Full Stack Development Term Project

Team:
Financial Tracker (Walletdog)

Teammates

Ke Wang	G21939527, Wangke@gwu.edu
Mengchen Pan	G49286494, mcpan@gwu.edu
Pei He	G34870360, phe@gwmail.gwu.edu
Yang Liu	G36939559, yangliu1989@gwmail.gwu.edu
Yuxin Kang	G21350525, kyx@gwu.edu

Introduction

Purpose

Track and manage personal expense in order to help users make a better budget and save money.

Function

User management

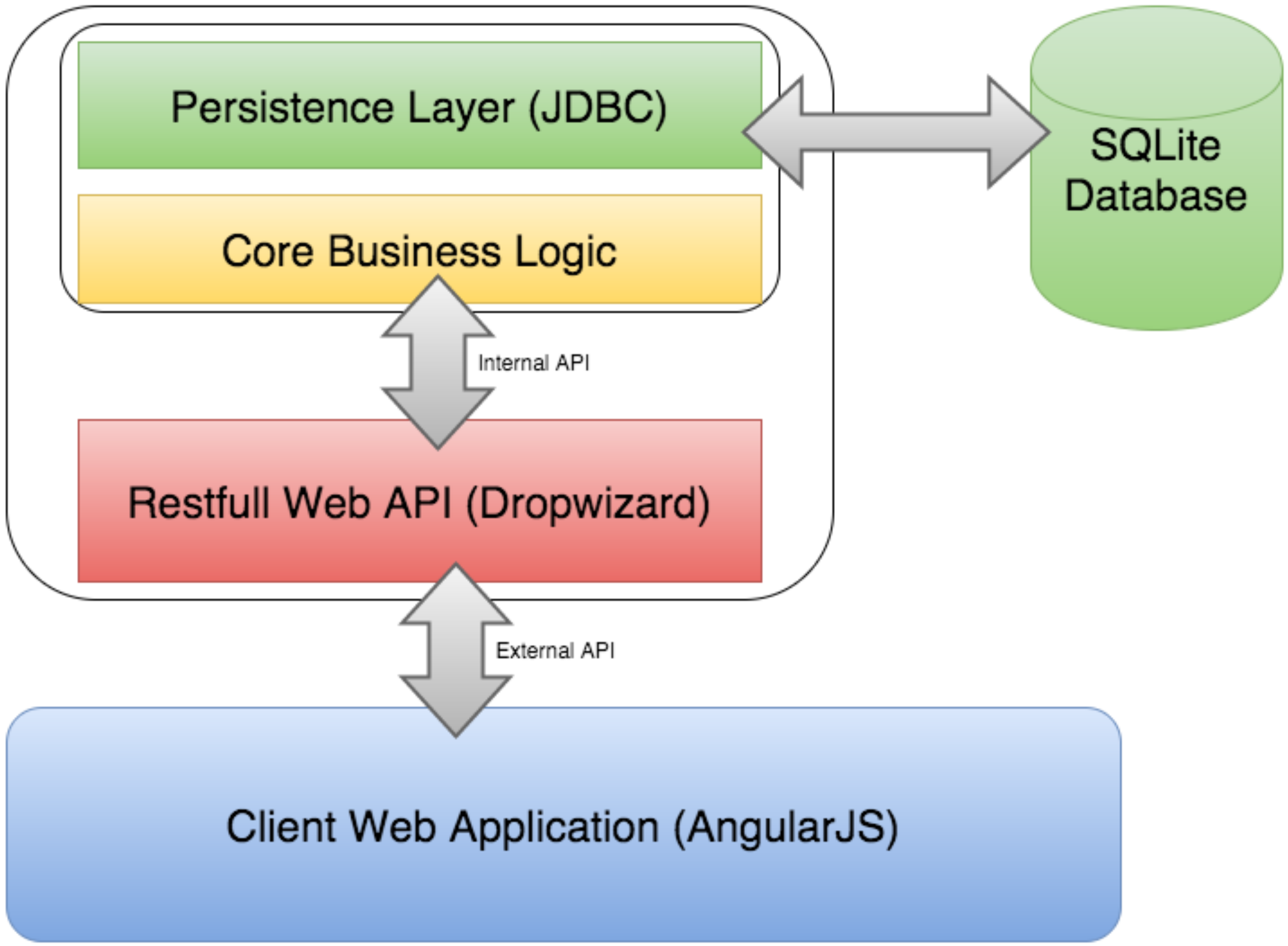
Create spending categories

Post expenses

View statistical charts of spendings

Img1 : save-spend-invest

Architecture - Service Oriented Architecture



Decisions

We choose JDBC instead of ORMs

We choose Dropwizard instead of Spring MVC

We choose SQLite instead of MySQL

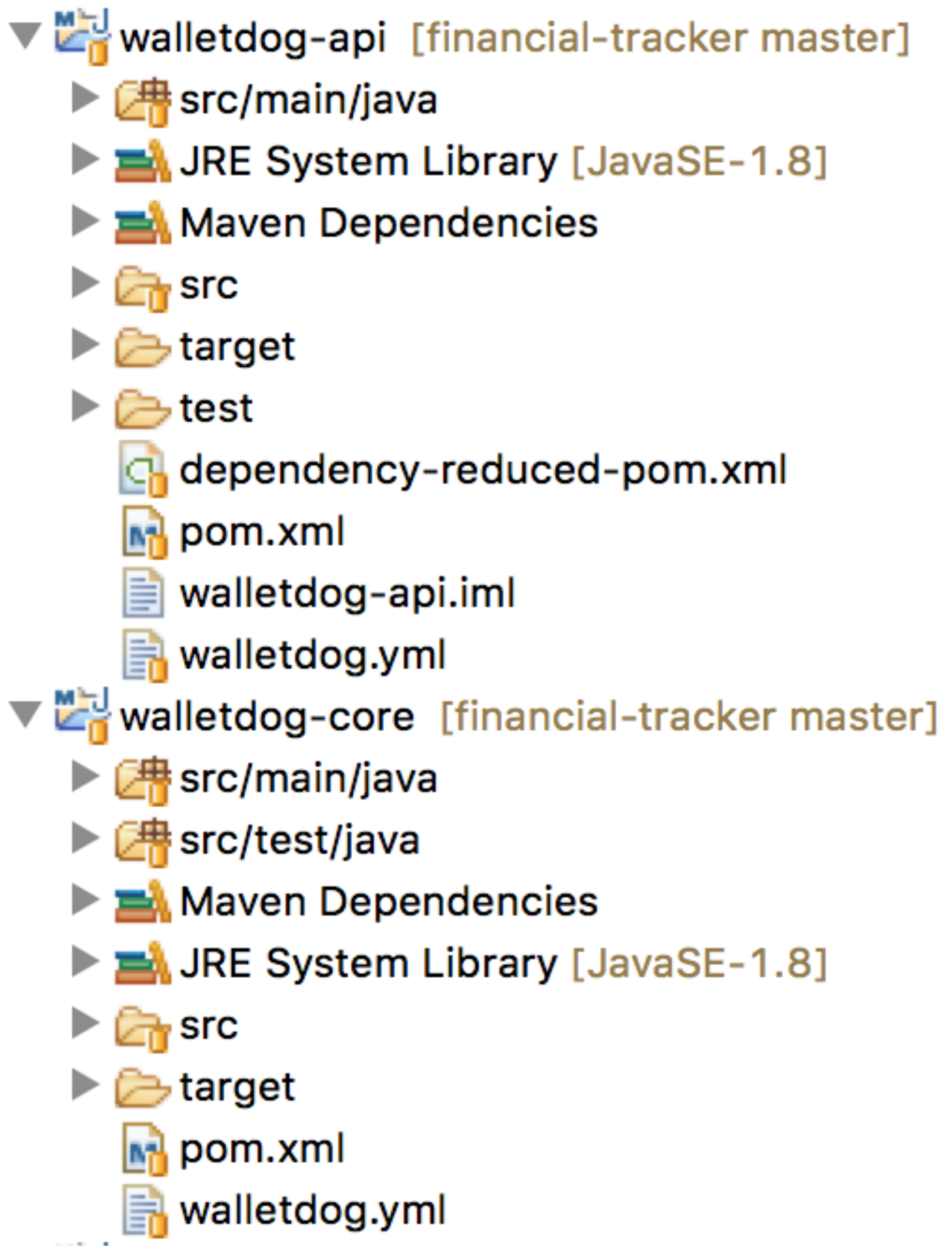
We choose Microservice architecture instead of Monolithic architecture

Backend Review

We separate our backend project into two parts :

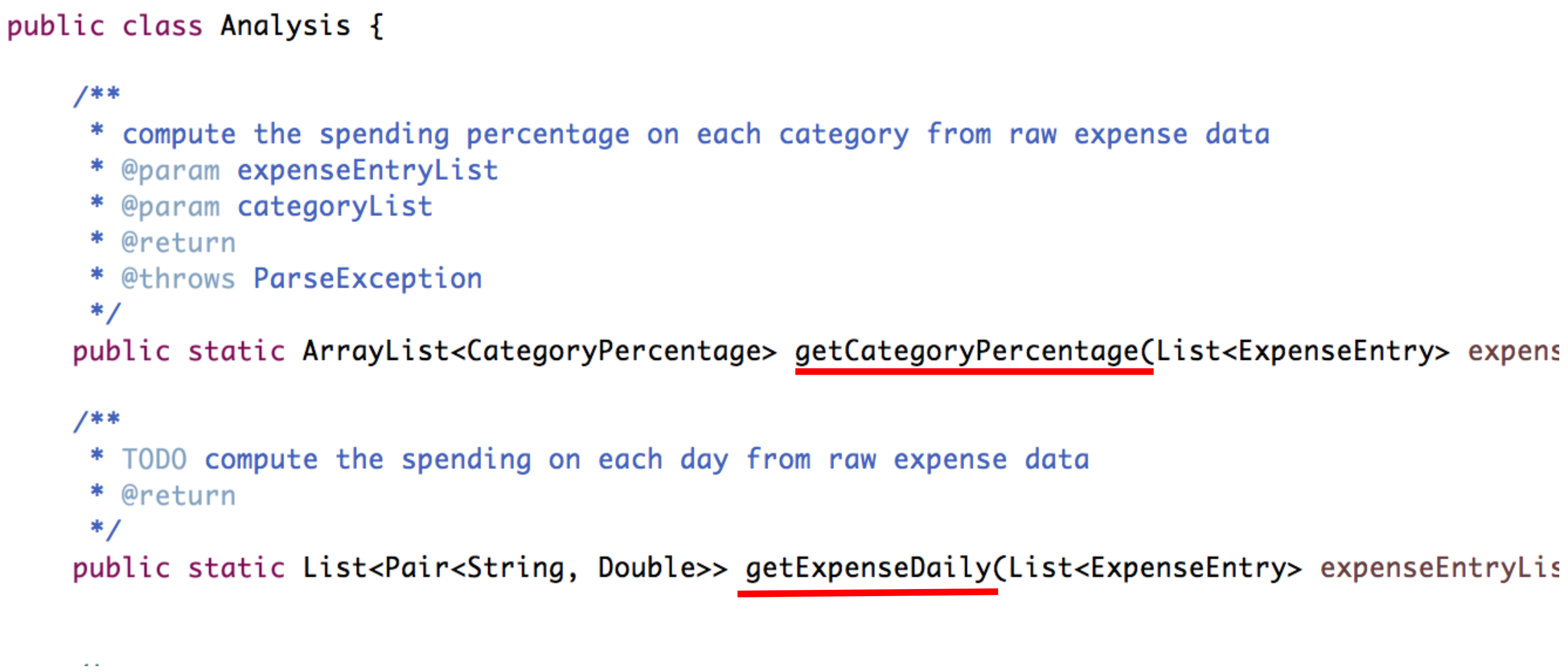
Core Module

Web API Module



```
▼ walletdog-api [financial-tracker master]
  ▶ src/main/java
  ▶ JRE System Library [JavaSE-1.8]
  ▶ Maven Dependencies
  ▶ src
  ▶ target
  ▶ test
    dependency-reduced-pom.xml
    pom.xml
    walletdog-api.iml
    walletdog.yml
▼ walletdog-core [financial-tracker master]
  ▶ src/main/java
  ▶ src/test/java
  ▶ Maven Dependencies
  ▶ JRE System Library [JavaSE-1.8]
  ▶ src
  ▶ target
    pom.xml
    walletdog.yml
```

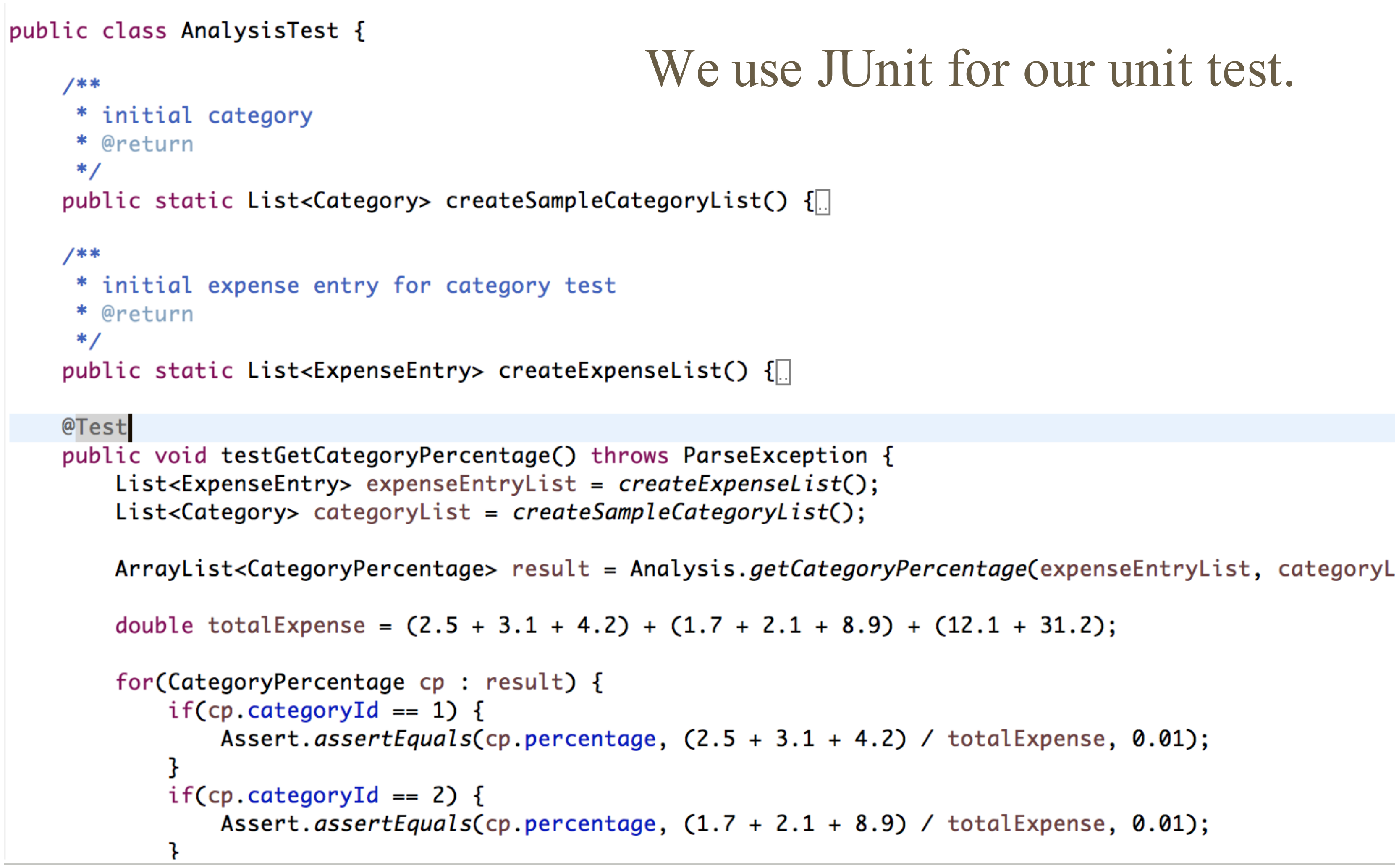
Backend Review - Core Business Logic



```
public class Analysis {

    /**
     * compute the spending percentage on each category from raw expense data
     * @param expenseEntryList
     * @param categoryList
     * @return
     * @throws ParseException
     */
    public static ArrayList<CategoryPercentage> getCategoryPercentage(List<ExpenseEntry> expens

    /**
     * TODO compute the spending on each day from raw expense data
     * @return
     */
    public static List<Pair<String, Double>> getExpenseDaily(List<ExpenseEntry> expenseEntryLis


    ..
```

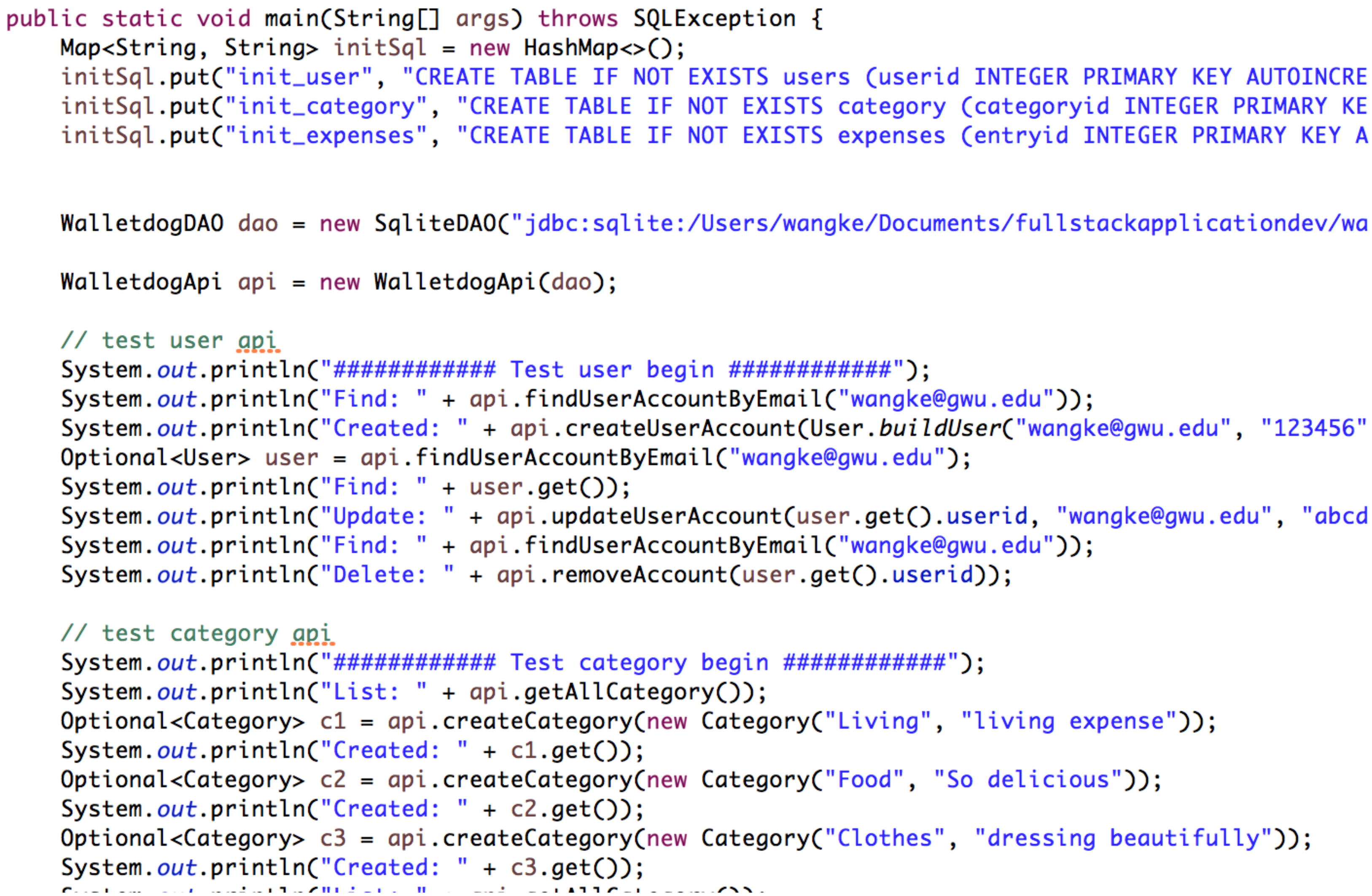
Backend Review - Unit Test

We use JUnit for our unit test.



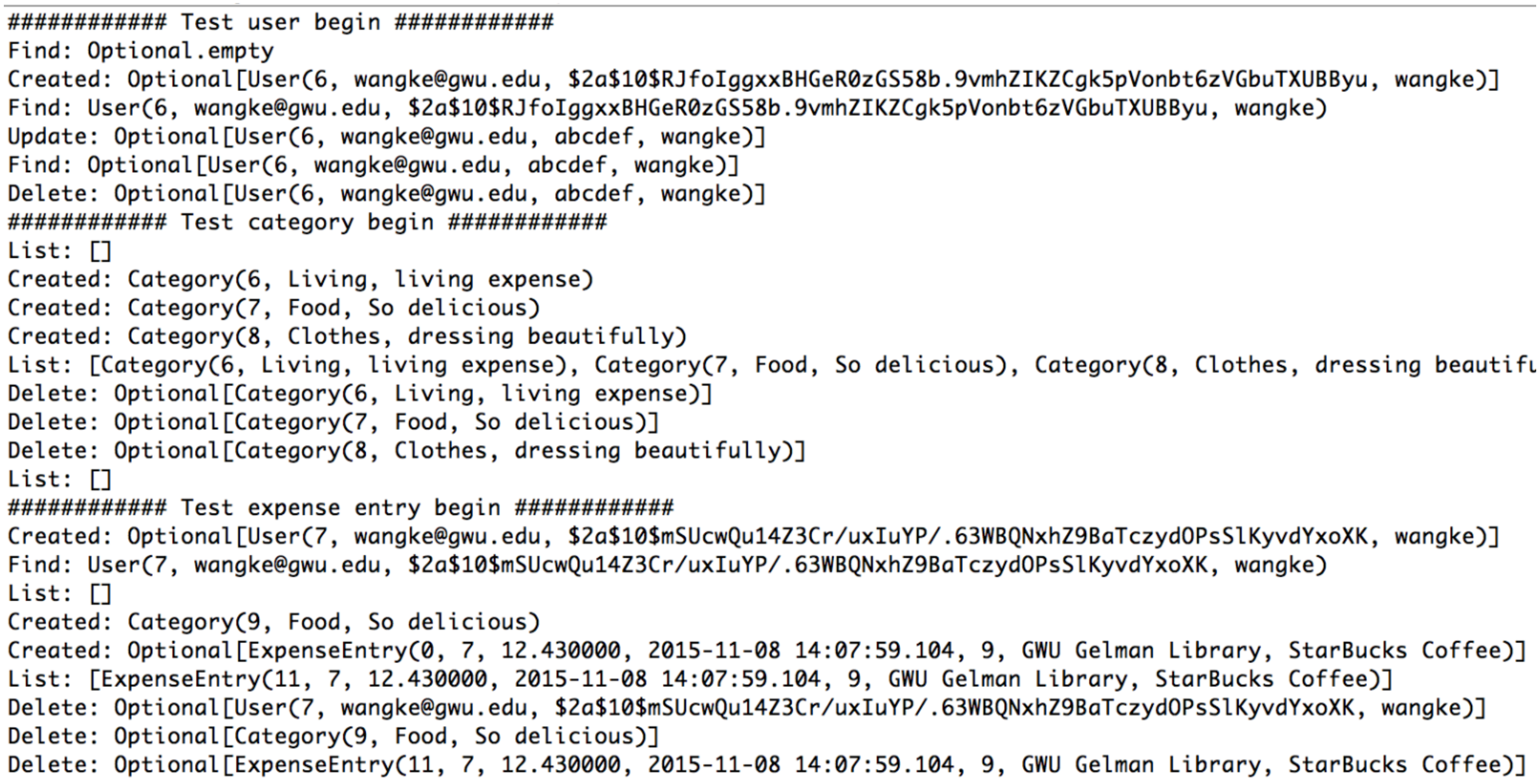
```
public class AnalysisTest {

    /**
     * initial category
     * @return
     */
    public static List<Category> createSampleCategoryList() {

    /**
     * initial expense entry for category test
     * @return
     */
    public static List<ExpenseEntry> createExpenseList() {

    @Test
    public void testGetCategoryPercentage() throws ParseException {
        List<ExpenseEntry> expenseEntryList = createExpenseList();
        List<Category> categoryList = createSampleCategoryList();

        ArrayList<CategoryPercentage> result = Analysis.getCategoryPercentage(expenseEntryList, categoryL

        double totalExpense = (2.5 + 3.1 + 4.2) + (1.7 + 2.1 + 8.9) + (12.1 + 31.2);

        for(CategoryPercentage cp : result) {
            if(cp.categoryId == 1) {
                Assert.assertEquals(cp.percentage, (2.5 + 3.1 + 4.2) / totalExpense, 0.01);
            }
            if(cp.categoryId == 2) {
                Assert.assertEquals(cp.percentage, (1.7 + 2.1 + 8.9) / totalExpense, 0.01);
            }
```

Backend Review - Integrated Test



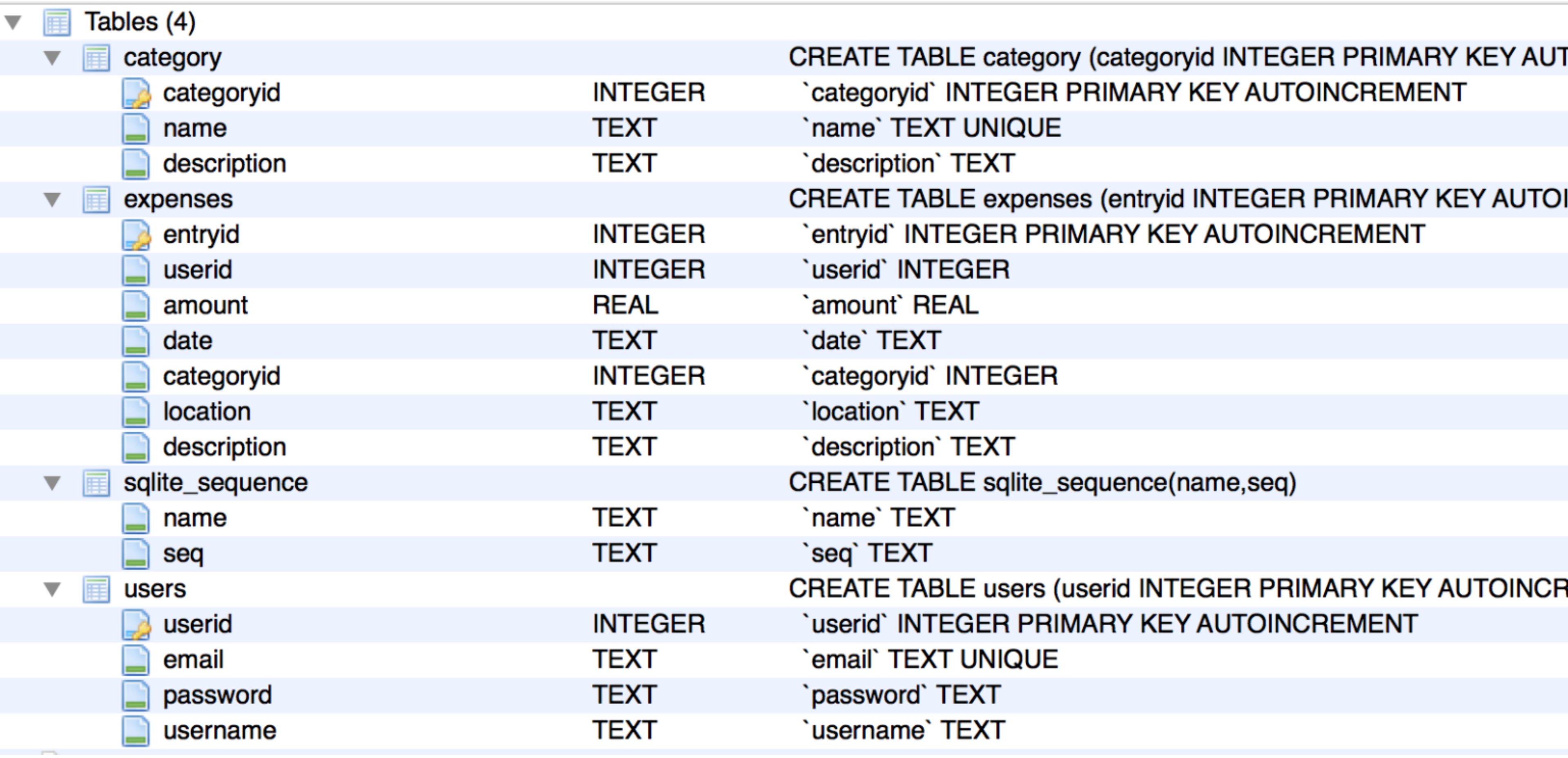
```
public static void main(String[] args) throws SQLException {
    Map<String, String> initSql = new HashMap<>();
    initSql.put("init_user", "CREATE TABLE IF NOT EXISTS users (userid INTEGER PRIMARY KEY AUTOINCRE
    initSql.put("init_category", "CREATE TABLE IF NOT EXISTS category (categoryid INTEGER PRIMARY KE
    initSql.put("init_expenses", "CREATE TABLE IF NOT EXISTS expenses (entryid INTEGER PRIMARY KEY A


    WalletdogDAO dao = new SqliteDAO("jdbc:sqlite:/Users/wangke/Documents/fullstackapplicationdev/wa

    WalletdogApi api = new WalletdogApi(dao);

    // test user api
    System.out.println("############ Test user begin ############");
    System.out.println("Find: " + api.findUserAccountByEmail("wangke@gwu.edu"));
    System.out.println("Created: " + api.createUserAccount(User.buildUser("wangke@gwu.edu", "123456"
    Optional<User> user = api.findUserAccountByEmail("wangke@gwu.edu");
    System.out.println("Find: " + user.get());
    System.out.println("Update: " + api.updateUserAccount(user.get().userid, "wangke@gwu.edu", "abcd
    System.out.println("Find: " + api.findUserAccountByEmail("wangke@gwu.edu"));
    System.out.println("Delete: " + api.removeAccount(user.get().userid));

    // test category api
    System.out.println("############ Test category begin ############");
    System.out.println("List: " + api.getAllCategory());
    Optional<Category> c1 = api.createCategory(new Category("Living", "living expense"));
    System.out.println("Created: " + c1.get());
    Optional<Category> c2 = api.createCategory(new Category("Food", "So delicious"));
    System.out.println("Created: " + c2.get());
    Optional<Category> c3 = api.createCategory(new Category("Clothes", "dressing beautifully"));
    System.out.println("Created: " + c3.get());
    System.out.println("List: " + api.getAllCategory());
```

Backend Review - Integrated Test Result



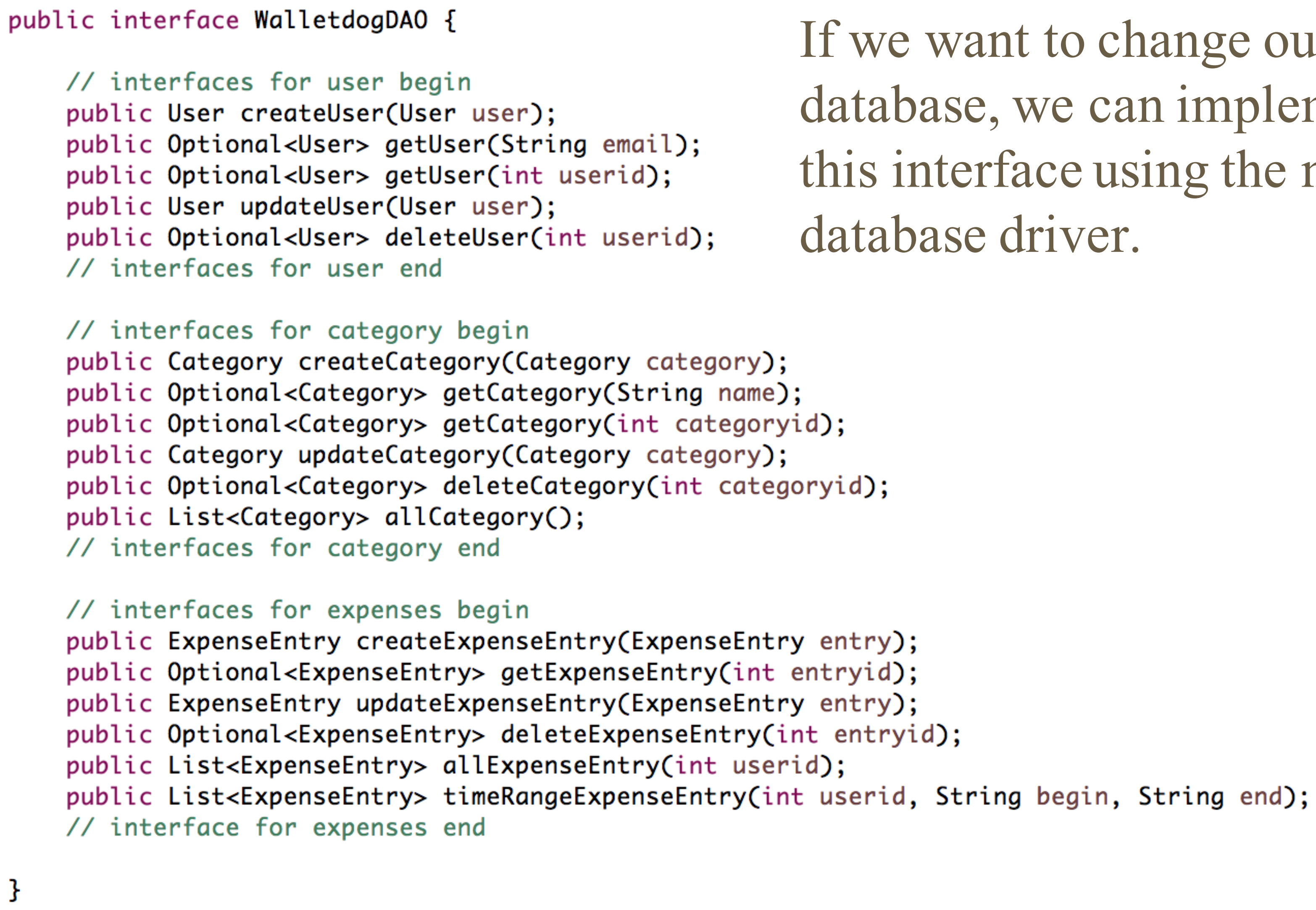
```
############ Test user begin ############
Find: Optional.empty
Created: Optional[User(6, wangke@gwu.edu, $2a$10$RJfoIggxxBHGeR0zGS58b.9vmhZIKZCgk5pVonbt6zVGbuTXUBByu, wangke)]
Find: User(6, wangke@gwu.edu, $2a$10$RJfoIggxxBHGeR0zGS58b.9vmhZIKZCgk5pVonbt6zVGbuTXUBByu, wangke)
Update: Optional[User(6, wangke@gwu.edu, abcdef, wangke)]
Find: Optional[User(6, wangke@gwu.edu, abcdef, wangke)]
Delete: Optional[User(6, wangke@gwu.edu, abcdef, wangke)]
############ Test category begin ############
List: []
Created: Category(6, Living, living expense)
Created: Category(7, Food, So delicious)
Created: Category(8, Clothes, dressing beautifully)
List: [Category(6, Living, living expense), Category(7, Food, So delicious), Category(8, Clothes, dressing beautifu
Delete: Optional[Category(6, Living, living expense)]
Delete: Optional[Category(7, Food, So delicious)]
Delete: Optional[Category(8, Clothes, dressing beautifully)]
List: []
############ Test expense entry begin ############
Created: Optional[User(7, wangke@gwu.edu, $2a$10$mSUcwQu14Z3Cr/uxIuYP/.63WBQNxhZ9BaTczydOPsSlKyvdYxoXK, wangke)]
Find: User(7, wangke@gwu.edu, $2a$10$mSUcwQu14Z3Cr/uxIuYP/.63WBQNxhZ9BaTczydOPsSlKyvdYxoXK, wangke)
List: []
Created: Category(9, Food, So delicious)
Created: Optional[ExpenseEntry(0, 7, 12.430000, 2015-11-08 14:07:59.104, 9, GWU Gelman Library, StarBucks Coffee)]
List: [ExpenseEntry(11, 7, 12.430000, 2015-11-08 14:07:59.104, 9, GWU Gelman Library, StarBucks Coffee)]
Delete: Optional[User(7, wangke@gwu.edu, $2a$10$mSUcwQu14Z3Cr/uxIuYP/.63WBQNxhZ9BaTczydOPsSlKyvdYxoXK, wangke)]
Delete: Optional[Category(9, Food, So delicious)]
Delete: Optional[ExpenseEntry(11, 7, 12.430000, 2015-11-08 14:07:59.104, 9, GWU Gelman Library, StarBucks Coffee)]
```

Backend Review - Database Structure



Tables (4)		
category		CREATE TABLE category (categoryid INTEGER PRIMARY KEY AUT
categoryid	INTEGER	\`categoryid\` INTEGER PRIMARY KEY AUTOINCREMENT
name	TEXT	\`name\` TEXT UNIQUE
description	TEXT	\`description\` TEXT
expenses		CREATE TABLE expenses (entryid INTEGER PRIMARY KEY AUTOI
entryid	INTEGER	\`entryid\` INTEGER PRIMARY KEY AUTOINCREMENT
userid	INTEGER	\`userid\` INTEGER
amount	REAL	\`amount\` REAL
date	TEXT	\`date\` TEXT
categoryid	INTEGER	\`categoryid\` INTEGER
location	TEXT	\`location\` TEXT
description	TEXT	\`description\` TEXT
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
name	TEXT	\`name\` TEXT
seq	TEXT	\`seq\` TEXT
users		CREATE TABLE users (userid INTEGER PRIMARY KEY AUTOINCR
userid	INTEGER	\`userid\` INTEGER PRIMARY KEY AUTOINCREMENT
email	TEXT	\`email\` TEXT UNIQUE
password	TEXT	\`password\` TEXT
username	TEXT	\`username\` TEXT

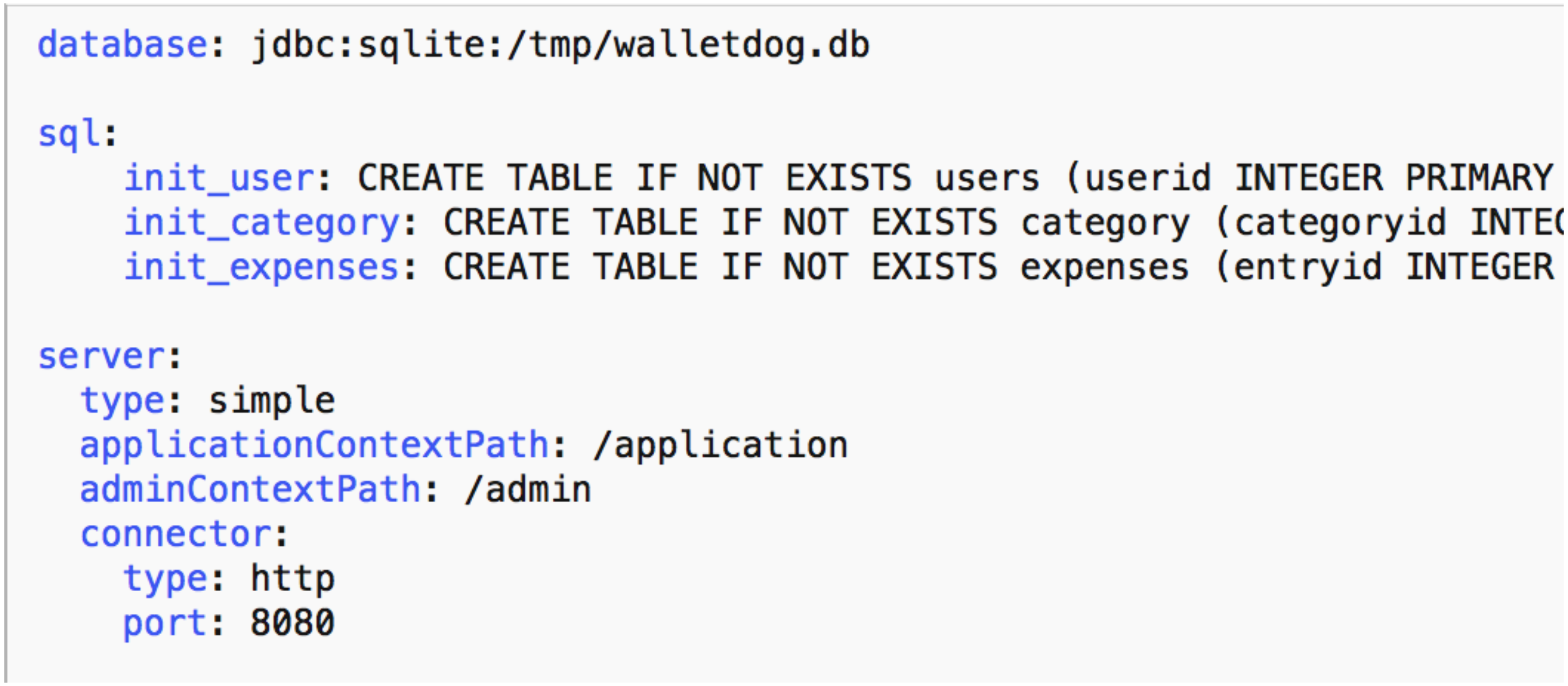
Backend Review - Database Interface



```java
public interface WalletdogDAO {

    // interfaces for user begin
    public User createUser(User user);
    public Optional<User> getUser(String email);
    public Optional<User> getUser(int userid);
    public User updateUser(User user);
    public Optional<User> deleteUser(int userid);
    // interfaces for user end

    // interfaces for category begin
    public Category createCategory(Category category);
    public Optional<Category> getCategory(String name);
    public Optional<Category> getCategory(int categoryid);
    public Category updateCategory(Category category);
    public Optional<Category> deleteCategory(int categoryid);
    public List<Category> allCategory();
    // interfaces for category end

    // interfaces for expenses begin
    public ExpenseEntry createExpenseEntry(ExpenseEntry entry);
    public Optional<ExpenseEntry> getExpenseEntry(int entryid);
    public ExpenseEntry updateExpenseEntry(ExpenseEntry entry);
    public Optional<ExpenseEntry> deleteExpenseEntry(int entryid);
    public List<ExpenseEntry> allExpenseEntry(int userid);
    public List<ExpenseEntry> timeRangeExpenseEntry(int userid, String begin, String end);
    // interface for expenses end

}
```

If we want to change our database, we can implement this interface using the new database driver.

Backend Review - Dropwizard Configuration



```yaml
database: jdbc:sqlite:/tmp/walletdog.db

sql:
    init_user: CREATE TABLE IF NOT EXISTS users (userid INTEGER PRIMARY
    init_category: CREATE TABLE IF NOT EXISTS category (categoryid INTE
    init_expenses: CREATE TABLE IF NOT EXISTS expenses (entryid INTEGER

server:
  type: simple
  applicationContextPath: /application
  adminContextPath: /admin
  connector:
    type: http
    port: 8080
```

Frontend Review - overview

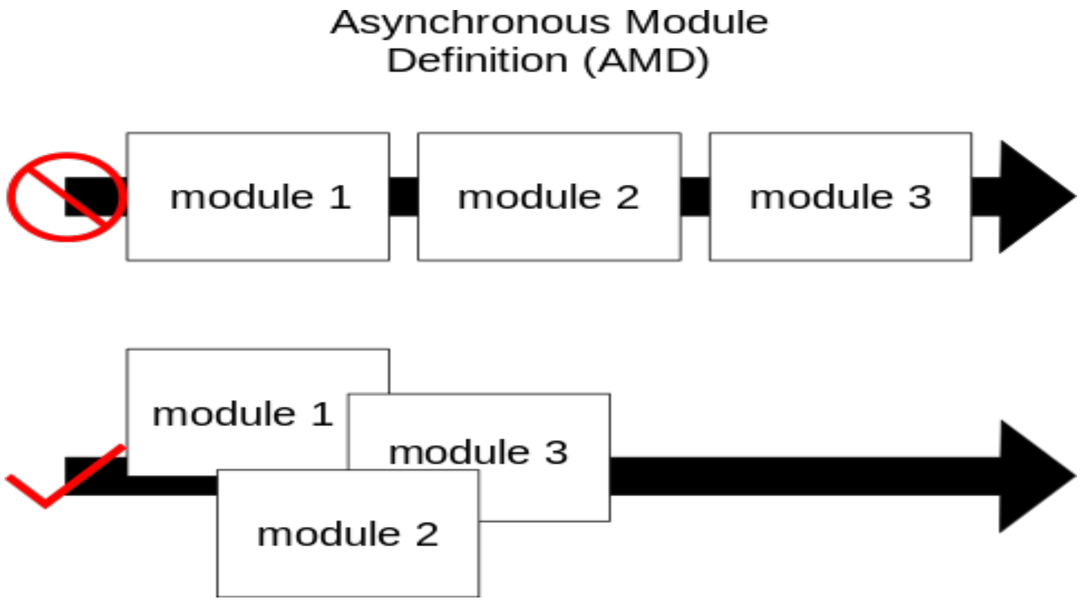
Architecture: RequireJS AMD (Asynchronous Module Definition)

Language: Javascript, AngularJs

Theme: Angular-Material, Bootstrap, ionicons

Architecture

The Asynchronous Module Definition (**AMD**) API specifies a mechanism for defining modules such that the module and its dependencies can be asynchronously loaded. This is particularly well suited for the browser environment where synchronous loading of modules incurs performance, usability, debugging, and cross-domain access problems.

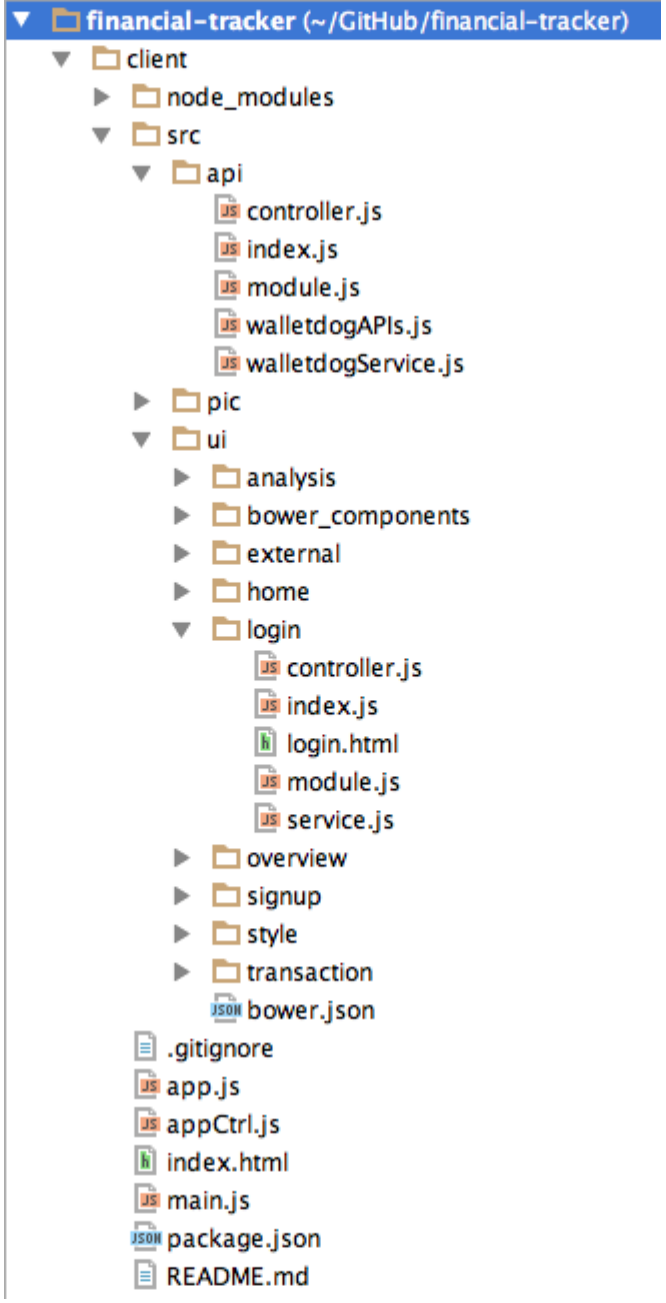


Frontend Review

We separate our frontend project into two parts :

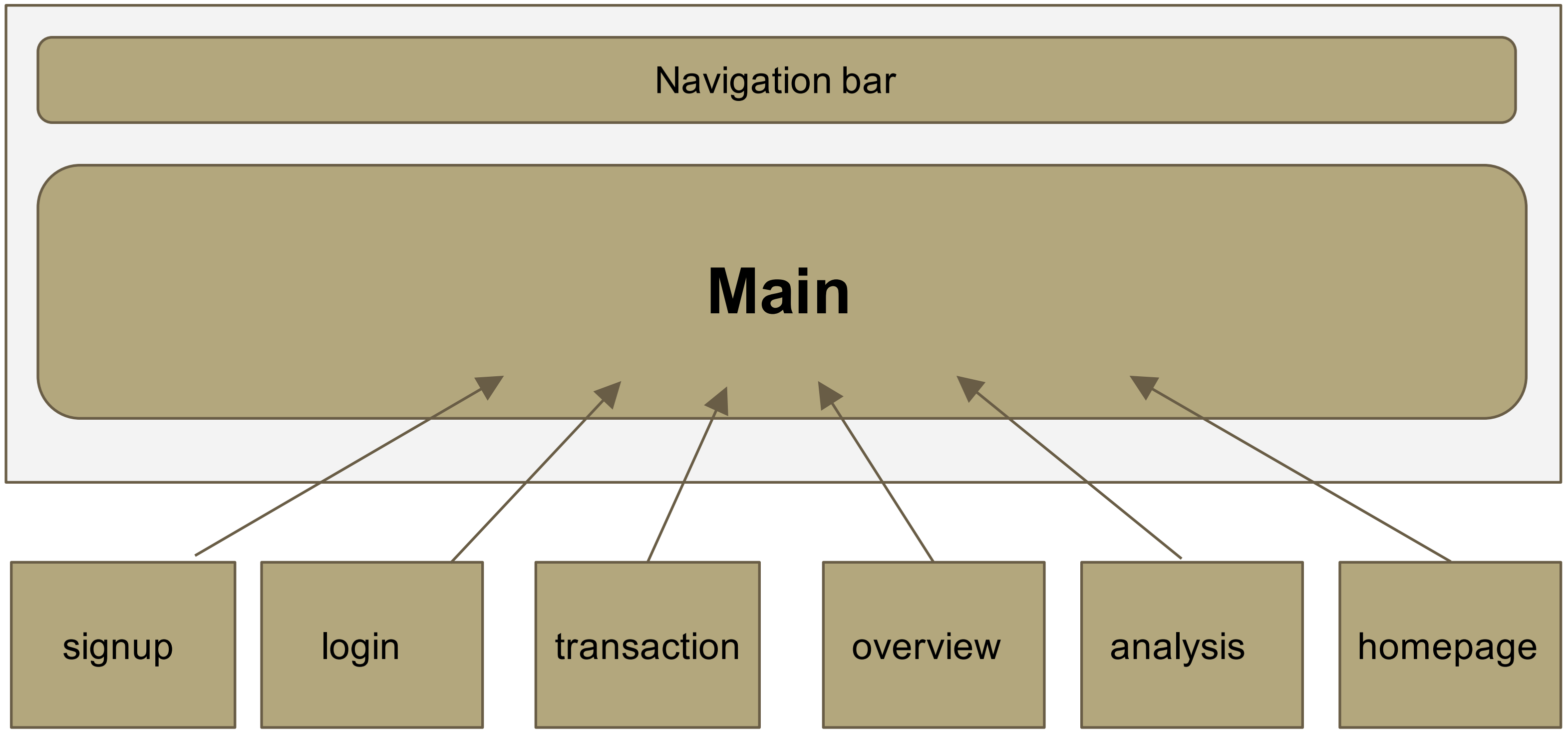
Web API

UI Modules



```
▼ financial-tracker (~/GitHub/financial-tracker)
  ▼ client
    ▶ node_modules
    ▼ src
      ▼ api
          controller.js
          index.js
          module.js
          walletdogAPIs.js
          walletdogService.js
      ▶ pic
      ▼ ui
        ▶ analysis
        ▶ bower_components
        ▶ external
        ▶ home
        ▼ login
            controller.js
            index.js
            login.html
            module.js
            service.js
        ▶ overview
        ▶ signup
        ▶ style
        ▶ transaction
          bower.json
      .gitignore
      app.js
      appCtrl.js
      index.html
      main.js
      package.json
      README.md
```

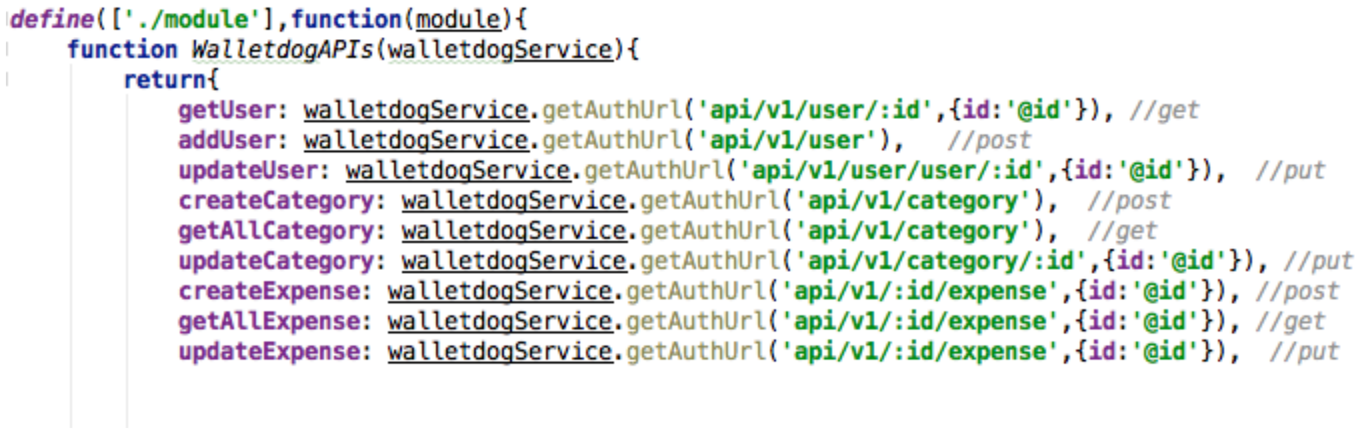
Frontend Review



Frontend Review

Web API - walletdogAPIs

use angularjs - $resource(walletdogService) send post/get/put to send payload to, get or update data from backend.

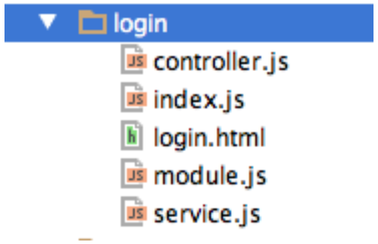


```
define(['./module'],function(module){
    function WalletdogAPIs(walletdogService){
        return{
            getUser: walletdogService.getAuthUrl('api/v1/user/:id',{id:'@id'}), //get
            addUser: walletdogService.getAuthUrl('api/v1/user'),    //post
            updateUser: walletdogService.getAuthUrl('api/v1/user/user/:id',{id:'@id'}),  //put
            createCategory: walletdogService.getAuthUrl('api/v1/category'),  //post
            getAllCategory: walletdogService.getAuthUrl('api/v1/category'),  //get
            updateCategory: walletdogService.getAuthUrl('api/v1/category/:id',{id:'@id'}), //put
            createExpense: walletdogService.getAuthUrl('api/v1/:id/expense',{id:'@id'}), //post
            getAllExpense: walletdogService.getAuthUrl('api/v1/:id/expense',{id:'@id'}), //get
            updateExpense: walletdogService.getAuthUrl('api/v1/:id/expense',{id:'@id'}),  //put
```

Frontend Review

UI Module

```
|- main.js   : load all dependencies

|- app.js    : combine all modules

  |-- index.js                                 |-- index.js

  |-- module.js                                |-- module.js
                          all modules
  |-- controller.js       .........            |-- controller.js

  |-- service.js                               |-- service.js

  |-- html                                     |-- html
```



```
▼ login
    controller.js
    index.js
    login.html
    module.js
    service.js
```

Reference

1. Dropwizard user manual

http://www.dropwizard.io/0.9.1/docs/manual/index.html

2. SQLite documents

https://www.sqlite.org/docs.html

3. Img1 Save Spend Invest Dice, comes from FreeDigitalPhotos.net:

http://www.freedigitalphotos.net/images/Finance_g198-Save_Spend_Invest_Dice_p69210.html

Demo

Click Here