Lab 4 and Beyond

First Steps...

Story 1 - Fetching data from an endpoint

As the branch manager

I Want to see the accounts provided by the banking API

So That I can help the clients when they arrive in the branch

Now your bank needs to fetch account data from an API

You can find the API Swagger at http://api.asep-strath.co.uk/swagger

The specific endpoint you should be using for this story is:

`GET http://api.asep-strath.co.uk/api/{bank}/accounts`

where {bank} is replaced by your team name e.g. “Team1”

UniRest Library

The UniRest Library has been provided to perform HTTP requests when accessing other APIs. You have also been provided a custom Jooby module which sets up automatic serialisation for the library, which will automatically parse JSON into a POJO (Plain Old Java Object).

```
HttpResponse<Book> bookResponse =
    Unirest.get("http://httpbin.org/books/1").asObject(Book.class);
Book bookObject = bookResponse.getBody();
```

Comprehensive documentation can be found: http://unirest.io/java.html

Story 2 - Storing data in a database

As the branch manager

I Want to be able to see the accounts if the banking API is not accessible

So That I can help the clients manage their accounts when there are network issues in the branch

Opening a connection

The Jooby framework has an already pre-configured a H2 in memory database. To use it you will need to fetch the datasource and open a connection to the database

```
DataSource db = require(DataSource.class);
Connection connection = cb.createConnection();
```

Once you have finished with the connection you need to close it to free up resources.

```
connection.close();
```

Create a table

Before entering data you will need to create a table. You will have to design the table appropriately to store the data you require.

```
Statement stmt = c.createStatement();
String sql = "CREATE TABLE IF NOT EXISTS employees (\n"
            + " id integer PRIMARY KEY,\n"
            + " name text NOT NULL,\n"
            + " ohone integer NOT NULL,\n"
            + " salary decimal NOT NULL,\n"
            + " dob date NOT NULL,\n"
            + " photo text);";
stmt.execute(sql);
```

Inserting Data

The connection can be used to execute SQL queries. These can either insert data or retrieve it.

```
String sql = "INSERT INTO employees (name, phone, salary, dob, photo) "
       + "VALUES (?,?,?,?,?)"; // Note: the ?s are important
PreparedStatement prep = c.prepareStatement(sql);
prep.setString(1, "Bob");
prep.setInt(2, 666666666);
prep.setDouble(3, 35000.00);
prep.setDate(4, anSqlDateObject);
prep.setBytes(5, aBytesArray);
prep.executeUpdate();
```

Retrieving Data

```
Statement stmt = c.createStatement();
String sql = "SELECT * FROM employees";
ResultSet rs = stmt.executeQuery(sql);
```

Retreive with Query

```
String sql = "SELECT * FROM employees WHERE name LIKE ?";
PreparedStatement prep = c.prepareStatement(sql);
prep.setString(1, "%Bob%");
ResultSet rs = prep.executeQuery();
```

Once you have a result set

```
while (rs.next()) {
    int id = rs.getInt("id");
    String name = rs.getString("name");
    int age = rs.getInt("phone");
    String address = rs.getString("address");
    double salary = rs.getDouble("salary");
    java.sql.Date date = rs.getDate("dob");
    Employee employee = new Employee(id, name, age, address, salary);
}
rs.close();
```

Feature Backlog

Transaction Information

As a bank manager

I want to be able to view Transaction information on bank accounts

So that I can advice customers based on their transactions

Transaction information must include at least:

- Account Balance Before - Initial Account Balance from /accounts
- Account Balance After - Account Balance after transactions have processed
- Number of Transactions Processed Total
- Number of Transactions Processed Per Account
- Number of Transactions Failed Per Account

Additional Details

A transaction is any occurrence of money being added or removed from an account. For example £20 moved from account #123A into account #124B.

If a transaction is attempted where there are insufficient funds in the source account, then it should fail and the recipient account should not receive funds.

You can find the documentation for the Transaction API : http://api.asep-strath.co.uk/swagger

User Interface

As a bank manager

I Want the UI of the system to be fluid, attractive and intuitive

So that interacting with the system is not needlessly cumbersome or unpleasant.

Colours and fonts should be in alignment with brand guidelines.

Handle Fraudulent Transactions

As a bank manager

I want the system to be abe to spot and stop fraudulent claims

So that there is no risk of fraudulent activity on accounts to comply with regulations

You can find the documentation for the Fraud API : http://api.asep-strath.co.uk/swagger

Pagination of tables

As a bank manager

I Want to view all information tables in pages without having to scroll when i see more than 20 entries

So That I can view information in small manageable chunks

Account Search

As a bank manager

I Want to be able to search for an account by name

So That I can quickly retrieve account and transaction info for a particular number

Mobile application implementation

As the mobile development team lead

I want Swagger API documentation

So that it can be used in the development of a mobile application

Transaction Reversal

As a bank manager

I want to be able to reverse a transaction which has already taken place on an account

So that I can fix any accidental transactions

Note: If the transaction is to/from another bank you need to notify them via the Transaction Reveral API

You can find the documentation for the Reversal Transaction API : http://api.asep-strath.co.uk/swagger

Transaction Repeat

As a bank manager

I want to be able to repeat a transaction which has already taken place on an account

So that I can save time by repeating a regular transaction which isn't a Direct Debit

Statement Export

As a bank manager

I want to be able to export the current page of account information to Excel

So that I can formulate reporting on bank performance to shareholders

Marking

Lab 6 and Lab 8

Team Retro (1 mark)

- Each team will run it's own retro covering their experience from the previous week of development.
- Someone should play the role of master of ceremonies
- Should try different styles of retro

Agile Delivery (3 marks)

- Are you working in an agile fashion?
- Do you have an up to date issue board?
- Are tasks being completed effectively?

Engineering Excellence (3 marks)

- Unit Tests, Do you have them? Are they providing adequate coverage?
- Code Review, walkthrough the code with one of the instructors, discussing solutions, issues and reasons for particular implementations.
- Branches and Commit History, are you using Git? Are you using branches, and do each of them link to an issue? Are you constantly committing and pushing?
- Application Versioning and CI, Are you continuously building, testing and packaging your application from git?

Feature Demo (3 marks)

- Demo of latest features to instructor.
- Demos must be ran from a packaged jar file. How to do this was explained in Lab 2

Final Presentation (+ 10 marks)

- Final product owner demo.