings.gradle
├── node_modules
├── OurAccountsServer
│   ├── certs
│   │   ├── server.crt
│   │   └── server.key
│   ├── vendor
│   ├── Gopkg.lock
│   ├── Gopkg.toml
│   └── main.go
├── src
│   ├── models
│   │   ├── AccountsReducer.js
│   │   ├── MonthsReducer.js
│   │   ├── StatisticsReducer.js
│   │   └── SyncReducer.js
│   ├── views
│   │   ├── AccountEdit.js
│   │   ├── Accounts.js
│   │   ├── HomeScreen.js
│   │   ├── Months.js
│   │   ├── MonthsDetail.js
│   │   ├── Signup.js
│   │   ├── Statistics.js
│   │   ├── Sync.js
│   │   ├── User.js
│   │   └── style.js
│   ├── App.js
│   └── reducer.js
├── __tests__
│   └── App-test.js
├── app.json
├── babel.config.js
├── index.js
├── metro.config.js
├── package.json
├── yarn-error.log
└── yarn.lock
```

index.js是主体程序的入口，其余的源代码都存放在src文件夹中。android文件夹保存Android 项目信息，主要是React-Native框架自动生成的代码和配置文件。

src/views目录保存各组件的视图界面代码；src/models目录保存各组件数据的容器代码； src/reducer.js保存Redux框架中程序的总数据容器；src/App.js将数据容器与视图部分结合。

4.3. 模块设计

4.3.1. 主界面

入口

整个应用的入口是index.js文件，index.js中注册src/App.js中定义的App组件。 App组件是真正的应用程序。

```
import {AppRegistry} from 'react-native';
import App from './src/App';
import {name as appName} from './app.json';

AppRegistry.registerComponent(appName, () => App);
```

src/App.js文件在数据容器与用户界面结合在一起。通过React-Redux框架的Provider组件 向用户界面提供数据。利用redux-persist提供的PersistDate

实现账目数据在手机本地的 可持久化存储。AppContainer是主界面组件。store是全局数据存储容器。

```
import AppContainer from './views/HomeScreen';
import { store, persistor } from './reducer';

const App = () => {
  return (
    <Provider store={store}>
      <PersistGate loading={<ActivityIndicator size="large"/>} persistor={
        <AppContainer />
      </PersistGate>
    </Provider>
  );
}

export default App;
```

数据容器

/src/reducer.js核心代码，创建全局数据容器并做可持久化处理。

```
const reducer = combineReducers({
  accountInfo: persistReducer(accountsPersistConfig, accountsReducer),
  monthInfo: monthsReducer,
  statisticsInfo: statisticsReducer,
});

const persistedReducer = persistReducer(persistConfig, reducer);

const store = createStore(persistedReducer);
const persistor = persistStore(store);
```

主界面

src/views/HomeScreen保存主界面。主界面由标题栏，正文和底部栏构成。

```
import AccountsView from './Accounts';
import MonthsView from './Months';
import StatisticsView from './Statistics';
import SyncView from './Sync';

/* 创建一个页面导航界面 */
const HomeNavigator = createBottomTabNavigator({
    /* 主界面的正文 */
  accounts: { screen:  AccountsView },      // 记账页面
  months: { screen: MonthsView },           // 列表统计页面
  statistics: { screen: StatisticsView },   // 图表统计页面
  sync: { screen: SyncView },               // 同步页面
}, {
    initialRouteName: 'accounts',
        /* 主界面底部栏 */
    tabBarComponent: props => {
      return (
        <Footer>
          <FooterTab>
            <Button>{/* 记账页面按钮 */}</Button>
            <Button>{/* 列表统计页面按钮 */}</Button>
            <Button>{/* 图表统计页面按钮 */}</Button>
            <Button>{/* 同步页面按钮 */}</Button>
          </FooterTab>
        </Footer>
      )
    }
  }
);
```

4.3.2. 记账模块

记账模块提供账单列表界面和账单编辑界面。账单列表界面显示所有的账目，可以在添加或删除一条账目。 账单编辑界面显示一条具体的账目信息，包括交易发生的日期、时间，交易类型、项目、金额、种类， 以及相关图片和交易发生的地址。

账单列表

账单列表部分对应src/views/Accounts.js文件。

核心代码如下。Container，Content， Button均是native-base提供的组件。 AccountList是自定义的列表界面。Accounts会从记账模块的数据容器中获取账目列表数据accounts 和一些可调用的函数如onClickDel，并将这些属性传递给子组件使用。

```
            <Container>
                <Content>
                    /* 添加新条目按钮 */
                    <Button onPress={...}>  // 调用回调函数
                        <Icon .../>
                        <Text>添加</Text>
                    </Button>
                    /* 账目列表 */
                    <AccountList
                        accounts={accounts} // 传递账目数据
                        onClickDel={...}
                        onClickEdit={...}
                    />
                </Content>
            </Container>
}
```

定义AccountList组件的代码如下，使用了react-native提供的基本组件FlatList作为列表界面。 AccountItem是自定义的账单条目的视图。AccountList从父组件处接收accounts和回调函数， 将其传给AccountItem。

```
        /* 对FlatList做封装，得到AccountList组件 */
        <FlatList
            data={accounts}
            renderItem={({item, index}) => (
                /* 每一个账单条目 */
                <AccountItem
                    account={item}        // 传递账单条目
                    index={index}
                    onClickDel={onClickDel}
                    onClickEdit={onClickEdit}
                />
            )}
        />
```

定义AccountItem的代码如下，AccountItem中定义了列表中应显示的简略账目信息， 以及左滑时出现的删除按钮。AccountItem接收accounts等属性，用于显示和交互。

```
        /* 将可滑动行做封装，得到AccountItem */
        <SwipeRow
      ...
            /* 条目简略信息 */
            body={
                <Button
          ...
                    onPress={() => onClickEdit(index)}>
                    <H2>
                        条目{index}:
                    </H2>
                    <Text>
```

```
                    {/* 显示简略的账目信息 */}
                </Text>
            </Button>
        }
        /* 左滑时出现的删除按钮 */
        right={
            <Button
    ...
                onPress={() => onClickDel(index)}>
                <Icon active name='trash' />
            </Button>
        }
    />
```

账单编辑

账单编辑部分对应src/views/AccountEdit.js文件。

核心代码如下，在一个表单里定义了各编辑组件。 Form、Item、Input、Label、Picker、Image等均为react-native或native-base 提供的基本组件。 MyDatePicker、MyTimePicker是自定义的选择日期和时间的组件。AccountEdit 从数据容器中接收属性accountData和一些回调函数。accountData表示当前修改的账单条目。

```
<Container>
  <Content>
    <Form>
                    /* 选择日期的组件 */
      <Item ...>
        <Label>日期</Label>
        <MyDatePicker ...
          date={accountData.date} onChangeDate={onChangeDate} />
      </Item>
                    ...
                    /* 选择账目类型的组件 */
      <Item ...>
        <Label>账目类型</Label>
        <Picker
          ...
          selectedValue={
            accountData.isIncome === 'undefined' ? true : accountDat
          }
          onValueChange={(itemValue) => onChangeType(itemValue)}>
          <Picker.Item label='收入' value={true} />
          <Picker.Item label='支出' value={false} />
        </Picker>
      </Item>
                    /* 选择消费种类（项目）的组件 */
      <Item ...>
        <Label>消费种类</Label>
        <Picker
          ...
          selectedValue={accountData.item}
          onValueChange={(itemValue) => onChangeItem(itemValue)}>
          <Picker.Item label='购物' value={'购物'} />
          <Picker.Item label='餐饮' value={'餐饮'} />
          <Picker.Item label='服装' value={'服装'} />
          <Picker.Item label='生活' value={'生活'} />
                        ...
                        ...
        </Picker>
      </Item>
                    ...
                    ...
    </Form>
  </Content>
</Container>
```

账单数据的存储容器

账单数据的保存应用了Redux框架，即程序中只有唯一的一份数据容器，且只能通过 回调函数间接操作数据内容，不允许在用户界面中对数据直接修改。

账目信息对应的数据容器实现在src/models/AccountReducer.js中。

定义账单数据类型。

```
class AccountData {
  constructor({key}) {
    this.key = key; // string
    this.date = moment(new Date()).format('YYYY-MM-DD'); // Date
    this.time = moment(new Date()).format("LT"); // Date
    this.isIncome = false; // boolean: is income or expense
    this.amount = "0" // string: the amount of money
    this.item = '购物' // string: on what item the transaction is
    this.desc = undefined // string: description of the transaction
    this.imgPaths = []; // array(string) paths of images
    this.position = undefined; // Position: the geolocation where the tran
  }
}
```

定义默认的容器数据。

```
const INITIAL_STATE = {
  next_key: 0,
  accounts: [],
  index: 0,
  accountData: new AccountData({}),
};
```

定义所有可能发生的数据操作行为。

```
const accountsReducer = (state = INITIAL_STATE, action) => {
  switch (action.type) {
    case "account_add": return handleAdd(state);
    case "account_del": return handleDel(state, action);
    case "account_edit": return handleEdit(state, action);
    case "account_save": return handleSave(state);
    case "account_edit_date": return handleEditDate(state, action);
    case "account_edit_time": return handleEditTime(state, action);
    case "account_edit_type": return handleEditType(state, action);
    case "account_edit_amount": return handleEditAmount(state, action);
    ...
        ...
  }
  return state;
}
```

4.3.3. 列表统计模块

列表统计模块将每月的总收入和总支出显示给用户。为了方便地查看某一日的具体开支，还要能提供快捷的日期跳转功能。

在react-native-calendars中，提供了Calendar组件以显示日历。日历具有点击左右箭头按钮切换月份、单击日期触发事件等功能。

```
<Calendar
    onDayPress={(day) =>
        // 点击日期切换至消费详细
        onClick(day, () => {
            // console.warn(day);
            navigation.navigate('monthsDetail');
        })
    }
    monthFormat = { 'yyyy年M月' }
    onMonthChange = {(month) => {
        onChange(month);
        onIncome(accounts);
        onExpense(accounts);
```

```
                }}
            />
```

同时，为了实现UI上的复用，避免多个模块风格不一致， 在Months类型被封装为用户可见的MonthsView前，我们使用react-navigation中的功能指定其“标题栏”的外观。

```
static navigationOptions({navigation}) {
      return {
          title: 'Months',
          header: (
              <Header>
                  <Left />
                  <Body>
                      <Title>月份</Title>
                  </Body>
                  <Right />
              </Header>
          )
      };
}
```

在列表统计模块内部的触发事件可分为以下几类。

- 点击日历上的具体日期
    - 此时更新状态中的年、月、日，对应到该日期，通过年月日筛选出该日期下的账目
    - 触发从当前视图MonthsView_转移到MonthsDetailView的事件
- 点击日历的左右切换月份按钮
    - 更新状态中的月
    - 更新当前月的总收入和总支出

```
const mapDispatchToProps = (dispatch) => ({
    onClick: (day, callBack) => {
        dispatch({ type: 'year_select', year: day.year });
        dispatch({ type: 'month_select', month: day.month });
        dispatch({ type: 'day_select', day: day.day });
        dispatch({ type: 'month_watch', callBack: callBack })
        console.log('WATCH');
    },
    onChange: (month) => {
        dispatch({ type: 'month_change', month: month.month });
    },
    onIncome: (accounts) => {
        dispatch({ type: 'month_income', accounts: accounts });
    },
    onExpense: (accounts) => {
        dispatch({ type: 'month_expense', accounts: accounts });
    },
});
```

点击某一具体日期后，查看具体账目所涉及的逻辑在MonthsDetail.js中。在AccountData这一对象类型数据中，date域为形如"YYYY-MM-DD"的字符串。需要注意的是，诸如6、7月在字符串中也会变为06、07。据此可以写出筛选当日账目并按具体时间排序显示的逻辑。

```
        var res = [];
        for (var i = 0; i < accounts.length; ++ i) {
            if ((month < 10 && year + "-0" + month + "-" + day == accounts
            (month >= 10 && year + "-" + month + "-" + day == accounts[i].
                res.push(accounts[i]);
            }
        }
        res.sort(function(a, b){return a.time - b.time});
```

4.3.4. 图表统计模块

我们以饼状图展现每月中，各收入/支出类别在当月总收入/支出的占比。

在react-native-chart-kit中，PieChart组件能够满足我们的需求。 在StatisticsReducer.js中，保存的状态用于图表统计模块的制图。 与列表统计模块不同的是，图表统计模块中使用CalendarList，无需点按左右箭头按钮切换月份，而是直接左右滑动屏幕即可。

状态中，categories中参与统计的收支类型需要与之前记账模块中内置的类型保持一致，income和expense两个数据分别用于收入图和支出图。 但因为所有数据全部为0时，在渲染时会报错，因此我们暂时将“其他”项这里修改为非0。 这只是权宜之计，实际统计后往往会被覆盖掉。

```
const INITIAL_STATE = {
  month: moment().format('YYYY-MM'),
  categories: [
    { name: '购物', income: 0, expense: 0, },
    { name: '餐饮', income: 0, expense: 0, },
    { name: '服装', income: 0, expense: 0, },
    { name: '生活', income: 0, expense: 0, },
    { name: '教育', income: 0, expense: 0, },
    { name: '娱乐', income: 0, expense: 0, },
    { name: '出行', income: 0, expense: 0, },
    { name: '医疗', income: 0, expense: 0, },
    { name: '投资', income: 0, expense: 0, },
    { name: '其他', income: 1, expense: 1, },
  ],
  yearData: [],
};
```

我们分别对总收入占比和总支出占比进行饼状图制图。

```
  <H3 style={{paddingLeft: 50,}}>月收入统计</H3>
  <PieChart
          ...
    data={ctg_income}
    accessor='population'
    backgroundColor='transparent'
  />

  <H3 ...>月支出统计</H3>
  <PieChart
          ...
    data={ctg_expense}
    accessor='population'
    backgroundColor='transparent'
  />
```

最后，在切换月份使得month状态刷新时，我们也要能即时刷新图表显示。这一步的逻辑与列表统计模块类似。 首先更新month属性，然后用更新过的时间属性筛选出对应时间范围内的账目，用它们更新categories中的各条目的收入支出。

4.3.5. 云端同步模块

我们的云端同步模块支持上传数据和下载数据功能。

同步功能需要用账户来同步数据。用户通过注册界面注册账户，或是通过登陆界面登陆账户。进入账户以后， 可以选择注销，上传数据或者下载数据。

在底部的导航栏“我的”项中，首先进入的是登录页面，这一页面提供了两个文本输入框，用来输入用户名和密码，以及两个功能按钮 ——在点按登录按

钮后，如果用户名和密码正确，则会进入用户的数据同步页面，否则弹出提示框通知用户这一错误。

```
<View
  style = {styles.container}
>
  {/* 账号输入框 */}
  <View style={[styles.view, styles.lineTopBottom]}>
    <Text style={styles.text}>
      账号
    </Text>

    <TextInput
      style={styles.textInputStyle}
      placeholder="请输入用户名"     // 没有任何文字输入时显示
      secureTextEntry={false}    // 是否敏感
      onChangeText={onChangeName}     // 文本框内容变化时调用
      value={name}
    />
  </View>

  {/* 密码输入框 */}
  <View style={[styles.view, styles.lineTopBottom]}>
    <Text style={styles.text}>
      密码
    </Text>

    <TextInput
      style={styles.textInputStyle}
      placeholder="请输入密码"
      secureTextEntry={true}
      onChangeText={onChangePswd}
      value={pswd}
    />
  </View>
</View>

{/* 登录按钮 */}
<View style={styles.buttonView}>
  <TouchableOpacity
    style={styles.button}
    onPress={() => signIn(() => navigation.navigate('userSync'))}     // 登
  >
    <Text style={styles.buttonText}>登录</Text>
  </TouchableOpacity>
</View>

{/* 注册按钮 */}
<View style={styles.buttonView}>
  <TouchableOpacity
    style={[styles.button, {backgroundColor: "yellow"}]}
    onPress={() => {navigation.navigate('signUp')}}     // 跳转至注册
  >
    <Text style={styles.buttonText}>注册</Text>
  </TouchableOpacity>
</View>
```

还有注册按钮，点击注册按钮后，会进入用户注册页面，包含三个输入框，用来输入用户名，密码以及重复密码。 注册模块会对用户名的合法性以及是否重复，还有两次密码的输入是否一致进行检查。

```
<View
  style = {styles.container}
>

  <View style={styles.inputView}>
    {/* 账号输入框 */}
    <View style={[styles.view, styles.lineTopBottom]}>
      <Text style={styles.text}>
        账号
```

```
      </Text>

      <TextInput
        style={styles.textInputStyle}
        placeholder="6~16位，仅包含数字和字母，区分大小写"     // 没有任何文字输入时
        secureTextEntry={false}     // 是否敏感
        onChangeText={onSignName}     // 文本框内容变化时调用
        value={sname}
      />
    </View>

    {/* 密码输入框 */}
    <View style={[styles.view, styles.lineTopBottom]}>
      <Text style={styles.text}>
        密码
      </Text>

      <TextInput
        style={styles.textInputStyle}
        placeholder="6~16位，仅包含数字和字母，区分大小写"
        secureTextEntry={true}
        onChangeText={onSignPswd}
        value={spswd}
      />
    </View>

    {/* 重复密码输入框 */}
    <View style={[styles.view, styles.lineTopBottom]}>
      <Text style={styles.text}>
        重复密码
      </Text>

      <TextInput
        style={styles.textInputStyle}
        placeholder="请确认两次密码输入一致"
        clearButtonMode="while-editing"
        secureTextEntry={true}
        onChangeText={onSignReppswd}
        value={reppswd}
      />
    </View>
  </View>

  {/* 注册按钮 */}
  <View style={styles.buttonView}>
    <TouchableOpacity
      style={styles.button}
      onPress={() => signUp(navigation.goBack)} // 注册功能
    >
      <Text style={styles.buttonText}>注册</Text>
    </TouchableOpacity>
  </View>

</View>
```

同步模块分为3个页面，因此共有三个文件分别实现UI及功能：Sync.js，User.js，Signup.js。以上已经给出了Sync.js和Signup.js的主要逻辑。而在User.js中，主要实现账目数据的上传与下载，以及注销功能——点按“退出”按钮后，界面会跳转到原先的登录界面。

需要说明的是，登录、注册、上传以及下载这4大功能，在服务器端制定了相应的API：signin、signup、syncup、syncdown。在react-native中，我们以fetch来向服务器发送请求，以实现这4大API。

所有的请求都是POST类型。请求内部的body是相应参数的JSON格式。

```
const handleSignIn = (state, {callBack}) => {
    if (state.name === '' || state.pswd === '') {
        Alert.alert('账号和密码不能为空!')
```

```
        return state
    }

    fetch('http://49.234.16.186:60000/signin', {
        method: 'POST',
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({
            name: state.name,
            pswd: state.pswd,
        })
    }).then((response) => {
        console.log("response: ")
        console.log(response)
        console.log("http status code: " + response.status)

        if (response.status == 200) {    //  账号密码正确，跳转至账户数据同步页面
            callBack()
        }
        else if (response.status == 400) {    // 账号密码错误
            Alert.alert('账号或密码错误!')
            return
        }
        else {    // 非程序内置逻辑
            Alert.alert('看到这个说明出 bug 啦!')
        }
    }).catch((error) => {
        console.error(error)
    })

    return state
}

const handleSignUp = (state, {callBack}) => {
    if (state.spswd != state.reppswd) {
        Alert.alert('两次密码输入不一致!')
        return state
    }

    if (state.sname === '' || state.spswd === '') {
        Alert.alert('账号和密码不能为空!')
        return state
    }

    fetch('http://49.234.16.186:60000/signup', {
        method: 'POST',
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({
            name: state.sname,
            pswd: state.spswd,
        })
    }).then((response) => {
        console.log("response: ")
        console.log(response)
        console.log("http status code: " + response.status)

        if (response.status == 200) {    //  账号密码正确，跳转回登录界面
            Alert.alert('注册成功!')
            callBack()
            return {...state, sname: '', spswd: '', reppswd: ''}
        }
        else if (response.status == 400) {    // 账号密码错误
            Alert.alert('账号已存在!')
        }
        else {    // 非程序内置逻辑
            Alert.alert('看到这个说明出 bug 啦!')
        }
    }).catch((error) => {
        console.error(error)
```

```
    })

    return state
}
```

以数据的上传为例，body中不仅要含有name和pswd以验证身份，还要将当前的账目数据accounts转化为JSON格式作为同步的数据data。

```
body: JSON.stringify({
  name: state.name,
  pswd: state.pswd,
  data: JSON.stringify(accounts),
})
```

4.3.6. 服务器端

我们的服务器端为客户端保存数据备份。客户端通过向服务端发送http请求来完成注册、登陆认证和数据同步。

我们在服务端使用GO语言作为开发语言，使用Gin来搭建服务端框架。数据库我们采用了MongoDB， 使用mongo-go-driver将go与MongoDB连接。

服务端核心代码如下。我们首先建立与MongoDB的连接，然后初始化一个Gin引擎，设置路由。 /signin用作登陆，/signup用作注册，/syncup用作上传数据，/syncdown用作下载数据。

具体代码参照OurAccountsServer/main.go文件。

```
func main() {
        // Set up DB
        clientOptions := options.Client().ApplyURI("mongodb://localhost:27

        client, err := mongo.Connect(context.TODO(), clientOptions)

        if err != nil {
                log.Fatal(err)
        }

        // Check the connection
        err = client.Ping(context.TODO(), nil)

        if err != nil {
                log.Fatal(err)
        }

        fmt.Println("Connected to MongoDB!")

        // Set up router
        router := gin.Default()

        config := cors.DefaultConfig()
        config.AllowAllOrigins = true

        router.Use(cors.New(config))

        router.POST("/signin", func(c *gin.Context) {
                handleSignIn(c, client)
        })
        router.POST("/signup", func(c *gin.Context) {
                handleSignUp(c, client)
        })
        router.POST("/syncup", func(c *gin.Context) {
                handleSyncUp(c, client)
        })
        router.POST("/syncdown", func(c *gin.Context) {
                handleSyncDown(c, client)
        })

        go router.RunTLS(":60001", "./certs/server.crt", "./certs/server.k
```

```
        router.Run(":60000")
}
```

5. 系统可能的拓展

1. 自动同步功能：支持登陆账户以后自动切换账目内容；支持用户修改账目之后自动上传至云服务器。

2. 导出功能：支持将选定日期范围内的账目导出至.csv或.xls格式的文件中，并能方便转移到其他平台。

3. 更详细的统计：支持更多维度的统计数据，如年消费项目占比，消费额变化曲线等。

4. 设置功能：制定多套显示模式、UI界面等，让用户自定义自己的记账软件。

6. 总结体会

这次综合作业集成了之前的Android作业，加入了Go语言服务器部分的代码，让我们在了解客户端开发的 同时也熟悉了服务端的开发框架。在期末阶段，对于学生来说，完成这些大作业是有着不小的压力， 但是收获也相应的非常巨大。

首先锻炼的是压力下高效产出代码的能力。我们小组两人在两周的时间内完成了记账软件的开发，对于 相对还缺乏开发经验的我们来说是个比较令人满意的结果。在这一学期的工程训练之后，不少同学对 项目的开发已经有了一定程度的熟悉和了解，这一点对于未来融入工作环境非常有益。

其次是学到了一些新技术。我们在开发过程中对React和Redux框架越来越熟悉。 在实现各种功能时，我们接触了不少React，React-Native框架的组件，深刻地认识到了开源社区的力量。 开源社区相互友好协助的氛围更有利于新技术的传播和改进。