

# Software Engineering 2 SUPD REPORT

## Status Update Report

<b>Team number:</b>	33
---------------------	----

Team member 1	
<b>Name:</b>	Vincent Zettl
<b>Student ID:</b>	01404131
<b>E-mail address:</b>	a01404131@unet.univie.ac.at

Team member 2	
<b>Name:</b>	Lukas Höwarth
<b>Student ID:</b>	01447910
<b>E-mail address:</b>	a01447910@unet.univie.ac.at

Team member 3	
<b>Name:</b>	Alexander Garber
<b>Student ID:</b>	01568645
<b>E-mail address:</b>	a01568645@unet.univie.ac.at

Team member 4	
<b>Name:</b>	TODØ
<b>Student ID:</b>	TODØ
<b>E-mail address:</b>	TODØ

# 1 Design Draft

## 1.1 Design Approach and Overview

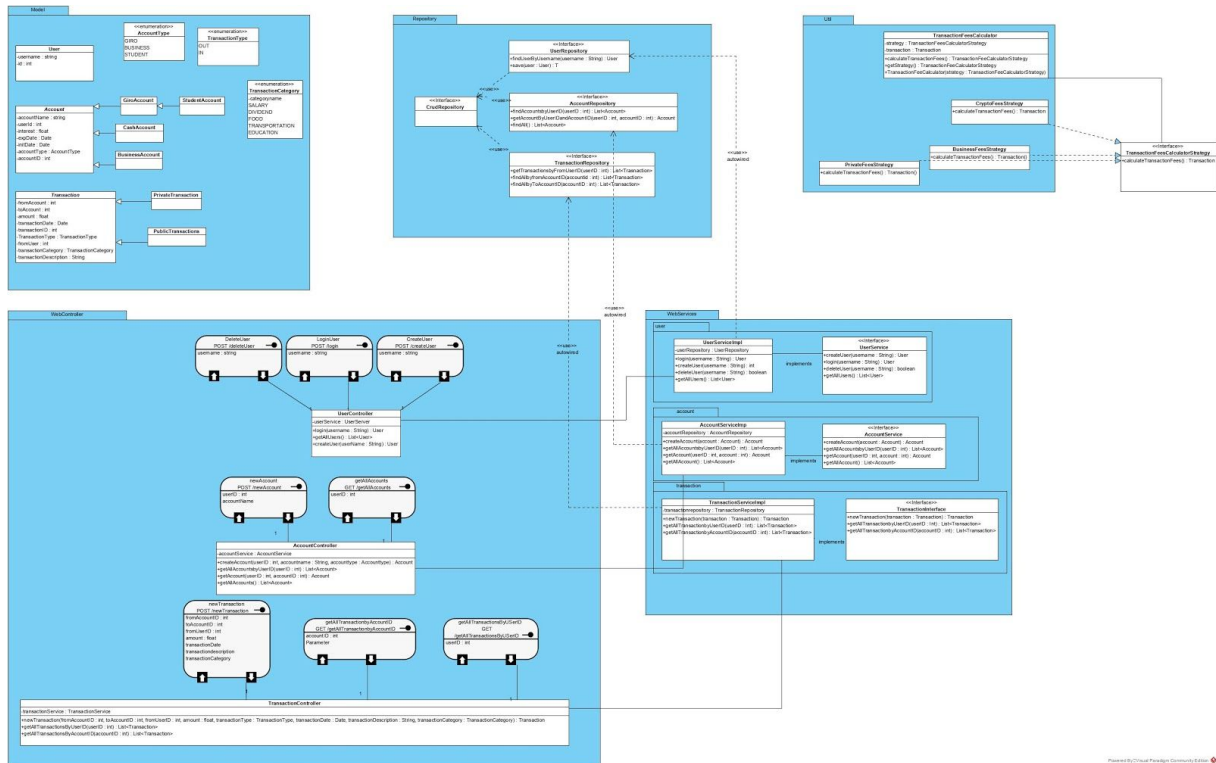
To create a simple prototype and get a visual representation of the basic features of our project we created a web based prototype using w3.ss. With this we layed out a first draft of the different classes in use. After getting confirmation, that it was okay, we decided to build our project using Rest with Spring Boot, which lead to us further improving the web based prototype and using it for a foundation of our implementation. We decided to create a form to decide which Framework we would use for the frontend which resulted in using Vanilla JavaScript.

For the database we had multiple options in mind. In the end H2 came out on top (reasoning see technology stack)

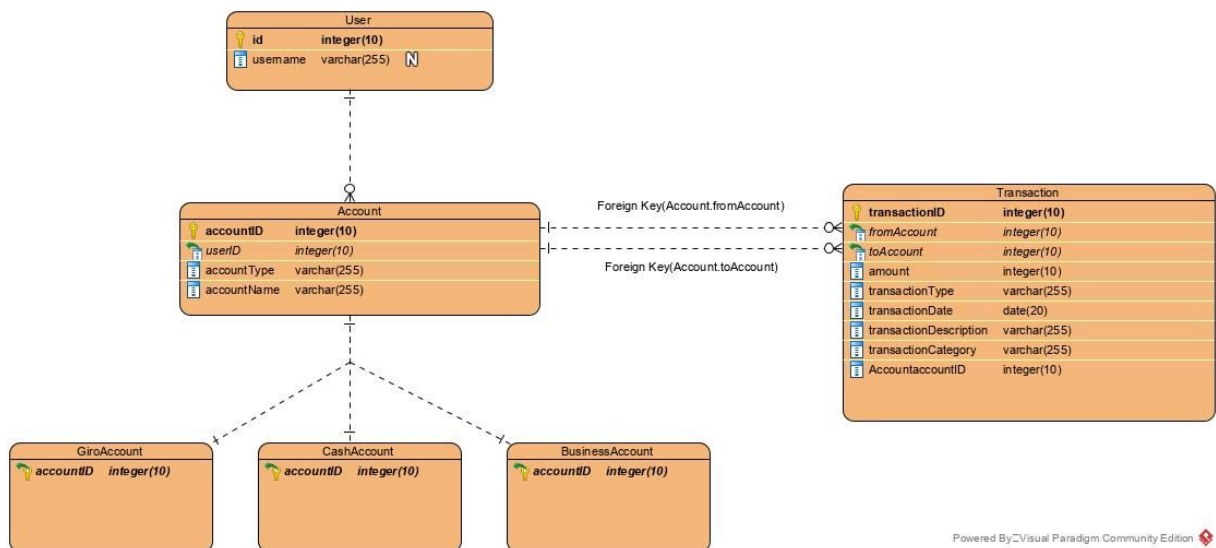
With regards to the assignment we created multiple diagram sketches, on paper, for the required functionalities of the classes and their associations with each other.

The Rest of the project process is shown on the following pages.

## 1.1.1 Class Diagrams



## Entity Relation Ship - Model



## 1.1.2 Technology Stack

As suggested for this course we use Java for our Project. We chose Java 11 because it's the latest version with Long Term Support. Because of that the documentation should already be solid.

<https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html>

We are building a web based application using Rest(Spring Boot). This decision was made for multiple reasons. First and foremost all of our team members have experience with Rest. Nonetheless this decision wasn't merely made because of our prior knowledge but because of the advantages Rest brings to the table. It offers a uniform interface on top of being modular as a result of the separated Client and Server. This streamlines the process of implementing new features.

<https://spring.io/>

For version control, as suggested, Gitlab is used. It gives a good overview of who did what and makes it easy to revert back to an older version if bigger mistakes are made and the projects needs to be restructured from a certain point. Furthermore the Issues are a nice addition for coordinating tasks among the team members.

<https://lab.swa.univie.ac.at/>

Our building tool of choice is Maven. The reason for this is the use of XML for the configuration file to build the project. To run into less trouble and because of our prior experience from two of our team members with Maven we chose it over Gradle. We used the latest version 3.6.1 when we started our project and will stick with it unless there are major bugs fixed with newer versions.

<https://maven.apache.org/>

For the database we chose H2. We made this decision due to several reasons. Firstly because it's a SQL database which offers a nice transition from the created diagrams to the relational database, secondly it brings a browser based console application to the table thus making debugging and checking the database entries on the fly an easier process. Thirdly and most important for our project it is an in-memory database, meaning it stores the data locally ergo it's ideal for the requirements of the given application. Finally it has a really small footprint and therefore keeps our project from becoming too bloated as it only ways in with a 2MB jar file.

<https://www.h2database.com/html/main.html>

We discussed different IDEs to use. Finally we chose IntelliJ over the rest. All of us worked with IntelliJ and Eclipse for Java development prior to this project. Reasons why IntelliJ made the cut are that it is easy and straightforward to initialize and refactor the project. Code inspection and useful shortcuts make this IDE a blast to use. Because of GIT integration it's made easy to keep all the work up to date. In addition the code analysis and debugging tools help keep the implementation clean and hopefully keep bugs to a minimum.

<https://www.jetbrains.com/idea/>

During the first few hours of implementation we had to decide on how to keep our code uniformly. We decided to use the Google Java Style Guide as it seemed to be one of the, if not even the most commonly used style out there.

<https://www.h2database.com/html/main.html>

For the creation of the diagrams Visual Paradigm Community Edition is used. It has a wide variety of different diagram options to choose from. Especially the possibility to have a template for modeling Rest functionalities is a nice inclusion. In prior courses we had to work with BeeUp which didn't make the cut. On the one side Visual Paradigm offers better stability, as a result of that it is less likely to lose our progress. On the other hand it offers the possibility for collaboration which, for a group project, plays right into our hands.

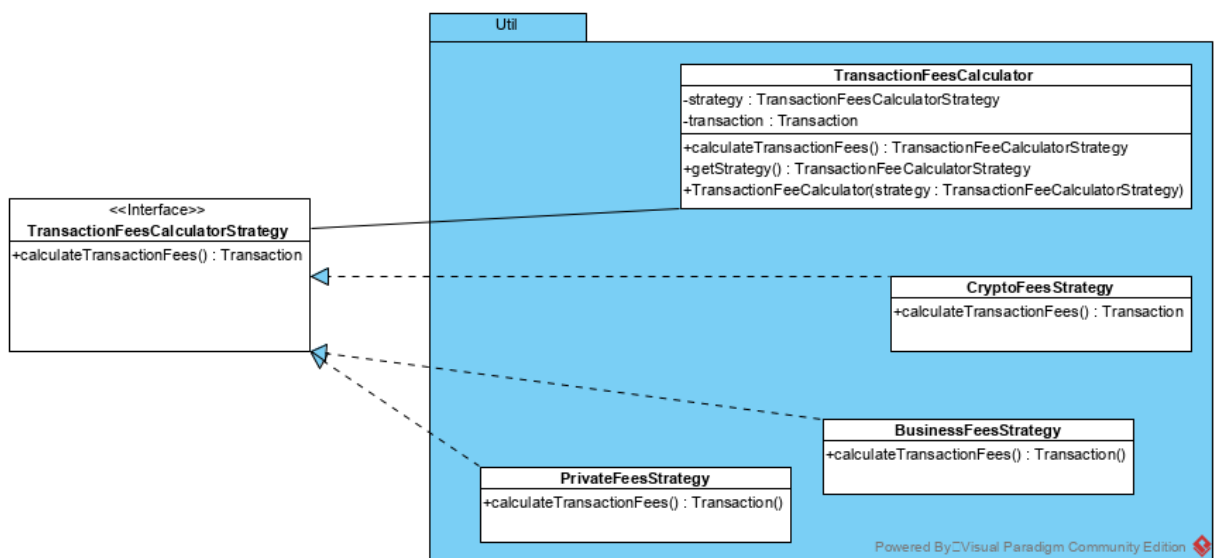
<https://www.visual-paradigm.com/>.

## 1.2 Design Patterns

### 1.2.1 Strategy Pattern

The strategy pattern makes it possible to achieve what's called “encapsulate what varies” in object oriented programming. It helps keeping the code modular. If for example we want to add another Strategy to calculate the fees differently without this pattern we would have every of those calculation algorithms in one file and one class. This violates the principle that new functionality should be added by writing new code instead of changing existing.

Using the strategy pattern it's now possible to add a new class that implements the interface, write one new algorithm to calculate the fees and we are set.



## Code Snippets:

### Strategy Interface

```
public interface TransactionFeeCalculatorStrategy {
    Transaction calculateTransactionFees(Transaction transaction);
}
```

### Context

```
public class TransactionFeeCalculator {
    private TransactionFeeCalculatorStrategy strategy;
    private Transaction transaction;

    public TransactionFeeCalculator(TransactionFeeCalculatorStrategy strategy) {
        this.strategy = strategy;
    }

    public Transaction calculateTransactionFees() {
        return strategy.calculateTransactionFees(transaction);
    }
}
```

```
public TransactionFeeCalculatorStrategy getStrategy() {
    return strategy;
}

public void setStrategy(TransactionFeeCalculatorStrategy strategy) {
    this.strategy = strategy;
}

public Transaction getTransaction() {
    return transaction;
}

public void setTransaction(Transaction transaction) {
    this.transaction = transaction;
}
}
```

### Concrete Strategy

```
public class CryptoFeesStrategy implements TransactionFeeCalculatorStrategy {
    @Override
    public Transaction calculateTransactionFees(Transaction transaction) {
        System.out.println("Using CryptoFeesStrategy Calculation Method");

        float fees = 0.0025f;

        transaction.setAmount(transaction.getAmount() + transaction.getAmount() * fees);

        return transaction;
    }
}
```

### Concrete Strategy

```
public class BusinessFeesStrategy implements TransactionFeeCalculatorStrategy {
    @Override
    public Transaction calculateTransactionFees(Transaction transaction) {
        System.out.println("Using BusinessFeesStrategy Calculation Method");
        float fees = 0.015f;

        transaction.setAmount(transaction.getAmount() + transaction.getAmount() * fees);

        return transaction;
    }
}
```

### Concrete Strategy

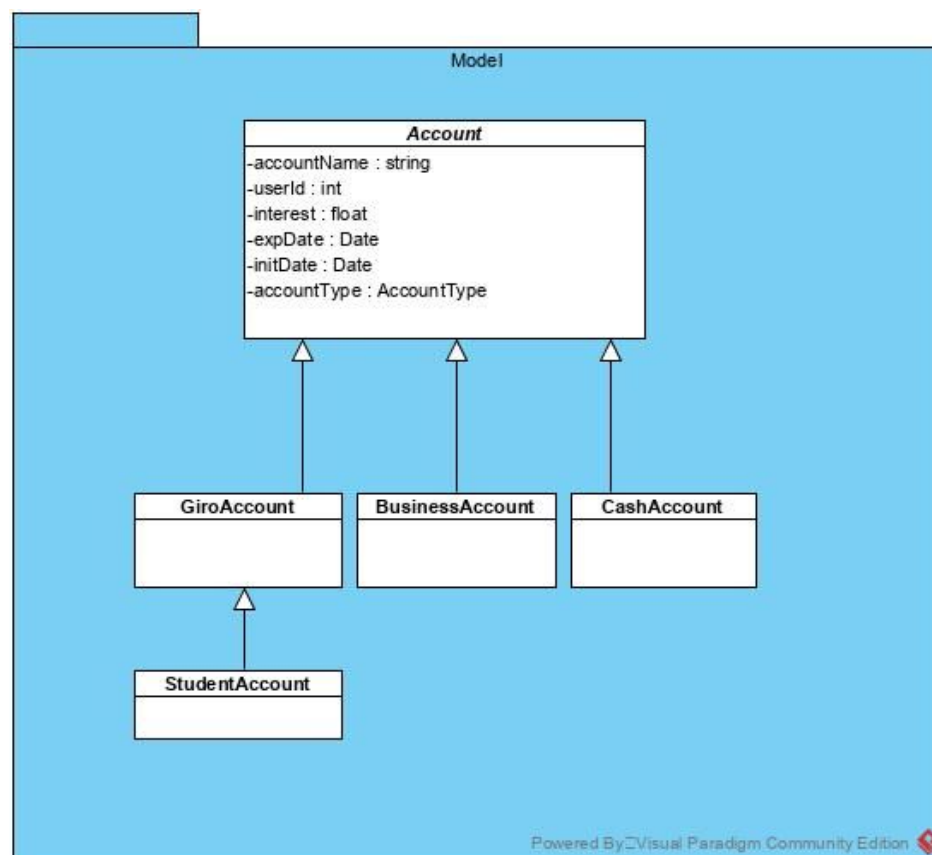
```
public class PrivateFeesStrategy implements TransactionFeeCalculatorStrategy {
    @Override
    public Transaction calculateTransactionFees(Transaction transaction) {
        System.out.println("Using noFees Calculation Method");
        return transaction;
    }
}
```

## 1.2.2 Composite Pattern

The composite pattern makes it possible to treat multiple objects as the same by creating a class containing its own objects. It helps to hide the differences between the single or partly compounded objects. For example, a hierarchical file-system uses such a composite pattern. In Java there are already written GUI-Element classes which uses the composite pattern like Swing or AWT which uses the class "component" to ensure the usage of the composite pattern. In addition, Java is build up on the composite pattern, because all classes and instances such as the datatype-classes like String, Integer, Float, Double, Character and every additional develop classes extends automatically inherit from the class object.

<https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html>

For the project we used the composite pattern for the specific account encapsulation where GiroAccount, BusinessAccount and CashAccount extends from the abstract class Account.





## Code Snippets:

### Base component - Abstract class Account

```
@Entity
public abstract class Account {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    int accountID;

    @NotNull int userID;
    AccountType accountType;

    @Column(unique = true)
    String accountName;

    float balance;

    public Account() {}

    /**
     * on creating of new Account sets starting balance to 0
     * @param userID unique identifies user
     * @param accountName unique per user
     * @param accountType defines type of account
     */
    public Account(int userID, String accountName, AccountType accountType) {
        this.userID = userID;
        this.accountType = accountType;
        this.accountName = accountName;
        this.balance = 0.0f;
    }

    abstract float getOverdraftlimit();
    abstract void setOverdraftlimit(float overdraftlimit);
}
```

## Child component - GiroAccount

```
@Entity
public class GiroAccount extends Account {

    float overdraftlimit;

    public GiroAccount() {
        super();
    };

    /**
     * has overdraftlimit as defining feature, if not set, set to 0
     *
     * @param userID unique identifies user
     * @param accountName unique per user
     * @param accountType defines type of account
     * @param overdraftlimit sets overdraftlimit to certain amount
     */
    public GiroAccount(
        int userID, String accountName, AccountType accountType, float overdraftlimit) {
        super(userID, accountName, accountType);
        this.overdraftlimit = overdraftlimit;
    };

    /**
     * if overdraftlimit not set, set to 0
     *
     * @param userID unique identifies user
     * @param accountName unique per user
     * @param accountType defines type of account
     */
    public GiroAccount(int userID, String accountName, AccountType accountType) {
        super(userID, accountName, accountType);
        this.overdraftlimit = 0.0f;
    };

    @Override
    public float getOverdraftlimit() {
        return overdraftlimit;
    }

    @Override
    public void setOverdraftlimit(float overdraftlimit) {
        this.overdraftlimit = overdraftlimit;
    }
}
```

## Child component of GiroAccount - StudentAccount

```
@Entity
public class StudentAccount extends GiroAccount {
    String student_number;

    public StudentAccount() {
        super();
    };

    /**
     * has student_number as defining feature
     *
     * @param userID unique identifies user
     * @param accountName unique per user
     * @param accountType defines type of account
     * @param overdraftlimit sets overdraftlimit to certain amount
     * @param student_number sets studentnumber as unique identifier
     */
    public StudentAccount(
        int userID,
        String accountName,
        AccountType accountType,
        float overdraftlimit,
        String student_number) {
        super(userID, accountName, accountType, overdraftlimit);
        this.student_number = student_number;
    };

    /**
     * has student_number as defining feature, overdraftlimit if not set, set to 0
     *
     * @param userID unique identifies user
     * @param accountName unique per user
     * @param accountType defines type of account
     * @param student_number sets studentnumber as unique identifier
     */
    public StudentAccount(
        int userID, String accountName, AccountType accountType, String student_number) {
        super(userID, accountName, accountType);
        this.student_number = student_number;
    };
}
```

## 2 Code Metrics

For Code Analysis we used the IntelliJ Plugin Metrics Reloaded.  
[<https://plugins.jetbrains.com/plugin/93-metricsreloaded/>]

### Total Lines of Code

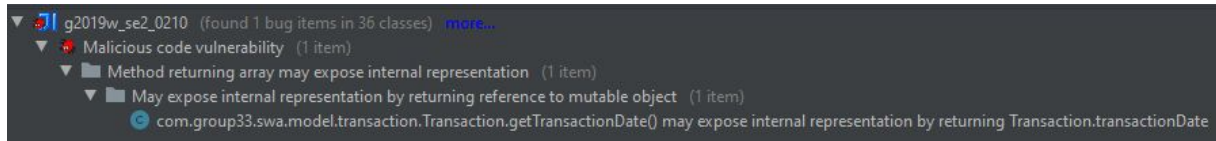
Package metrics	Module metrics	File type metrics	Project metrics			
package	LOC	LOC(rec)	▼LOCp	LOCp(rec)	LOCt	LOCt(rec)
public	557	985	557	985	0	0
com.group33.swa.model.account	301	301	301	301	0	0
com.group33.swa.model.transaction	298	298	298	298	0	0
public.js	225	225	225	225	0	0
public.prototype	203	203	203	203	0	0
com.group33.swa.webController.transaction	167	167	167	167	0	0
com.group33.swa.webController.account	144	144	144	144	0	0
com.group33.swa.webServices.account	106	106	106	106	0	0
com.group33.swa.util	102	102	102	102	0	0
com.group33.swa.webServices.transaction	85	85	85	85	0	0
com.group33.swa.webServices.user	85	85	85	85	0	0
com.group33.swa.webController.user	65	65	65	65	0	0
com.group33.swa.exceptions	48	48	48	48	0	0
com.group33.swa.model.user	44	44	44	44	0	0
	40	2,604	40	2,604	0	0
com.group33.swa.repository.transaction	39	39	39	39	0	0
com.group33.swa.repository.account	29	29	29	29	0	0
com.group33.swa.repository.user	28	28	28	28	0	0
com.group33.swa	21	1,579	21	1,579	0	0
com.group33.swa.webController	17	393	17	393	0	0
com.group33.swa.webServices		276		276		0
com		1,579		1,579		0
com.group33.swa.model		643		643		0
com.group33		1,579		1,579		0
com.group33.swa.repository		96		96		0
Total	2,604		2,604		0	
Average	130.20		130.20		0.00	

## Total number of Classes in packages

Package metrics	Module metrics	Project metrics					
package	▲	C	C(rec)	Cp	Cp(rec)	Ct	TC(rec)
			36		36		0
com			36		36		0
com.group33			36		36		0
com.group33.swa		2	36	2	36	0	0
com.group33.swa.exceptions		3	3	3	3	0	0
com.group33.swa.model			14		14		0
com.group33.swa.model.account		6	6	6	6	0	0
com.group33.swa.model.transaction		7	7	7	7	0	0
com.group33.swa.model.user		1	1	1	1	0	0
com.group33.swa.repository			3		3		0
com.group33.swa.repository.account		1	1	1	1	0	0
com.group33.swa.repository.transaction		1	1	1	1	0	0
com.group33.swa.repository.user		1	1	1	1	0	0
com.group33.swa.util		4	4	4	4	0	0
com.group33.swa.webController		1	4	1	4	0	0
com.group33.swa.webController.account		1	1	1	1	0	0
com.group33.swa.webController.transaction		1	1	1	1	0	0
com.group33.swa.webController.user		1	1	1	1	0	0
com.group33.swa.webServices			6		6		0
com.group33.swa.webServices.account		2	2	2	2	0	0
com.group33.swa.webServices.transaction		2	2	2	2	0	0
com.group33.swa.webServices.user		2	2	2	2	0	0
Total		36		36		0	
Average		2,25		2,25		0,00	

## JavaDoc Lines of Code

package ▲	Jc	Jf	JLOC	Jm
com.group33.swa	100.00%		2	0.00%
com.group33.swa.exceptions	100.00%	0.00%	15	50.00%
com.group33.swa.model.account	100.00%	0.00%	61	18.92%
com.group33.swa.model.transaction	57.14%	0.00%	64	16.67%
com.group33.swa.model.user	100.00%	0.00%	10	25.00%
com.group33.swa.repository.account	100.00%		19	100.00%
com.group33.swa.repository.transaction	100.00%		26	100.00%
com.group33.swa.repository.user	100.00%		18	100.00%
com.group33.swa.util	100.00%	0.00%	42	55.56%
com.group33.swa.webController	100.00%		6	100.00%
com.group33.swa.webController.account	100.00%	0.00%	35	100.00%
com.group33.swa.webController.transaction	100.00%	0.00%	31	75.00%
com.group33.swa.webController.user	100.00%	0.00%	16	100.00%
com.group33.swa.webServices.account	100.00%	0.00%	58	100.00%
com.group33.swa.webServices.transaction	100.00%	0.00%	42	87.50%
com.group33.swa.webServices.user	100.00%	0.00%	45	100.00%
<b>Total</b>			<b>490</b>	
Average	91.67%	0.00%	30.62	48.55%



Following Bug was found when trying to save a `java.sql.Date`. FindBugs-IDEA issued a mutable date-variable which “*may expose internal representation by returning reference to mutable object*”.

We fixed this bug by correcting the Setter-Method and create a Deep-Copy before setting..

[<https://stackoverflow.com/questions/18954873/malicious-code-vulnerability-may-expose-internal-representation-by-incorporati>]

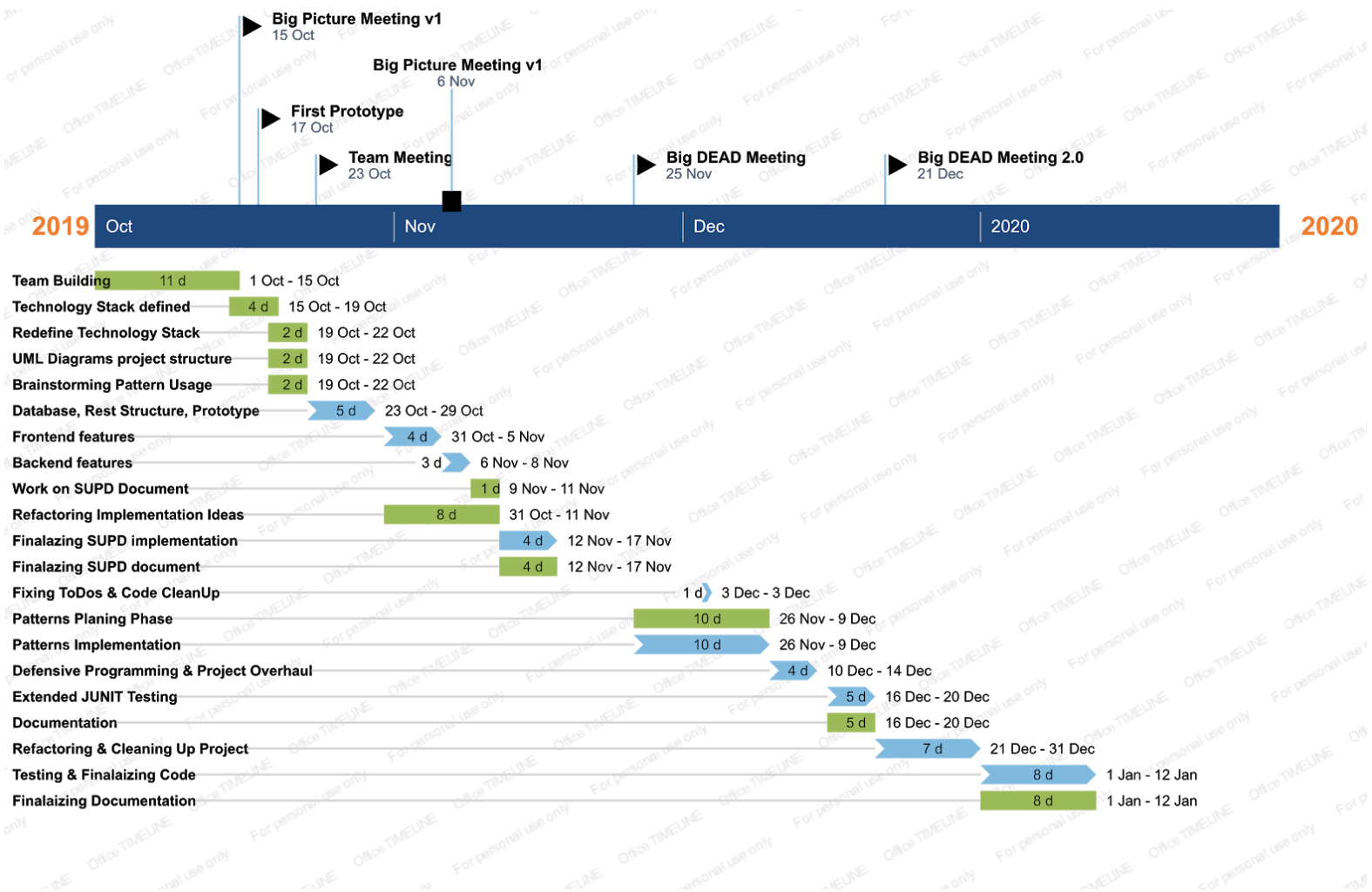
```
public java.sql.Date getTransactionDate() {  
    return new Date(transactionDate.getTime());  
}
```

For spotting Bugs and Security Issues in our Code we used the IntelliJ Plugin FindBugs-IDEA\*

\*<https://plugins.jetbrains.com/plugin/3847-findbugs-idea>

## 3 Team Contribution

### 3.1 Project Tasks and Schedule





### 3.2 Distribution of Work and Efforts

<b>Tasks</b>	<b>Members</b>	<b>Time Investment</b>
<i>Planning</i>	Alex, Lukas, Vincent	<i>15 hours</i>
<i>Meetings</i>	Alex, Lukas, Vincent	<i>10 hours</i>
<i>Documentation</i>	Alex, Lukas, Vincent	<i>20 hours</i>
<i>Project Init</i>	<i>Alex</i>	<i>2 hours</i>
<i>Database Init</i>	<i>Lukas</i>	<i>1 hours</i>
<i>Prototype</i>	<i>Vincent</i>	<i>3 hours</i>
<i>Backend</i>	<i>Lukas, Alex</i>	<i>80 hours</i>
<i>Frontend</i>	<i>Vincent</i>	<i>40 hours</i>
<i>Patterns</i>	Alex, Lukas, Vincent	<i>10 hours</i>
<i>Testing</i>	Alex, Lukas, Vincent	<i>5 hours</i>

Most of the time invested in this project was spent together as a team at university. This streamlined the process of communication within the team. Problems and the discussion of how to achieve certain features was achieved much faster and easier because every team member was at hand. The workload was pretty much equally split among the three of us.

# A1 HowTo

Step by Step Guide build, start, initialize:

1. Clone the Git Repository

```
console> git clone https://lab.swa.univie.ac.at/submission/g2019w_se2_0210.git
```

2. Navigate to the subdirectory

```
console> cd /g2019w_se2_0210
```

3. Checkout SUPD Branch

```
console/g2019w_se2_0210> git checkout 2019w_se2 _supd
```

4. Maven clean to prevent building errors

```
console/g2019w_se2_0210> mvn clean
```

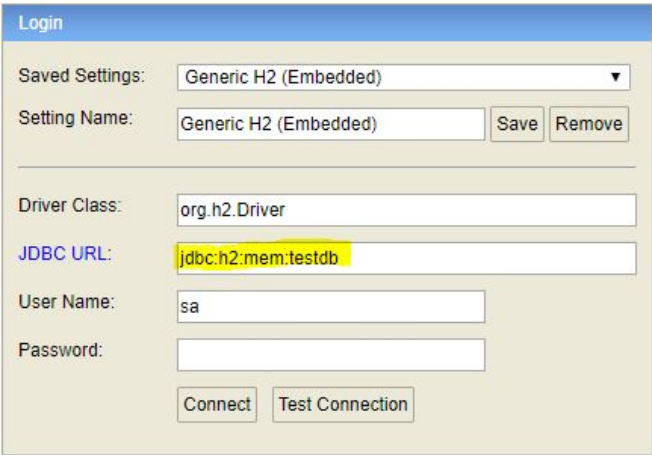
5. Build the jar file

```
console/g2019w_se2_0210> mvn package
```

6. Run the jar file

```
console/g2019w_se2_0210> java -jar ./target/g2019w_se2_2019-0.0.1-SNAPSHOT.jar
```

7. to access the expense tracker open <http://localhost:8080> in your browser
8. <Optional> For Database Access navigate to <http://localhost:8080/h2> and change the JDBC URL to that shown in picture



Login

Saved Settings: Generic H2 (Embedded) ▼

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:testdb

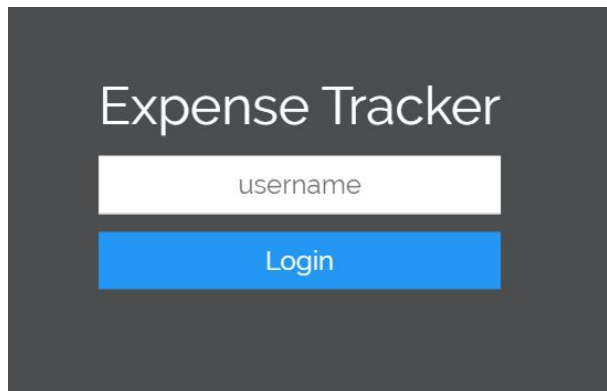
User Name: sa

Password:

Connect Test Connection

## Use Case Executions:

### Use Case 1: Logging in

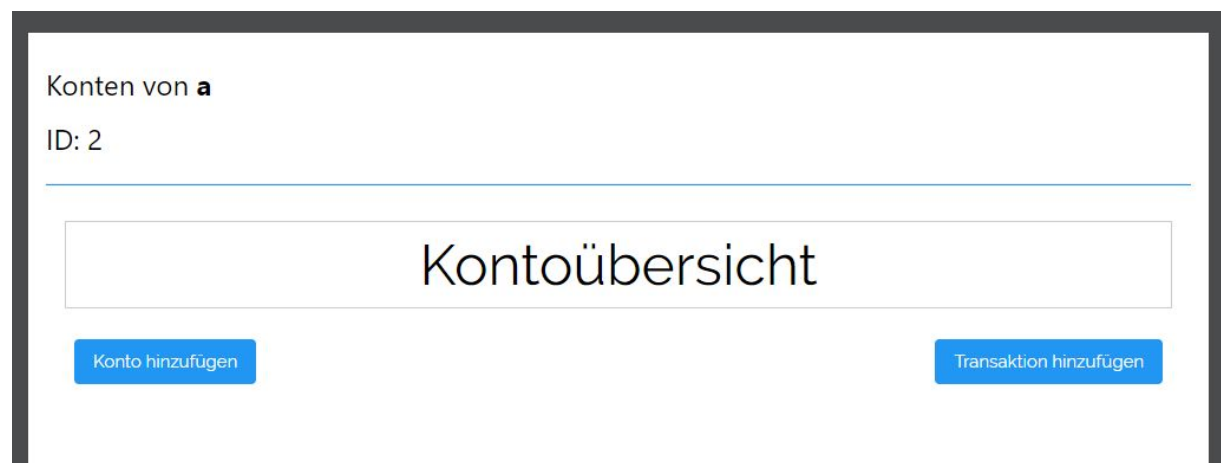


The image shows a login form for an "Expense Tracker". It has a dark grey background. At the top, the text "Expense Tracker" is displayed in white. Below it, there is a white input field with the placeholder text "username". Underneath the input field is a blue button with the text "Login" in white.

Path: ./

Login by entering your username.

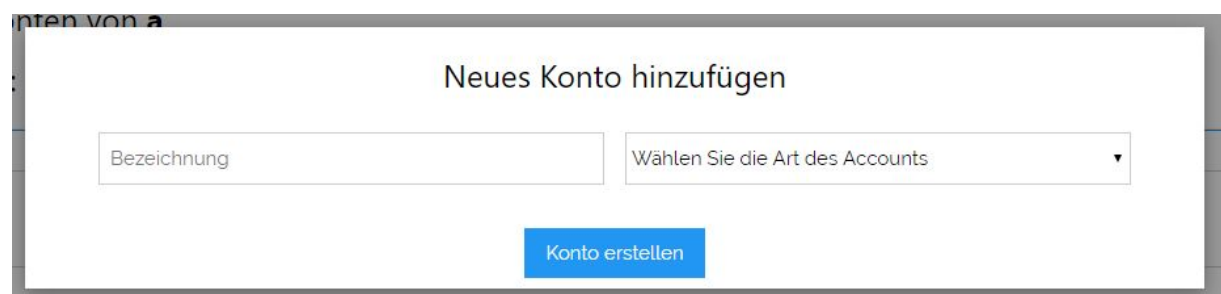
### Use Case 2: Add an Account



The image shows a web page titled "Kontoübersicht". At the top left, it says "Konten von a" and "ID: 2". Below this, there is a large white box with the title "Kontoübersicht". At the bottom of the page, there are two blue buttons: "Konto hinzufügen" on the left and "Transaktion hinzufügen" on the right.

Path: ./ktouebersicht.html

After logging in you are greeted with the „Kontoübersicht“-Page. You can add a new account by clicking the „Konto hinzufügen“ Button.



The image shows a form titled "Neues Konto hinzufügen". It has a light grey background. At the top, the title "Neues Konto hinzufügen" is displayed in black. Below it, there are two input fields: "Bezeichnung" on the left and "Wählen Sie die Art des Accounts" on the right, which is a dropdown menu. At the bottom of the form, there is a blue button with the text "Konto erstellen" in white.

A new modal opens. After entering the Account-Description and selecting the type of account, other fields for entering additional information may appear.

Use Case 3: Add a Transaction:

The screenshot shows a modal titled "Neue Transaktion hinzufügen". It contains several input fields: a date field with "17.11.2019", a text field with "Expensive Burgers", a numeric field with "3", a dropdown menu with "Food", a numeric field with "2", a dropdown menu with "PRIVATE", and a numeric field with "100". A blue button labeled "Transaktion erstellen" is at the bottom.

On clicking „Neue Transaktion“ a new modal appears. Upon entering all the necessary Information (Date, Title, fromAccount, toAccount, Amount, Category and Type) and clicking „Transaktion erstellen“ a new Transaction is made.

Use Case 4: List Transactions:

konto 2		
Typ: CASH		Kontostand: 345
<div>Transaktion hinzufügen</div> <div>Sort By ▼</div>		
Rent SELF 2019-11-17	-600	
Expensive Burgers FOOD 2019-11-17	-50	
Cheap Burgers FOOD 2019-11-17	-5	
Cash StartAmount SELF 2019-11-17	1000	

When clicking the title of an Account you will get to the transaction listing. Using the „Sort By“ dropdown menu you can list the Transaction by Incoming/Outgoing, Category and Type