



# BancoMax

Modul 326V Projektarbeit

Sven, Lewin & Manuel

2GWI KSB

13.06.2021

# Inhalt

1	Vorwort .....	5
1.1	Aufgabenstellung.....	5
1.2	Projektauswahl.....	5
1.3	Unsere Ziele.....	6
2	Planung.....	6
2.1	Pflichtenheft .....	6
2.2	Ressourcenplanung .....	6
2.3	UI-Konzept.....	7
2.3.1	Stand-by-View .....	7
2.3.2	Login-View .....	8
2.3.3	Master-View .....	8
2.3.4	AccountInfo-View .....	9
2.3.5	PinChange-View.....	9
2.3.6	Deposit-Views.....	10
2.3.7	DepositConfirm-View .....	10
2.3.8	Withdrawal-View.....	11
2.3.9	WithdrawalConfirm-View.....	12
2.3.10	TransactionSuccess-View .....	12
2.3.11	Admin-View .....	13
2.3.12	Payout-View .....	13
2.3.13	Dashboard-View .....	14
3	Diagramme .....	15
3.1	Datenbankdiagramm.....	15
3.2	Anwendungsfalldiagramm .....	16
3.3	Aktivitätsdiagramm .....	17
3.4	Sequenzdiagramm.....	19
4	Realisierung .....	20

4.1	Journal .....	20
4.2	Herangehensweise .....	22
4.2.1	Arbeitsaufteilung .....	22
4.2.2	Beziehung zur Realität .....	22
4.2.3	Design .....	23
4.2.4	JavaFX .....	25
4.2.5	GitHub .....	26
4.2.6	Datenbank .....	29
4.2.7	Sicherheit .....	29
4.3	Herausforderungen und Ereignisse .....	29
4.3.1	Server .....	29
4.3.2	Exchange-Rate API .....	30
4.3.3	Banknoten-Bilder .....	31
4.3.4	PDF-Maker .....	31
5	Programmcode .....	32
5.1	Projekt-Struktur .....	32
5.2	Implementierung PDF-Maker .....	33
5.3	Implementierung Exchange-Rate API .....	35
5.4	Datenbankverbindung .....	36
5.5	Interaktion mit der Datenbank .....	37
5.5.1	Beispiel: SELECT-Befehl .....	37
5.5.2	Beispiel: INSERT-Befehl .....	37
5.5.3	Beispiel: UPDATE-Befehl .....	38
5.6	Utility .....	38
5.6.1	Enum-Beispiel: Salutation .....	39
5.6.2	isNumeric()-Methode .....	39
5.6.3	formatMoney()-Methode .....	39
5.7	Security: Implementierung .....	40

5.7.1	Hashing .....	40
5.7.2	Kartenummer-Verschlüsselung .....	40
6	Abweichungen vom UI-Konzept .....	41
7	Schlussbericht .....	43
7.1	Reflexion .....	43
7.2	Fazit .....	43

# 1 Vorwort

## 1.1 Aufgabenstellung

Projektgruppenbildung war schnell klar, denn es gab in unserer Halbkasse mindestens eine Dreiergruppe, diese bildeten wir drei, Sven, Lewin und Manuel. Was die Projektthemen anbelangt, war alles offen, einzig die untenstehenden Anforderungen müssen erfüllt werden. Die Projektvoraussetzungen sind, dass das Projekt in Java und mit der IDE Eclipse zu programmieren ist. Wir haben Herrn Glanzmann gefragt, ob wir anstatt Eclipse IntelliJ benutzen dürfen. Zudem ist vorgegeben, dass die Praktiken der objektorientierten Programmierung und die strikte Datenkapselung eingehalten werden muss. Die zu erwartenden Resultate sind die Folgenden:

- Lauffähige Applikation
- Dokumentation
- UML-Konzepte
  - UseCase-Diagramm
  - Aktivitäts-Diagramm
  - Sequenz-Diagramm
  - Klassen-Diagramm

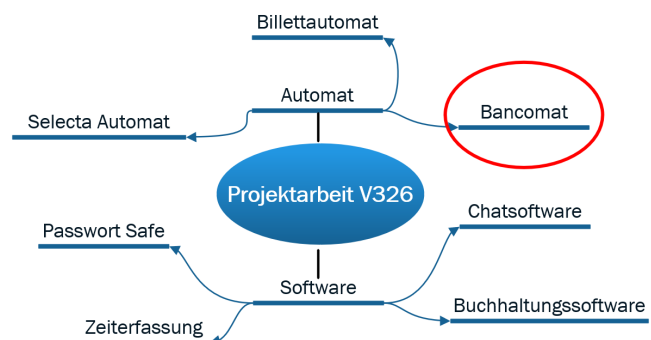
Die Bewertungsgewichtung für diese Resultate sieht wie folgt aus:

- Lauffähige Applikation 60%
- Dokumentation 20%
- UML-Konzepte 20%
  - UseCase-Diagramm
  - Aktivitäts-Diagramm
  - Sequenz-Diagramm
  - Klassen-Diagramm

Der Nutzen dieses Projekts ist es, das objektorientierte Programmieren in Java und das Wissen über die UML-Diagramme zu vertiefen. Natürlich ist es auch eines unserer Ziele, eine gute Note zu bekommen.

## 1.2 Projektauswahl

Als Herr Glanzmann uns vom Projekt zu erzählen begann, hatten wir nicht wirklich eine Idee, was wir machen sollten. Darum haben wir uns entschieden eine Mindmap zu erstellen. Dann plötzlich sprudelten viele Ideen aus uns dreien heraus. Wie man in der Mind-Map erkennen



kann, hatten wir verschiedenste Vorschläge, z.B. einen Selecta-Automaten, ein Passwort Safe, eine Zeiterfassung, Chatsoftware, usw. Schlussendlich haben wir uns nach für einen Bankautomaten entschieden.

## 1.3 Unsere Ziele

Unsere Hauptziele, die wir uns bezüglich dieses Projekts und des Bancomaten gesetzt haben, sind die Folgenden:

- Das GUI soll übersichtlich gestaltet werden
- Die Benutzerfreundlichkeit muss gewährleistet sein
- Das Programm soll auf Deutsch bedient werden können
- Damit es für die Augen angenehmer zu benutzen ist, soll das UI ein «Darkmode»-Design haben
- Es soll die Grundlegenden Funktionen eines Bancomaten beinhalten
- Es soll eine Datenbank integriert sein
- Der Prozess soll so wirklichkeitsgetreu gestaltet sein wie möglich

## 2 Planung

### 2.1 Pflichtenheft

Muss-Kriterien	<ul style="list-style-type: none"> <li>• Stand-by Screen, der anfangs angezeigt wird</li> <li>• Kartenummer kann eingegeben werden*</li> <li>• Verifikation mit PIN-Nummer*</li> <li>• Kontostand kann angezeigt werden</li> <li>• Geld kann abgehoben werden</li> <li>• Bancomat hat nur begrenzte Geldreserven</li> <li>• Leerer Automat kann keine Ausgaben ausführen</li> <li>• Quittung kann ausgedruckt werden (wird angezeigt)</li> <li>• Automat kann nachgefüllt werden</li> </ul>
Soll-Kriterien	<ul style="list-style-type: none"> <li>• Man kann Geld einzahlen</li> <li>• Fremdwährungen können ausgegeben werden</li> <li>• Alle Daten sind in einer strukturierten Datenbank gespeichert</li> </ul>
Kann-Kriterien	<ul style="list-style-type: none"> <li>• Fremdwährungen sind aktuell (API)</li> <li>• Fremdwährungen können eingezahlt (&amp; umgewandelt) werden</li> <li>• Neue Kreditkarten und User können erstellt werden</li> <li>• Letzte Bewegungen können unter Kontoinformationen angezeigt werden</li> <li>• Zusatzdaten von Kreditkarte werden ebenfalls abgespeichert</li> </ul>

\*Es werden Beispielbenutzer von uns erstellt, die mit allen dazugehörigen Informationen in der Datenbank eingespeichert sind.

### 2.2 Ressourcenplanung

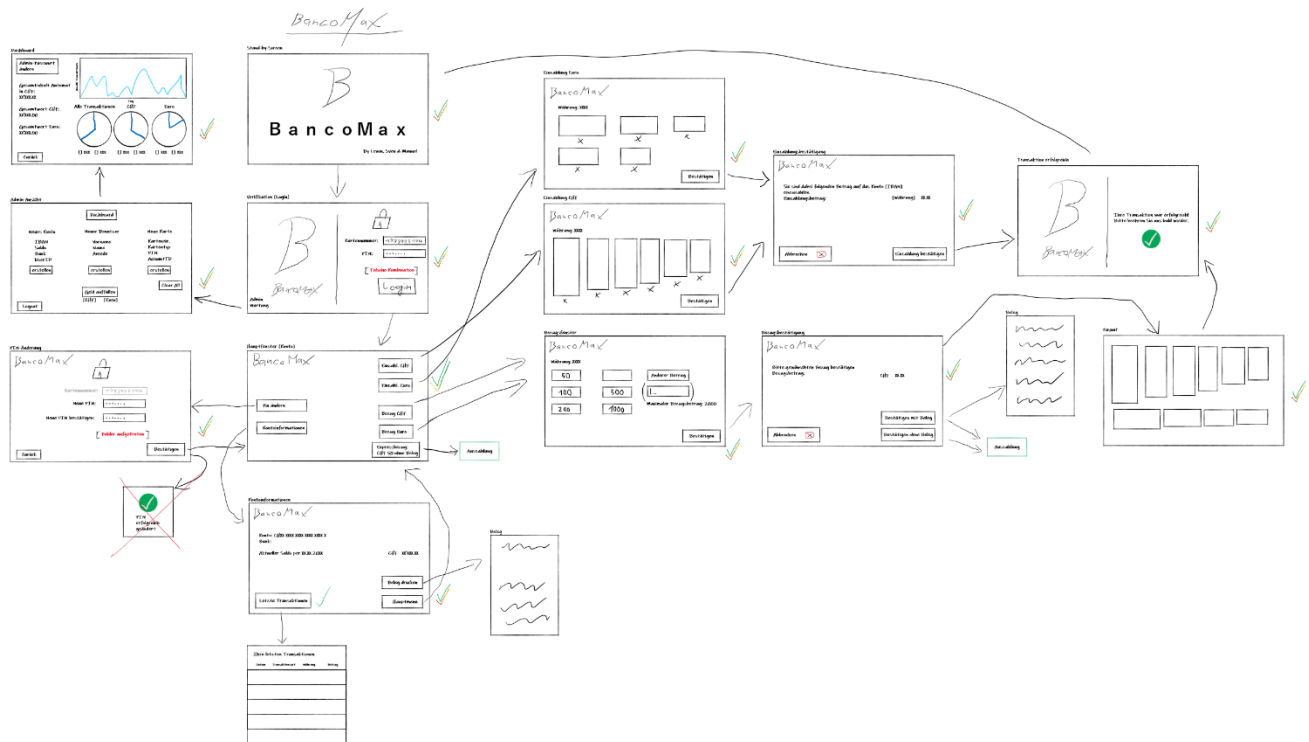
Um unser Projekt umzusetzen, werden wir die Programmiersprache Java und für die Datenbank MySQL verwenden.

Die Tools, die wir benutzen wollen, sind:

- Git/Github für die Sicherungsstruktur
- IntelliJ als IDE für die Programmierung
- MS Visio für die Erstellung von Diagrammen
- MS Word um die Dokumentation zu erstellen
- MySQL Workbench für die Verwaltung der Datenbank

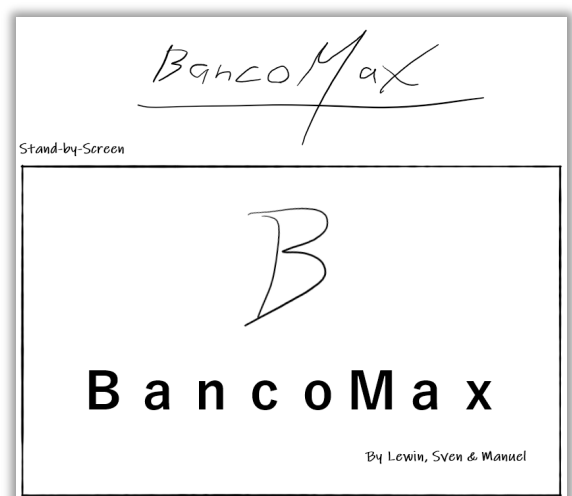
## 2.3 UI-Konzept

Um uns die Implementierung zu vereinfachen, haben wir Entwürfe für jedes Fenster des GUIs erstellt. Dieses Entwurfskonzept haben wir mit Microsoft Whiteboard und einem Notebook-Stift gezeichnet. Das Design ist darin noch nicht ausgereift, jedoch legt es schon einen soliden Grundstein für die Funktionalität des Programmes. Mit der Zeit sind immer mehr Fenster dazu gekommen, als uns noch mehr Erweiterungen für das Projekt eingefallen sind. Wir werden in den folgenden Unterkapiteln noch auf die einzelnen Fenster eingehen aber hier erst einmal die vollständige UI-Konzeptzeichnung für unser Projekt:



### 2.3.1 Stand-by-View

Als erstes haben wir einen Stand-by-screen erstellt, der auftauchen soll, wenn das Programm gestartet wird. Dieser zeigt unser Logo und den Firmenschriftzug an und leitet auf die Nächste Seite weiter, sobald er angeklickt oder berührt wird.



### 2.3.2 Login-View

Als nächstes im Ablauf ist der Login-Screen.

Rechts kann man seine Kartennummer eingeben, was das Einstecken der Karte simulieren soll, und anschliessend die PIN eingeben.

Unten kann man danach auf einen Login-Button klicken, um die Eingabe zu bestätigen. Sollte es ein Problem mit den Login-Daten geben, erscheint eine rote Meldung mit der Ursache des Problems.

Links sieht man nochmal unser Logo und weiter unten gibt es Buttons für separate Logins. Diese sind für ein Admin-Login und falls der Bancomat gewartet werden muss.

Verifikation (Login)

Verifikation (Login)

BancoMax  
Admin  
Wartung

Kartennummer: 1273985-1736

PIN: .....

[ Falsche Kombination ]

Login

### 2.3.3 Master-View

Der nächste Entwurf beschreibt das Hauptfenster welches man nach dem Login zu sehen bekommt. Links kann man seinen PIN ändern oder Informationen über dieses Konto abrufen. Rechts gibt es Buttons, um Einzahlungen oder Bezüge zu tätigen. Um Benutzern mit wenig Zeit einen schnellen Weg zum Bezug von Bargeld zu schaffen, gibt es einen Button «Expressbezug», der direkt, also ohne vorherige zweite Bestätigung 50 CHF ausgibt.

Hauptfenster (Konto)

BancoMax

Pin ändern

Kontoinformationen

Einzahl. CHF

Einzahl. Euro

Bezug CHF

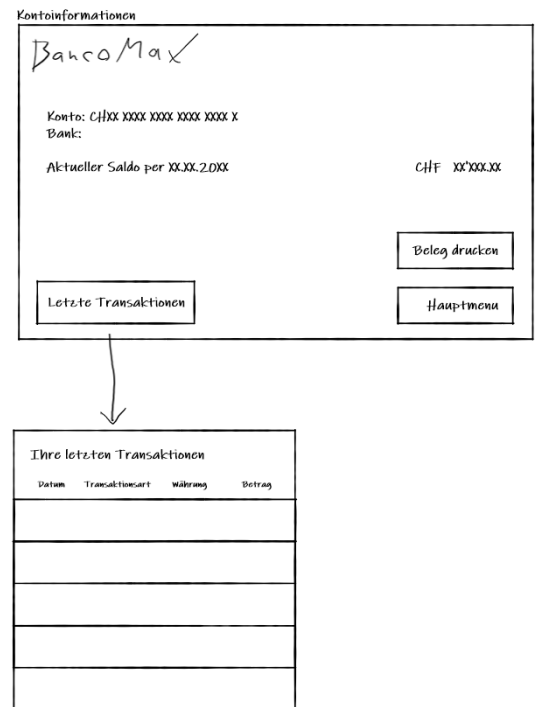
Bezug Euro

Expressbezug:  
CHF 50 ohne Beleg



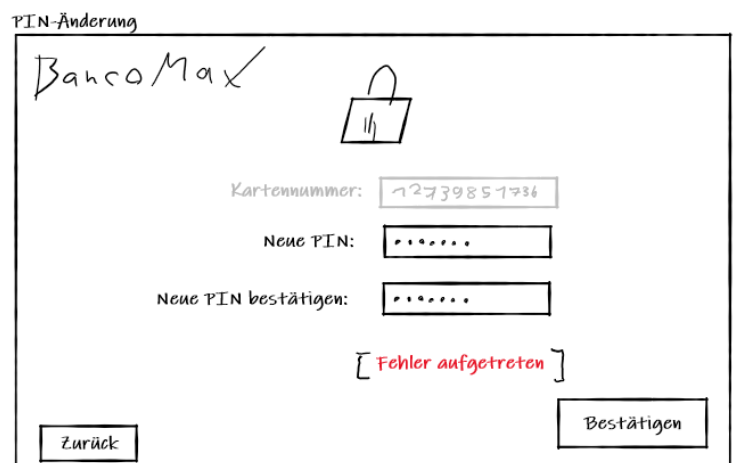
### 2.3.4 AccountInfo-View

Auf dem Kontoinformationsfenster sieht man die IBAN-Nummer, den Namen der Bank und den aktuellen Kontostand sowie das Datum. Unten links könnten alle bisher getätigten Transaktionen über eine PopUp-Liste eingesehen werden. Unten rechts kann ein Beleg gedruckt werden, auf dem der Name der Bank, der Kartentyp, die Kartennummer, die IBAN-Nummer, der Kontostand und das aktuelle Datum festgehalten werden. Wenn man fertig ist, kann man unten rechts auch auf den Button «Hauptmenu» klicken, um zurück zur Master-View zu gelangen



### 2.3.5 PinChange-View

Wenn man die Option «Pin ändern» auswählt, wird man auf dieses Fenster weitergeleitet. Hier sieht man seine Kartennummer, welche man aber nicht verändern kann. Darunter kann man seinen neuen PIN eingeben, der mit einer zweiten Eingabe bestätigt werden muss. Sollte ein Fehler auftreten, wird dies darunter, mit einer roten Meldung über die Ursache, angezeigt. Unten rechts kann man die Eingabe bestätigen und unten links kann man wieder auf das Hauptfenster zurück.



### 2.3.6 Deposit-Views

Auf diesen Entwürfen kann man die Einzahlungsfenster betrachten. Für die beiden Währungen Euro und Franken mussten wir unterschiedliche Fenster erstellen, weil beide unterschiedlich dargestellt werden. Euro werden waagrecht und Franken senkrecht dargestellt. Im Fenster kann man oben links die Währung sehen und im Mittelpunkt des Bildes stehen die Noten, die man Auswählen kann. Auf diesen sieht man jeweils unterhalb nochmal die Anzahl, wie viele Exemplare der jeweiligen Note man beziehen will. Unten rechts kann man anschliessend seine Eingabe bestätigen.

Einzahlung Euro

Einzahlung CHF

### 2.3.7 DepositConfirm-View

Nach der ersten Bestätigung wird man auf dieses Fenster weitergeleitet. Hier werden die Informationen aus der vorherigen Einzahlung noch einmal aufgelistet. Die IBAN Nummer des Bankkontos, der Betrag und die Währung. Nun kann man mit dem Button rechts unten die Einzahlung ein zweites und finales Mal bestätigen. Wenn man sich aber umentschieden hat, kann man den Vorgang links unten mit abbrechen, und zur Master-View zurückkehren.

Einzahlungsbestätigung

### 2.3.8 Withdrawal-View

Auf dieser View wird das Fenster für den Bezug von Geld dargestellt. Links oben kann wieder die Währung eingesehen werden. Darunter befinden sich die auswählbaren Beträge, welche ausgezahlt werden können. Sollte man einen nicht aufgeführten Betrag ausgeben wollen, befindet sich rechts ein Feld, indem man einen eigenen Betrag

Bezugsfenster

BancoMax

Währung: XXXX

50		Anderer Betrag
100	500	(  ... )
200	1000	

Maximaler Bezugsbetrag: 2000

Bestätigen

angeben kann, solange er in Noten auszahlbar ist. Der Maximalwert einer einzigen Auszahlung beträgt 2000 CHF.-. Nun kann man den gewählten Betrag rechts unten bestätigen.

### 2.3.9 WithdrawalConfirm-View

Wenn man einen Bezug machen will, kann man auf diesem Fenster noch einmal die Informationen über die Transaktion einsehen. Die ersichtlichen Infos sind die Währung und der zu beziehende Betrag. Rechts unten kann man den Bezug bestätigen und auswählen, ob man einen Beleg dazu haben möchte. Sollte man sich das Ganze anders überlegt haben, kann der Vorgang links unten mit einem Klick auf Abbrechen terminiert werden.

Bezugsbestätigung

The interface for the WithdrawalConfirm-View is a rectangular box. At the top left, the text 'BancoMax' is written in a handwritten style. Below it, the text 'Bitte gewünschten Bezug bestätigen' is followed by 'Bezugsbetrag:' and 'CHF XX.XX'. At the bottom left, there is a button labeled 'Abbrechen' with a red 'X' icon. At the bottom right, there are two buttons: 'Bestätigen mit Beleg' and 'Bestätigen ohne Beleg'.

### 2.3.10 TransactionSuccess-View

Wenn man eine Transaktion durchgeführt hat, wird man auf dieses Fenster weitergeleitet. Hier ist links unser Logo zu sehen und rechts eine Bestätigung, dass die Transaktion erfolgreich war. Sobald man irgendwo in diesem Fenster klickt oder es berührt, wird man wieder zur StandBy-View weitergeleitet.

Transaktion erfolgreich

The interface for the TransactionSuccess-View is a rectangular box. On the left side, there is a large stylized 'B' logo with 'BancoMax' written below it. On the right side, there is a vertical line separating the logo from the text. To the right of the line, the text reads 'Ihre Transaktion war erfolgreich!' followed by 'Bitte beehren Sie uns bald wieder.' Below this text is a green circular icon with a white checkmark.

### 2.3.11 Admin-View

Sollte man sich als Admin einloggen, erscheint die rechts abgebildete Ansicht. In der Mitte sieht man 3 Spalten, mit je einem «erstellen»-Button darunter. Diese dienen dem Erstellen neuer Konten, Benutzer und neuer Karten. Neben diesen Spalten gibt es Textfelder, in welche die entsprechenden Daten einge-  
füllt werden sollen. Rechts darunter gibt es einen «Clear All»-Button, um alle Ein-

Admin-Ansicht

Das Diagramm zeigt die Admin-Ansicht mit folgenden Elementen:

- Dashboard**: Ein Button oben in der Mitte.
- Neues Konto**: Eine Spalte mit den Textfeldern IBAN, Saldo, Bank und UserID, gefolgt von einem **erstellen**-Button.
- Neuer Benutzer**: Eine Spalte mit den Textfeldern Vorname, Name und Anrede, gefolgt von einem **erstellen**-Button.
- Neue Karte**: Eine Spalte mit den Textfeldern KartenNr., Kartentyp, PIN und AccountID, gefolgt von einem **erstellen**-Button.
- Clear All**: Ein Button rechts unten.
- Geld auffüllen**: Ein Button in der Mitte, unter dem die Währungen **[CHF]** und **[Euro]** angegeben sind.
- Logout**: Ein Button unten links.

gaben in den Textfeldern zu löschen. Unten in der Mitte befindet sich ein «Geld auffüllen»-Button. Wenn man diesen drückt, erscheinen zwei neue Buttons auf dem Fenster, diese sind mit «CHF auffüllen» und «Euro auffüllen» beschriftet. So kann der Administrator sich entscheiden, welche Währung er auffüllen möchte. Drückt man einen der Beiden wird man auf die Einzahlungsseite der entsprechenden Währung weitergeleitet. Links unten kann man schlussendlich noch auf Logout drücken, um sich wieder abzumelden.

### 2.3.12 Payout-View

In diesem Fenster, das bei jeder Auszahlung aufgerufen wird, werden nach einem Bezug alle Noten angezeigt, welche abgehoben worden sind. Die senkrecht stehenden Rechtecke stehen für Schweizer Franken und die waagerechten für Euro. Dies ist nur symbolisch, im Programm werden immer nur die Noten der ausgegebenen Währung angezeigt.

Payout

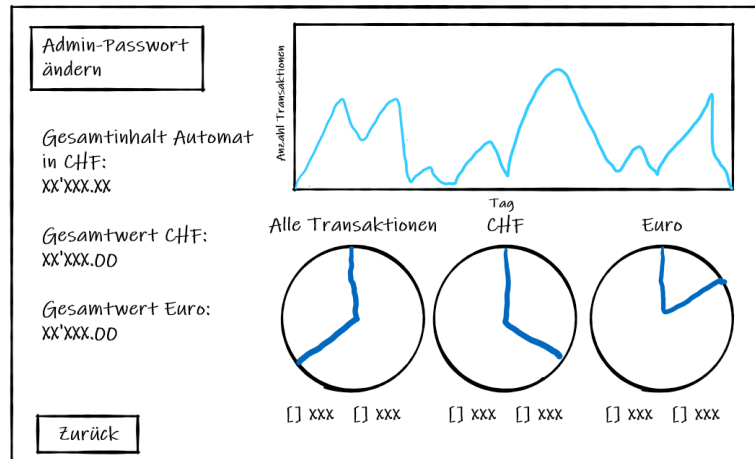
Das Diagramm zeigt die Payout-Ansicht mit zwei Reihen von Rechtecken:

- Obere Reihe**: Sechs senkrecht stehende Rechtecke, die Schweizer Franken repräsentieren.
- Untere Reihe**: Vier waagrecht stehende Rechtecke, die Euro repräsentieren.

### 2.3.13 Dashboard-View

Das Dashboard ist ein, nur vom Administrator einsehbares, Fenster, über das der aktuelle Status des Bankomaten angezeigt wird. Links oben gibt es einen Button, um das Admin-Passwort zu ändern. Darunter sieht man als erstes den Zusammengerechneten Gesamtwert des ganzen Geldvorrats des

Dashboard

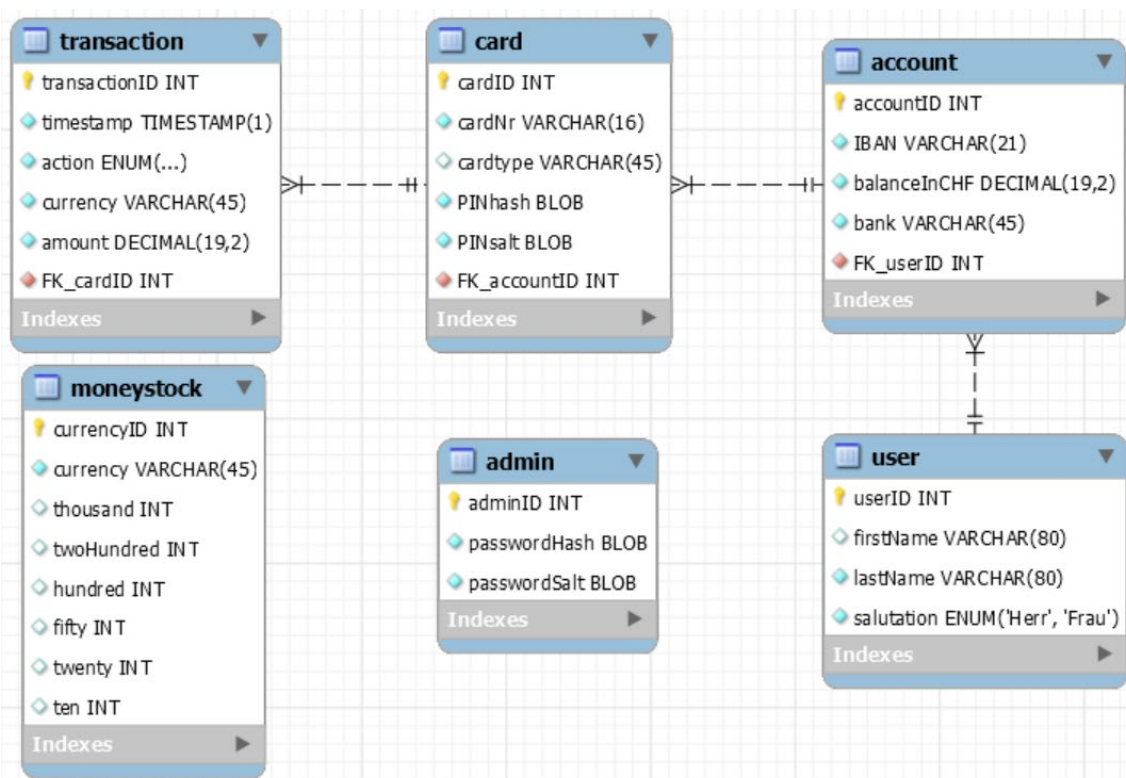


Bankautomaten mit dem aktuellen Währungskurs. Die beiden Werte darunter sind die Totalsummen sämtlicher im Vorrat vorhandenen Noten der entsprechenden Währung. Oben rechts wird ein Liniendiagramm angezeigt, welches im Zeitraum des aktuellen Monats veranschaulicht, an welchem Tag wie viele Transaktionen getätigt wurden. Darunter befinden sich drei Kreisdiagramme. Das erste zeigt, welchen Anteil der Transaktionen die beiden Währungen ausmachen. Die beiden rechten Kreisdiagramme zeigen den Anteil der Bezüge und Einzahlungen bei Transaktionen in der jeweiligen Währung an.

### 3 Diagramme

Die Erstellung von Diagrammen ist ein hilfreicher Weg, um Teile oder auch das Ganze Programm schon vor seiner Implementierung darzustellen. Deshalb haben wir mehrere davon schon sehr früh in der Entwicklung erstellt, um eine bessere Idee von der nötigen Funktionalität zu haben. Diese Diagramme haben uns im Verlauf des Projekts begleitet und unsere Vision fixiert. Wir haben sie aber auch noch während des laufenden Projekts angepasst, da sich unsere Pläne und Ziele ebenfalls verändert haben. Wir haben die Diagramme so ausgewählt, dass sie für den abzubildenden Anwendungsfall am besten geeignet sind. Vor allem bei Besprechungen und Teamdiskussionen über die Implementierung haben sie sich als sehr hilfreich erwiesen.

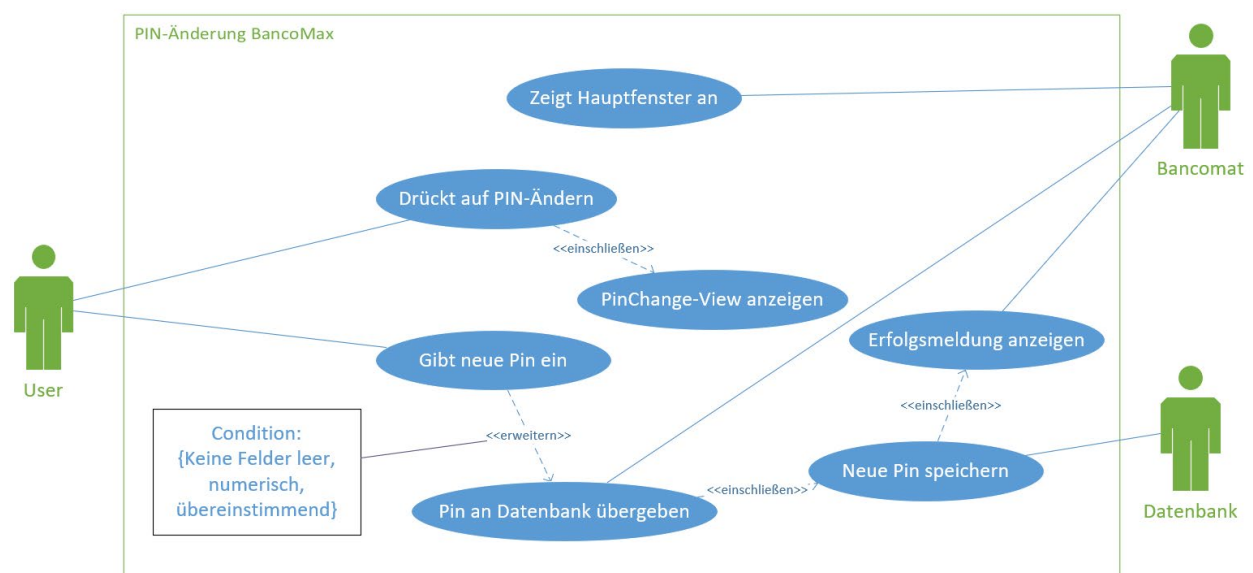
#### 3.1 Datenbankdiagramm



In unserem Datenbankdiagramm wird beschrieben, wie genau unsere Datenbank aufgebaut ist und in welcher Beziehung die Tabellen zu Einander stehen. Wir haben die Tabellen Transaction, Card, Account, Moneystock, Admin und User erstellt. Moneystock und Admin aber jedoch keine Beziehung zu den anderen Tabellen. Diese beiden werden ausschliesslich zur Speicherung von separaten Daten verwendet. Der Zweck von Moneystock ist es, die Anzahl der Noten im Notenvorrat des Bancomaten zu speichern sowie eine Übersicht über die Währungsarten zu haben. Im MoneyStock haben wir uns entschieden, 10er bis 1000er-Noten zur Verfügung zu stellen, dabei ist bei den Euros die Anzahl 1000er-Noten immer auf null, da es in dieser Währung keine solchen Noten gibt.

Die Tabelle Admin beinhaltet ein verschlüsseltes Admin-Passwort, mit dem sich ein Administrator anmelden und so Einsicht in den Notenvorrat und getätigte Transaktionen erhalten kann. Die Tabellen Account, Card und User sind da, damit sich ein Benutzer in den Bancomaten einloggen kann. Dieser steckt eine Karte in den Bancomaten und gibt den PIN ein. Mit einer Beziehung von Card zu Account wird das Bankkonto erkannt und der Bancomat weiss wie viel Geld der Kunde noch hat. Weil ein Account immer von einem User besessen wird, ist auch eine Beziehung zu User vorhanden. Ein User kann mehrere Bankkonten haben und jedes dieser Bankkonten kann mehrere aktive Kreditkarten haben. Der Zweck der Tabelle Transaction ist es, die getätigten Transaktionen aufzuzeichnen und zu speichern, was speziell für die Buchhaltung der Bank wichtig ist.

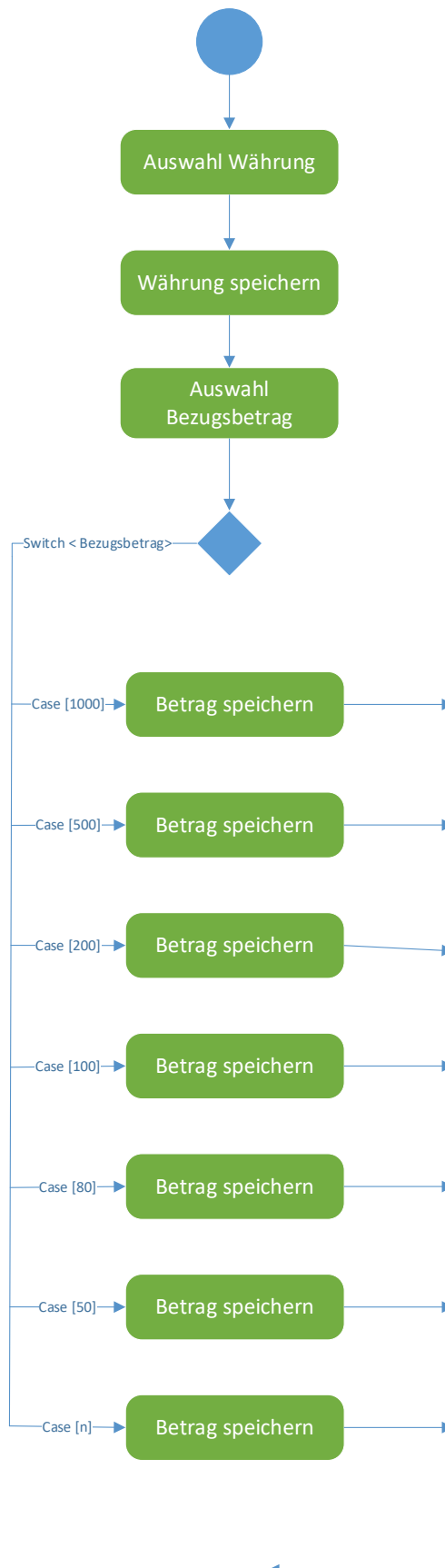
### 3.2 Anwendungsfalldiagramm

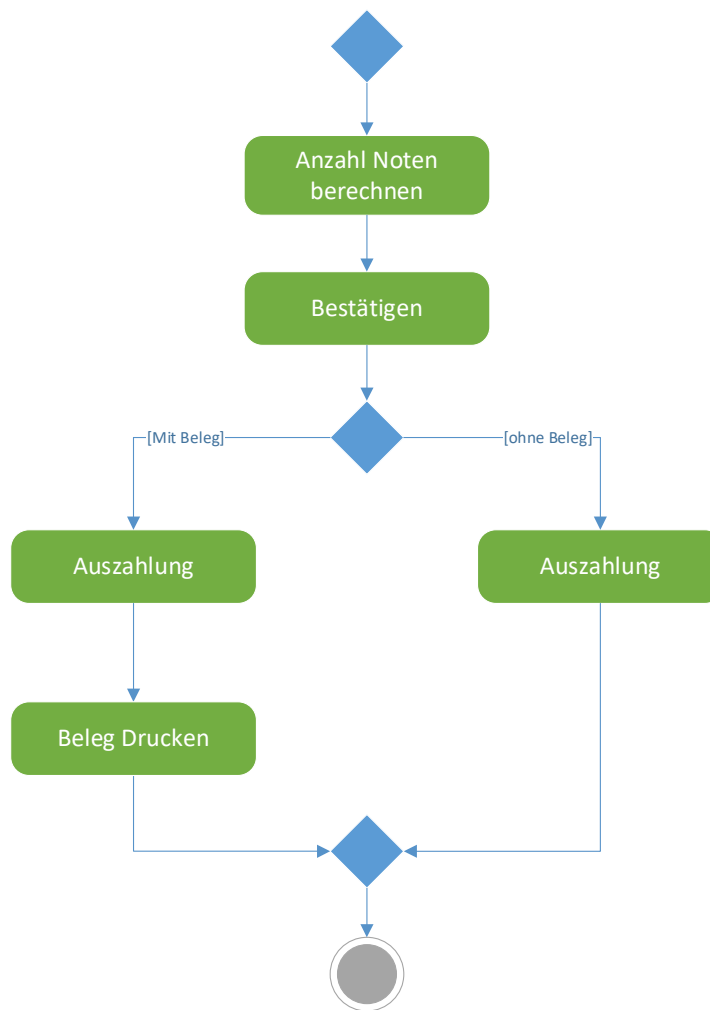


Das Anwendungsfalldiagramm beschreibt, wie die Pin im Programm geändert werden kann. Dazu muss der Nutzer im Hauptmenü die Option «Pin-Ändern» anwählen. Anschliessend zeigt der Bancomat die sogenannte PinChange-View an. Dort kann der Nutzer seine neue Pin eingeben und muss dann die Eingabe bestätigen. Die beiden Felder für die Eingabe und Bestätigung der neuen Pin überprüfen die folgenden Kriterien: Sie müssen beide ausgefüllt sein, sie müssen vom Inhalt her übereinstimmen und der Inhalt muss ein numerischer Wert, also eine Zahl, sein. Wenn der Nutzer zufrieden ist, kann er die Pin bestätigen, womit dann der Bancomat mit einer Bestätigung reagiert. Zugleich wird die Pin in der Datenbank entsprechend angepasst.



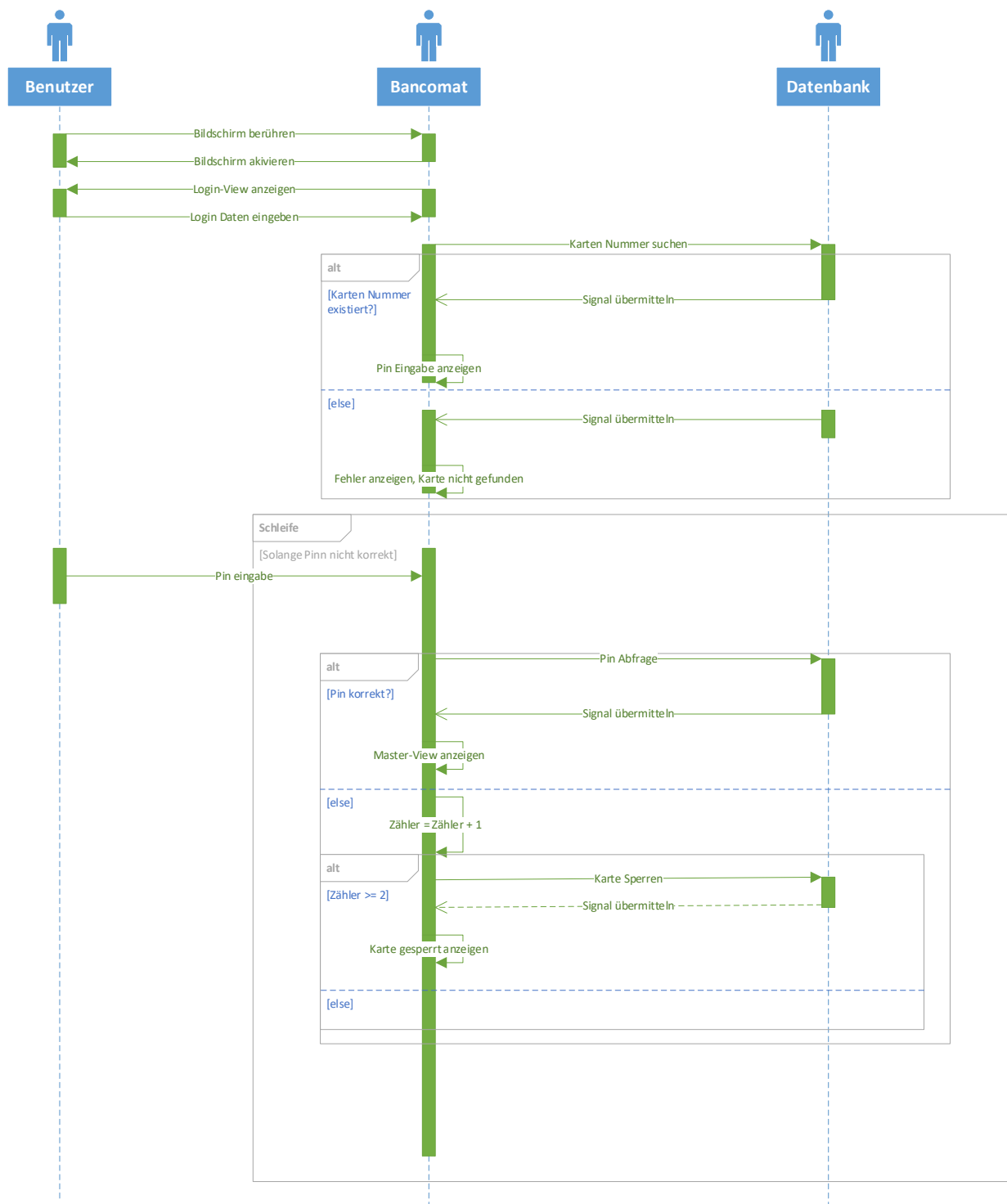
### 3.3 Aktivitätsdiagramm





Unser Aktivitätsdiagramm beschreibt den Auszahlungsprozess des Programmes. Als erstes wird vom User die Währung gewählt, sowie der Bezugsbetrag. Danach werden anhand der Grösse des Betrages die höchsten ausgebbaren Noten ausgesucht. Dies muss dann noch bestätigt werden und der Kunde wird gefragt, ob er einen Beleg möchte. Sollte er dies mit ja beantworten, wird der Beleg der Transaktion gedruckt.

### 3.4 Sequenzdiagramm



Das Sequenzdiagramm beschreibt den Login-Prozess. Wir wollen hier noch kurz anmerken, dass wir den Login-Vorgang im effektiven Programm mit einigen Abweichungen von diesem Programm implementiert haben. Dazu muss der Nutzer zuerst den Bildschirm des Bancomaten berühren oder anklicken. Dieser zeigt sogleich die Login-View an, damit der Nutzer seine Anmeldedaten eingeben kann. Das Programm durchsucht anschliessend die Datenbank und schaut, ob diese Bankkarte überhaupt existiert. Dann muss

die Pin eingegeben werden, welche auch mit der Datenbank abgeglichen wird. Wenn der Nutzer zu oft eine falsche Pin eingibt, wird die Karte gesperrt.

## 4 Realisierung

### 4.1 Journal

Datum	Ergebnisse & Tätigkeiten
09.02.2021	Aufgabenstellung wurde vorgestellt, Gruppenbildung, Festlegung Rahmenbedingungen für Projekt
16.02.2021	Repetition OOP, GUIs & UML-Strukturen
23.02.2021	Pflichtenheft erstellt, Klassendiagramm erstellt
02.03.2021	Grobplanung angefangen, Softwarekonzept angefangen
09.03.2021	Grobplanung fertig
13.03.2021	GitHub-Repository erstellt
14.03.2021	Temporäres StandBy-View Design, Datenbankverbindung
16.03.2021	Input Programmierung, Weiterarbeiten an Softwarekonzept
18.03.2021	Weiterarbeiten an Views
20.03.2021	Admin-View und Button hinzugefügt, Security Klasse für Password-Hashing angefangen und beendet, Vergleich mit Datenbank bei Login implementiert
21.03.2021	Bugfix: Kartenerstellung, User- und AccountInfo hinzugefügt, mehrere SQL-Abfragen erstellt, Bugfix: Karte mit 16 Zeichen
23.03.2021	UserInfo & AccountInfo zusammengefasst zu Info, Verbesserungen im LoginController, createAccount und createUser-Optionen zu AdminController hinzugefügt; Bugfix: createAccount und insertAccount Funktionen
25.03.2021	PINchange und dazugehöriger Controller hinzugefügt
27.03.2021	AccountInfo-Klasse erstellt, diverse Code-Strukturen verbessert
29.03.2021	Voll funktionsfähige Withdrawal-Klasse implementiert, WithdrawalConfirm-Klasse angefangen
30.03.2021	WithdrawalConfirm fertig gemacht
31.03.2021	Funktionierende Wechselkurs-Kalkulation für Euro
02.04.2021	Bugfix: Exchange-Rate API, org.json-Library hinzugefügt
05.04.2021	Bugfix: Expressbezug, MoneyStock wird nun bei Transaktionen verändert, Standard-Wechselkurs hinzugefügt
06.04.2021	Zwischenpräsentation mit Review & Demo
11.04.2021	TransactionSuccess-View fertig implementiert
12.04.2021	Bei Geldbezügen wird Transaktion in Datenbank erstellt, Transaktionsliste als PopUp in AccountInfo-View hinzugefügt
13.04.2021	Bugfix: Transaktionsliste limitiert auf max. 10 Transaktionen; DepositCHF implementiert, DepositEuro angefangen, DepositInfo erstellt, neue Deposit-Buttons in Master-View hinzugefügt, voll funktionsfähige depositConfirm-Klasse erstellt, depositEuro fertig implementiert, Bugfix: Transaktionsliste richtig sortiert, Funktion für Notenangabe per Klick implementiert bei Bezügen, Auffüllen-Funktion in Admin-View hinzugefügt
16.04.2021	Design Überarbeitung: Login-View v2
17.04.2021	Design Überarbeitung: Master-View v1.2, Master-View v2, PinChange v2, AccountInfo v2, weitere Design-Tweaks bei Login

18.04.2021	Design Überarbeitung: deposit-Views, depositConfirm, withdrawal-View, withdrawalConfirm, TransactionSuccess-View, Admin-View v2, Vollständiges Hotkey-Handling implementiert, Bugfix: Zerohandling bei Einzahlungen
19.04.2021	Transaktions-Liste vollständig neu implementiert (Struktur, Funktion & Design ebenfalls neu, es können nun unlimitiert viele Transaktionen angezeigt werden)
20.04.2021	BancoMax-Logo geändert, Ripple-Effekt bei Button-Klick, Dashboard-Struktur erstellt, Datenbank-Update: 'currency'-Datensatz zu Transaktions-Tabelle hinzugefügt
21.04.2021	Dashboard erstellt, BancoMax-Lettering (Schriftzug) geändert
24.04.2021	Transaktions-Timeline zum Dashboard hinzugefügt, kleine Button-Design-Tweaks, Bugfix: LineChart zeigt nun nur noch Ganzzahlen an, Code-CleanUp sämtlicher Klassen durchgeführt
27.04.2021	PDF-Maker hinzugefügt (AccountInfo- & Bezugsbeleg)
28.04.2021	File-Encryption & Decryption für Kartendateien implementiert, FileChooser bei Login & Master hinzugefügt, Icon-Library ins Projekt integriert, kleinere Login- & Master-View Design-Überarbeitungen
29.04.2021	Implementierung eines Sliders für die Notengröße beim Bezug
30.04.2021	Slider von Withdrawal-Klasse zu WithdrawalConfirm-Klasse verschoben
04.05.2021	Vorgehen zum Aufsetzen eines eigenen Servers besprochen und geplant
09.05.2021	Neuen Server zuhause aufgesetzt und eingerichtet, Datenbankverbindung im Programm entsprechend angepasst und Datenbank wieder aufgefüllt
11.05.2021	Datenbankverbindung: Error-dialog wird nun bei einem Verbindungsfehler angezeigt
13.05.2021	Payout-View erstellt
18.05.2021	Euro-Noten Bilder zugeschnitten
22.05.2021	Bugfix: Bearbeitete Euro-Bilder in Projekt eingefügt, um einen Anzeigefehler in der Payout-View zu eliminieren
25.05.2021	Weiterarbeit an Dokumentation (Diagramme überarbeitet und beschrieben)
29.05.2021	Weiterarbeit an Dokumentation (Erster Teil UI-Konzept beschrieben, Journal aktualisiert und überarbeitet; Aufgabenstellung, Projektauswahl und Ziele dokumentiert)
01.06.2021	Logo-Schriftzug überarbeitet, Weiterarbeit an Dokumentation (Exchange-Rate API & Implementierung dokumentiert, zweiter Teil UI-Konzept beschrieben, PDF-Maker & Implementierung dokumentiert)
02.06.2021	GitHub, Weiterarbeit an Dokumentation (Beziehung zur Realität und Design dokumentiert)
03.06.2021	Weiterarbeit an Dokumentation (Reflexion angefangen, UI-Konzeptbeschreibung weitergemacht, Arbeitsaufteilung dokumentiert, SQL-Befehl Erklärungen angefangen)
04.06.2021	An Dokumentation weitergearbeitet, PowerPoint für Schlusspräsentation erstellt
05.06.2021	Weiterarbeit an Dokumentation (Titelblatt erstellt und kleinere Verbesserungen)
08.06.2021	Weiterarbeit an Dokumentation, Vorbereitung Abschlusspräsentation
12.06.2021	Dokumentation fertig gemacht
13.06.2021	<b>Projektgabe um 23:59 Uhr</b>
15.06.2021	<b>Abschlusspräsentation</b>

## 4.2 Herangehensweise

### 4.2.1 Arbeitsaufteilung

Die zeitliche Aufteilung der Arbeit erfolgte gemäss den Vorgaben von Herr Glanzmann, nämlich mindestens 60 Stunden pro Person, was bei einer Dreiergruppe heisst, 180 Stunden im Gesamten. Davon haben wir 40 Stunden in der Schule gearbeitet. Die Arbeitsaufteilung wurde immer schrittweise geplant, denn je nachdem wer bereits mit seiner vorherigen Arbeit fertig war, bekam einen neuen Auftrag. Durch diese agile Vorgehensweise konnte das Team auch sehr flexibel entscheiden, wann etwas neu dazukommen, oder verbessert werden soll. Dies kam uns oft zugute, denn z.B. der Server funktionierte zuerst (wahrscheinlich wegen dem Provider) nicht, weshalb ein anderes Teammitglied dies übernehmen musste. Wir sind froh, dass wir recht früh mit Bereitstellung der Infrastruktur begonnen haben, denn Probleme wie dieses hätten unter Zeitdruck verheerende Auswirkungen haben können, wenn sie zu spät angegangen worden wären. Das Git-Repository eröffnete für unser Team neue Möglichkeiten, somit konnten wir einfacher zusammenarbeiten und zugleich hatten wir die Sicherheit, dass es immer ein Backup gab. Beim Programmieren hat sich das Team abgewechselt, damit man schneller vorankommt und nicht eine Person alles machen muss.

### 4.2.2 Beziehung zur Realität

Unser oberstes Ziel ist es, den Bankautomaten so auszugestalten, dass er einen realen Bankautomaten simuliert. Dazu sind wir am Anfang des Projektes zu dritt an einen Bankautomaten der Raiffeisen-Bank gegangen und haben verschiedene Optionen ausprobiert und zu allem Fotos gemacht. Wir haben uns auch Belege für eine Auszahlung und einen Kontoauszug ausdrucken lassen, damit wir unsere Belege möglichst authentisch aussehen lassen können. Um selbst einen Überblick über das Projekt und die Funktionalitäten zu haben, erstellten wir das UI-Konzept, welches in [Kapitel 2.3](#) genau beschrieben wird, anhand der gemachten Fotos und eigener Ideen. Allerdings störte uns an den realen Automaten neben dem veralteten Design, welches noch in [Kapitel 4.2.3](#) angesprochen wird, noch eine andere Sache. Die Funktionalität war uns schlicht zu wenig vielseitig. Einen Bankautomaten für blosser Auszahlungen, bei denen man nicht einmal die Notengrösse genauer spezifizieren kann, zu implementieren, wäre unserer Meinung nach ein viel zu einfaches Projekt gewesen. Also haben wir Ideen für zahlreiche Erweiterungen gesammelt und in unser UI-Konzept eingebaut. Dabei stand immer im Vordergrund, dass dieser Bankautomat genauso in der realen Welt stehen könnte und somit benutzerfreundlich sein, und sinnvolle, nützliche Funktionen bieten muss. Die kritischen Stellen für die Aufrechterhaltung der Immersion, man bediene gerade einen realen Bankautomaten, waren immer die, an denen der Benutzer in der echten Welt physikalisch mit dem Automaten interagieren müsste. Dies ist zum Beispiel beim Einschieben der Karte, der Ein- und Auszahlung von Geld und beim Ausdrucken von Belegen der Fall. Im Folgenden beschreiben wir kurz, wie wir die Realitätsnähe in diesen besonderen Fällen umgesetzt haben.

Beim Einschieben der Karte wird in Wirklichkeit der Chip ausgelesen, und man muss nur noch die PIN und keine Kartennummer eingeben. Dies haben wir so umgesetzt, dass man nur beim ersten Login manuell die Kartennummer vollständig eingeben muss. Dann kann man sich in der Master-View eine sogenannte Karten-Datei an einem benutzerdefinierten Ort abspeichern lassen. Diese Datei wird als .txt-File erstellt und mit einer verschlüsselten, unleserlichen Zeichenfolge beschrieben (Die Implementierung dazu finden Sie im [Kapitel 5.7.2](#)) Wenn beim nächsten Login, bei dem alternativ auch erneut die Kartennummer eingegeben werden könnte, die Kartendatei angewählt wird, liest das Programm den Zeichencode aus und entschlüsselt ihn. Wenn man die Datei nicht verändert hat und der Schlüssel gültig ist, wird nun automatisch das Feld mit der Kartennummer ausgefüllt. Dies funktioniert immer, unabhängig davon, wie lange die Nummer ist.

Bei einer Einzahlung werden dem Benutzer die auszahlbaren Noten in der gewählten Währung als Bilder angezeigt, er kann entweder die Anzahl eingeben, oder einfach auf eine Note klicken, dann wird die gewünschte Anzahl dieser Note um eins erhöht. Dieser Vorgang ist unser Weg, das physikalische Einschleichen von Noten zu simulieren.

Bei Auszahlungen wird eine Payout-View angezeigt. In diesem Fenster sind sämtliche ausgezahlten Noten als Bilder sichtbar, doppelte Noten werden auch mehrfach angezeigt, sodass es ziemlich genau die Ausgabe von Noten in der realen Welt widerspiegelt.

Wenn man die Ausgabe eines Beleges wünscht, wird in dem Verzeichnis, in dem das Programm abgespeichert ist, ein PDF-File mit dem Beleg erstellt. Zusätzlich wird dieses File auch automatisch in der gewählten Windows Standard-App für .pdf-Dateien geöffnet und ist somit einsehbar. Wie die Belege aussehen, wird im [Kapitel 4.3.4](#) beschrieben.

### 4.2.3 Design

Es ist uns ein Anliegen mit hoher Priorität, das Design ansprechend, professionell und übersichtlich zu gestalten. Das Design der realen Bankautomaten, die wir aufgesucht haben, hat uns allerdings nicht wirklich angesprochen und wirkte recht veraltet. Also haben wir zuerst im Team abgesprochen, welche Farben wir für das Programm benutzen wollten. Wir haben uns auf ein dunkles Design mit Akzentfarben von Rot



bis Gelb entschieden. Um dauerhaft auf die Hex-Werte der verschiedenen ausgewählten Farben zugreifen zu können, haben wir uns ein Farbschema auf einer [Webseite](#) zusammengestellt. Das sieht dann wie folgt aus:

Die Erstellung dieses Farbschemas hat sich als äusserst praktisch erwiesen, da wir so ein einheitliches Farbdesign einhalten, und die Hex-Codes einfach rauskopieren konnten.

Im nächsten Schritt wollten wir unserer imaginären Firma «BancoMax» eine richtige Identität schaffen. Dazu benötigt sie ein eigenes Logo und einen Schriftzug. Diese haben wir in Photoshop erstellt.

Im Folgenden sieht man die erste Version vom Logo und Schriftzug.



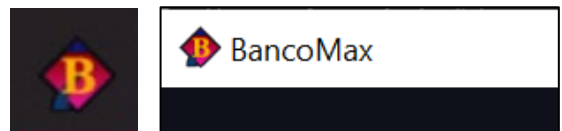
Mit diesen Designs waren wir allerdings nach einer Weile nicht mehr zufrieden. Zu dieser Zeit standen wir allerdings unter schulischem Prüfungsdruck und wir halten uns alle für nicht besonders künstlerisch begabt. Deshalb hat Manuel seinen Bruder, der in der Freizeit gerne Designs erstellt, gefragt, ob er Lust habe ein Logo für unsere Firma zu zeichnen und sich dabei nach dem Farbschema zu richten. Er hat sich über den Vorschlag gefreut und uns ein Logo erstellt, mit dem wir sehr zufrieden waren. Wir wollen an dieser Stelle nochmals explizit einen Dank an Fabian Schmid aussprechen, der uns das Logo-Design kostenlos zur freien Verfügung gestellt hat. Passend zum Logo haben wir in Photoshop noch einen Firmenschriftzug in ähnlichen Farben erstellt. In der unteren Grafik sind das neue Logo und der Schriftzug auf der Hintergrundfarbe des Programmes, die, unserer Meinung nach, die Farben sehr gut betont und einen edlen Eindruck vermittelt, abgebildet.





Die Entscheidung dem Programm einen dunklen Hintergrund zu geben, haben wir ebenfalls im Team gefällt. Wir wollen uns damit von den traditionellen hell thematisierten Programmen hervorheben und neben der Ästhetik finden wir es auch wesentlich Angenehmer für die Augen, wenn man lange auf den Bildschirm schauen muss. Ein völlig schwarzer Hintergrund gefiel uns allerdings nicht und so entschieden wir uns für diesen sehr dunklen Blauton. Wir sind im Ganzen sehr zufrieden mit dem Stand des Designs und der Firmenidentität.

Wir haben auch darauf geachtet, dass in der Taskleiste und im Fensterrahmen das korrekte Logo angezeigt wird.



#### 4.2.4 JavaFX

Da wir uns vorgenommen haben, das Design so ansprechend wie möglich zu gestalten, haben wir uns entschieden, für dieses Projekt das JavaFX-Framework und nicht Swing zu benutzen. Der Aufbau von JavaFX-Applikationen ist grundsätzlich sehr ähnlich zu dem von Webseiten. Im Controller (.java) wird die Funktionalität programmiert, im FXML-File wird die Struktur und die Formatierung festgelegt und optional

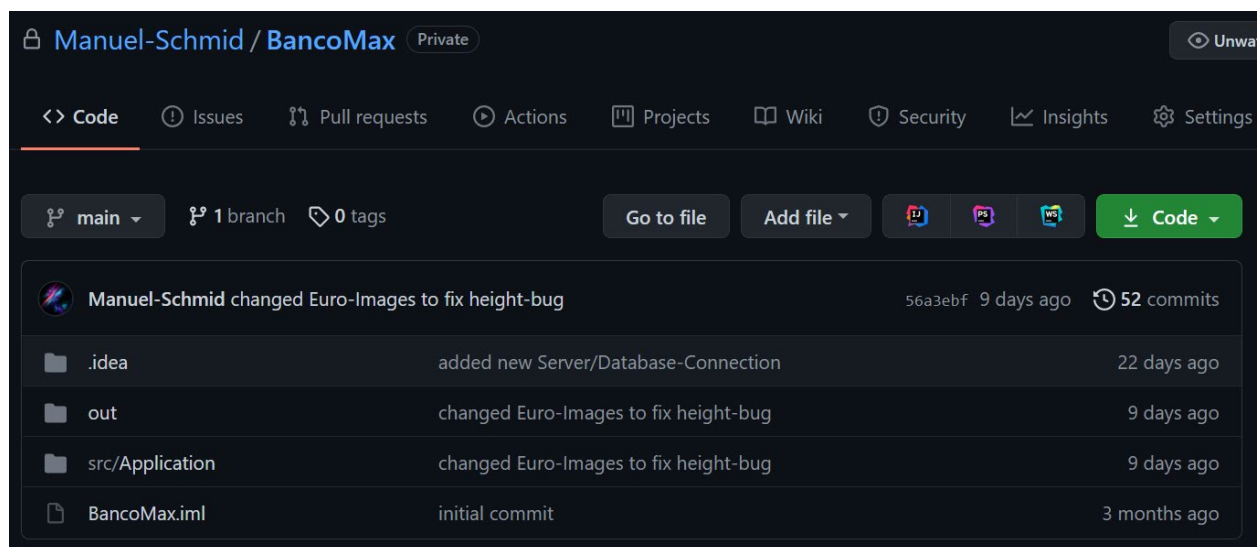
kann auch noch ein Stylesheet (CSS) hinzugefügt werden. Dieses neue Framework zu lernen war herausfordernd, aber auch spannend. Die Sprache FXML gleicht HTML im Allgemeinen jedoch sehr und so fanden wir uns auch dort schnell zurecht.

#### 4.2.5 GitHub

Unser GitHub Repository finden Sie [hier](#).

##### Übersicht

Dies ist das Dashboard des GitHubs von unserem Bancomaten Projekt. Hier kann man Verschiedenstes anschauen, z.B. oben die Reiter, Insights, Issues oder eine Pull Request erstellen. Zudem wird direkt auch angezeigt, wie viele Branches das Projekt momentan besitzt. Der Nutzer kann, wenn er will, den Code herunterladen oder direkt in auf der Website einsehen. Unten im Feld sind alle Dateien und Verzeichnisse, welche im Laufe der Zeit hinzugefügt wurden.

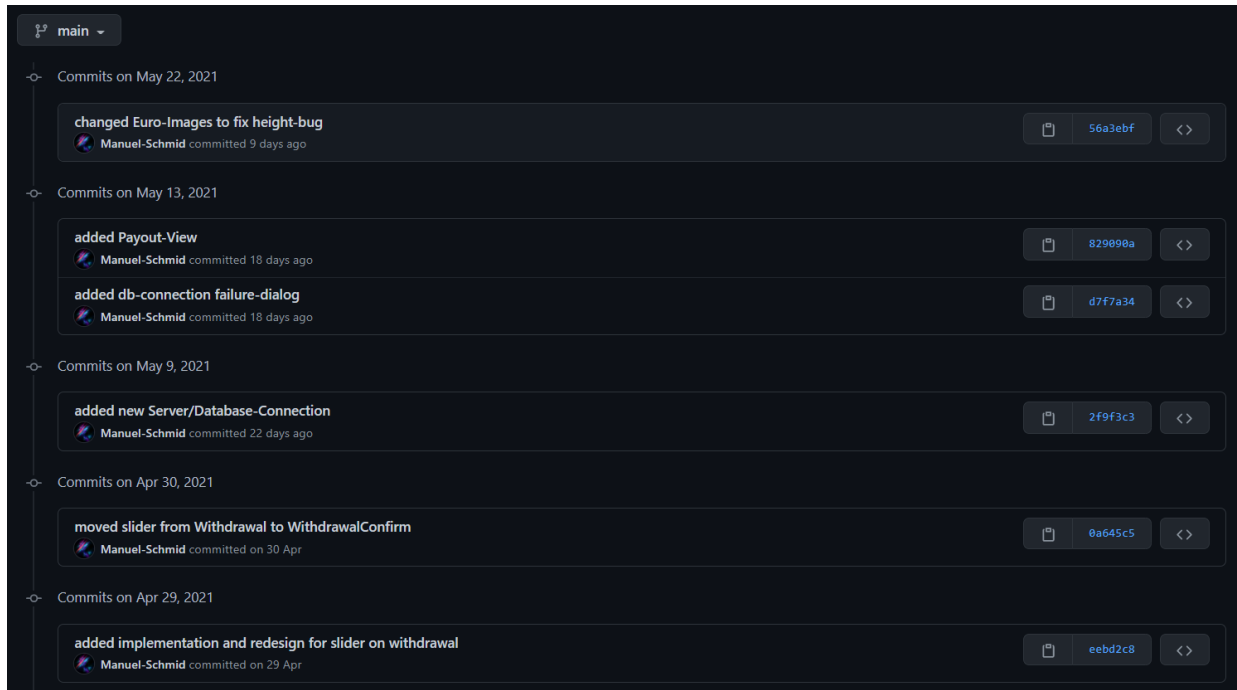


##### Commits

Wenn man in den Brach «Main» wechselt, sieht man alle Commits, welche getätigt wurden. Die Commits werden in der Kommandozeile mit folgenden Befehlen erstellt:

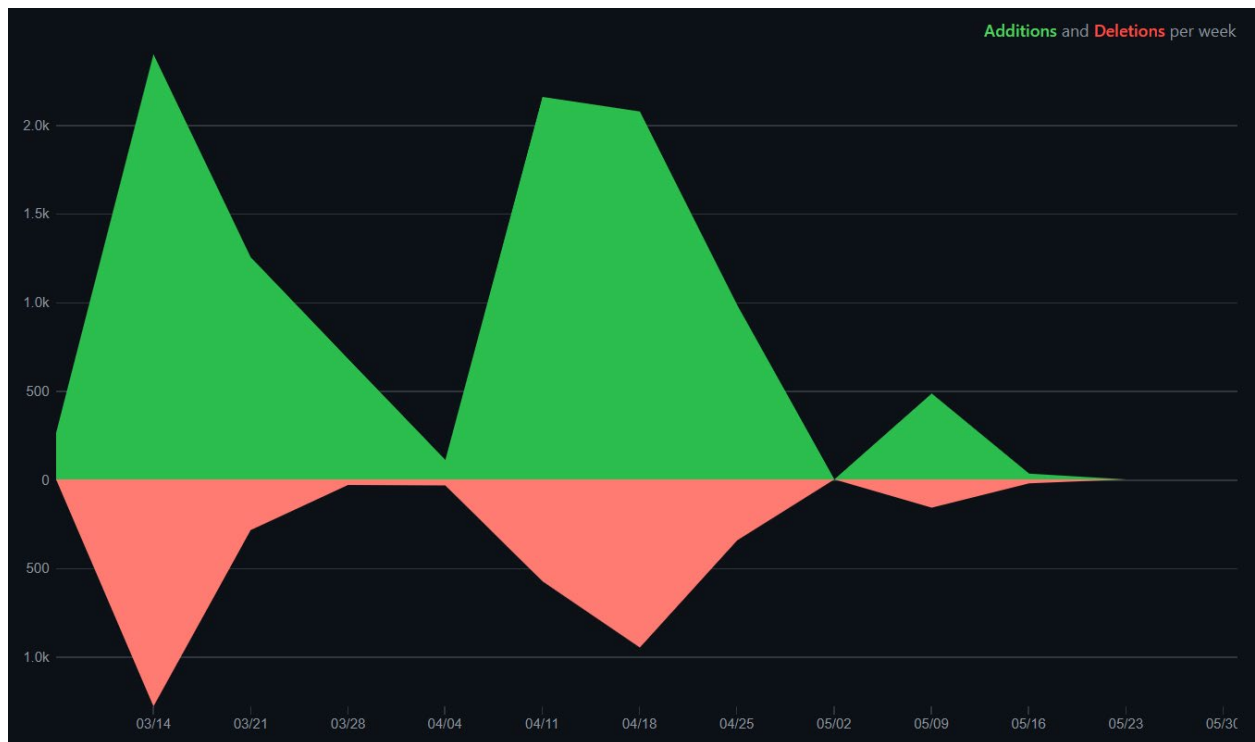
```
git add .
git commit -m "commit message"
git push --set-upstream origin BranchName
```

«git add .» schaut, welche Änderungen vorgenommen wurden. «git commit-m...» Verpackt die Änderungen in einem Paket, welches mit einer Nachricht versehen wird. «git push» übergibt schlussendlich dieses Paket dem Branch. Erst wenn dieser Befehl ausgeführt wird, wird der Code in das Repository hochgeladen. «--set-upstream origin BranchName» muss nur gemacht werden, wenn man sich in einem anderen Branch ausser Main befindet. Im folgenden Bild sieht man ein paar unserer Com-mits.



### Additions and Deletions

Github erstellt zu jedem Projekt automatisch einen Chart. Der Chart, welcher unten zu sehen ist, kann vom Dashboard unter Insights, dann unter Commits angeschaut werden. Dieser Chart zeigt, wie oft pro Woche neuer Code hinzugefügt und gelöscht wurde. Das «Löschen» ist nicht direkt als löschen zu verstehen, denn auch wenn man Refactoring betreibt, löscht man sehr oft Linien, deshalb ist die «Löschen-Kurve» so gross.



#### 4.2.6 Datenbank

Die Datenbank haben wir auf unserem Server ([Kapitel 4.3.1](#)) eingerichtet und mit Hilfe der «MySQL Workbench» mit Daten befüllt. Die wiederverwendbaren Queries und wichtigen Befehle haben wir lokal abgespeichert, um sie nicht immer neu eingeben zu müssen. Das Datenbankdiagramm konnten wir auch in diesem Programm erstellen und anpassen. Wir sind sehr von dieser Applikation überzeugt und würden sie bei einem weiteren Projekt sofort wieder benutzen.

#### 4.2.7 Sicherheit

Es war unser Ziel, die Sicherheit wo immer es ging, möglichst gut zu gewährleisten. Passwörter für den Server, die Datenbank und den Administrator haben wir absichtlich nicht zu kurz und einfach gewählt. Wir wollten die Gelegenheit dieses Projekts auch nutzen, die Speicherung von Daten in der Datenbank sicherer zu implementieren. Es ist für uns ein No-Go, Passwörter in Klartext in der Datenbank zu speichern, auch wenn dies natürlich wesentlich einfacher wäre. So haben wir Recherche betrieben und uns entschieden, den Hashing-Algorithmus SHA zu implementieren, da die Einweg-Verschlüsselungstechnologie, die bei Hashing zur Anwendung kommt, unserer Meinung sehr clever ist. So wissen nicht einmal wir als Datenbankadministratoren, welche Passwörter unsere Bankkunden verwenden. Beim Verschlüsseln der Kartendateien für das Login kommt ebenfalls ein Verschlüsselungsalgorithmus zum Einsatz, diesmal aber einer, der rückgängig gemacht werden kann, da die Kartenummer ja wieder ausgelesen werden muss. Die konkrete Implementierung dieser Security-Algorithmen wird im [Kapitel 5.7](#) beschrieben.

### 4.3 Herausforderungen und Ereignisse

#### 4.3.1 Server

Anfangs haben wir es versucht eine Datenbank mithilfe eines Raspberry Pi's zu erstellen, dazu haben wir Verschiedenes ausprobiert. Anfangs luden wir Raspberry Pi OS herunter. Dort haben wir MySQL, Apache und MariaDB heruntergeladen, damit wir einen Datenbankclient haben. Mit MySQL, welches wir auch installierten, war es uns möglich die Datenbank zu bearbeiten und verwalten. Mit Raspberry Pi OS hat dies nicht wirklich funktioniert, weshalb wir den Raspberry Pi nochmals aufsetzten und Ubuntu darauf installierten. Dort haben wir wieder das genau gleiche gemacht, diesmal funktionierte es glücklicherweise. Damit wir darauf zugreifen konnten, öffneten wir den Port von SSH (22), Apache (80) und MariaDB (3306), damit wir mit SSH den Raspberry konfigurieren können und auf die Datenbank zugreifen können. Damit der Raspberry Pi ein wenig besser geschützt ist, haben wir einerseits sehr starke Passwörter verwendet und zudem auch eine Firewall, Fail2Ban, installiert. Diese haben wir aber nicht richtig zum Laufen gekriegt, weshalb wir sie daraufhin wieder deinstallierten. Als wir die Verbindung über ein phpMyAdmin-Interface getestet haben, hat alles wunderbar funktioniert. Der Versuch, dies im Code zu integrieren, war leider


nicht erfolgreich, da der Port nicht direkt adressiert werden konnte. Wir haben lange an diesem Problem gegrübelt, jedoch keine Lösung gefunden.

Wir haben uns dann entschieden, einen alten PC, den einer von uns noch zu Hause hatte, zu einem Server für die Datenbank umzufunktionieren. Als erstes haben wir recherchiert, welches Betriebssystem für den Server am besten wäre. Wir haben uns für Ubuntu Server entschieden und den Release 20.04 gewählt, da dies die neueste LTS-Version ist. Das Aufsetzen des Servers war recht ungewohnt und auch herausfordernd, da Ubuntu Server keinerlei Benutzeroberfläche bietet, sondern vollständig durch die Konsole bedient werden muss. Wir haben den PC neu aufgesetzt und mit dem neuen OS gebootet. Dann haben wir MySQL und MariaDB eingerichtet und dem Server eine statische IP-Adresse zugewiesen. Schliesslich mussten wir noch für den SSH-Verbindungs-Port zum Server (22) und den Standard MySQL Port (3306) im Router-Menü eine Port-Weiterleitung konfigurieren, damit sie auch von ausserhalb des eigenen Netzwerks ansprechbar sind. Die SSH-Verbindung wäre theoretisch nicht notwendig gewesen, da der Server ja auch lokal bedient werden kann. Wir haben uns aber doch für SSH entschieden, da wir so auch von der Schule aus auf den Server zugreifen, und die Datenbank nötigenfalls umstrukturieren können. Zudem konnten wir so auch diese äusserst nützliche Technologie kennenlernen. Nun war es an der Zeit, eine Datenbank und einen Benutzer zu erstellen. Was wir auch getan haben. Der Datenbankbenutzer hat ausschliesslich Zugriff auf die BancoMax-Datenbank und sein Passwort ist 16-Zeichen lang mit Gross-/Kleinschreibung und Sonderzeichen. Wir wollen an dieser Stelle noch anmerken, dass die Verbindung zur Datenbank und somit auch das Programm selbst nur dann funktioniert, wenn im Netzwerk des Benutzers ausgehende Verbindungen mit diesen Ports erlaubt sind. Auf die Implementierung der JDBC-Datenbankverbindung gehen wir in [Kapitel 5.4](#) noch genauer ein.

#### 4.3.2 Exchange-Rate API

Wir haben uns entschieden, dass wir die Kann-Anforderung «Fremdwährungen sind immer aktuell (API)» umsetzen wollen. Als Fremdwährung soll allerdings der Euro zur Verfügung stehen. Bevor wir mit Berechnungen und spezifischen Fenstern für Euro-Transaktionen anfangen konnten, mussten wir einen Weg finden, den Eurokurs zuverlässig und ganz aktuell abfragen zu können. Dafür verwenden wir eine API, von der man bis zu 1000 Abfragen im Monat kostenlos tätigen kann. Für den Umfang und Zweck unseres Projekts sind wir der Meinung, dass diese Anzahl fürs erste durchaus ausreicht. Die Webseite, die wir für die API benutzen, heisst [Exchangeratesapi.io](https://exchangeratesapi.io).

Auf der Webseite kann man nachschauen, wie viele Abfragen diesen Monat schon getätigt wurden und welchen prozentualen Anteil diese vom maximalen Gratis-Limit ausmachen. Der Wechselkurs wird jede Stunde aktualisiert und ist so immer ziemlich aktuell. Dafür dass die API

 **Exchangeratesapi.io**

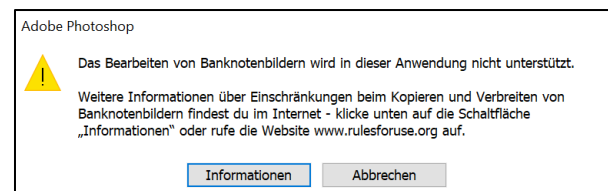
Your subscription: Free Plan

<b>Subscription:</b>	Free Plan
<b>Billing period:</b>	2021-06-01 - 2021-06-30
<b>API Usage:</b>	0% (0 / 1,000) <a href="#">API Usage</a>

kostenlos ist, waren wir positiv überrascht, wie problemlos die Abfragen und die monatliche Zurücksetzung der Abfragen funktioniert hat. Die genaue Implementierung der API-Abfragen ist im [Kapitel 5.3](#) beschrieben.

### 4.3.3 Banknoten-Bilder

Damit wir den Prozess so wahrheitsgetreu abbilden können, haben wir uns überlegt, an den Nutzer Bilder von Banknoten auszugeben. Dazu haben wir von den Euro und Schweizer Franken die notwendigen Noten im Internet gesucht. Dies war ziemlich schwierig, da die meisten Bilder von der Auflösung unbrauchbar waren oder ein Wasserzeichen beinhalteten. Schlussendlich haben wir aber alle gefunden. Nun gab es noch ein Problem, alle hatten verschiedene Grössen, was wir zuerst mithilfe von Photoshop versucht haben zu beheben. Jedoch, als wir die Bilder öffneten, wurden wir von einer wunderbaren Nachricht begrüsst, dass das Bearbeiten von Noten nicht unterstützt werde. Also mussten wir uns auf die Suche nach einem anderen Bildbearbeitungsprogramm machen. Deshalb haben wir GIMP gewählt. Nun haben wir jedes Bild auf die exakt gleichen Masse skaliert.



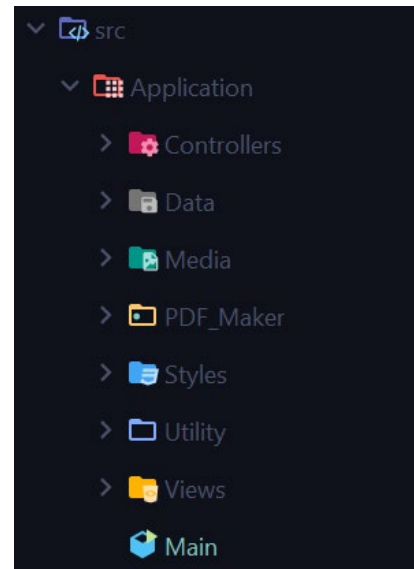
### 4.3.4 PDF-Maker

Wir hatten von Anfang an die Idee, dass der Automat dem Nutzer einen Informationsbeleg und einen Auszahlungsbeleg ausgibt. Darum haben wir einen PDF-Maker erstellt, welcher genau dies macht. Dazu haben wir nach geeigneten Tutorials im Internet gesucht, jedoch war nichts Inhaltlich sehr wertvoll. Jedoch gab uns diese Videos die Idee, die gleiche Bibliothek zu finden. Also haben wir darauf ITextPDF 5 heruntergeladen. Dies ist eine sogenannte Legacy-Version, da diese nicht mehr unterstützt wird, jedoch durch das gratis verfügbar ist. Das Formatieren war nicht schwierig, jedoch hätte man dies viel eleganter gestalten können, doch dann hätte man viel mehr Zeit in das Lesen der Dokumentation der Bibliothek stecken müssen, um die notwendigen Syntax-Angaben zu finden. Das Erstellen funktionierte jetzt, jedoch wollten wir, dass wenn es bereits eine Datei von diesem Namen gab die Alte nicht einfach überschrieben wird, sondern dass ein Zähler in dem Namen nach oben zählt und dann die neue mit diesem neuen Zähler benannt. Dies war das schwierigste an diesem Maker, da wir noch alle Dateien im Speicherverzeichnis des Programmes durchgehen müssen, um zu schauen ob bereits eines existiert. Zudem haben wir noch eingebaut, dass dieses erstellte PDF direkt danach in dem Standard PDF-Programm geöffnet wird. Der gesamte Code vom PDF-Maker kann im [Kapitel 5.2](#) angeschaut werden.

## 5 Programmcode

### 5.1 Projekt-Struktur

In unserem Source-Folder haben wir einen Ordner Application erstellt, in dem wiederum die Main-Klasse und sämtliche Packages des Projekts gespeichert sind. `Controllers`, `Styles` und `Views` sind für JavaFX und werden im entsprechenden Kapitel ([4.2.4](#)) noch genauer behandelt. Im Package `Data` sind Klassen zur temporären Informationsspeicherung, die `Database`-Klasse, in der alle SQL-Abfragen implementiert sind und die Klasse zur API-Einbindung für den Wechselkurs. `Media` beinhaltet alle Bilder, die im Projekt verwendet werden, dazu gehören unter anderem unser Logo, unser Firmenschriftzug und die Bilder der CHF und Euro-Noten. Die meisten dieser Bilder sind als .png-Dateien gespeichert, und haben einen transparenten Hintergrund. Das `PDF_Maker`-Package enthält zwei Klassen, die zur Erstellung von PDF-Belegen benötigt werden (Siehe [Kapitel 4.3.4](#)). Dem `Utility`-Package haben wir das [Kapitel 5.6](#) gewidmet.





## 5.2 Implementierung PDF-Maker

Die Schlüssel Methode ist die `createWithdrawInfoDeposit()`. Sie regelt den gesamten Inhalt der PDF's. Mit einer Auswahl, wird entschieden welche Informationen gebraucht werden. Diese Informationen werden mithilfe von verschiedensten Paragraphen und auch einer Tabelle dargestellt. Die Daten werden direkt aus der Datenbank übergeben.

Einer der Wichtigsten Methoden in dieser Klasse ist die `fileAlreadyExistsReader()`. Diese Methode wird aufgerufen, bevor eine neue Datei erstellt wird, denn sie geht durch alle Namen der Dateien, welche sich am Speicherort des Programmes befinden. Somit ist es möglich herauszufinden, ob bereits ein Auszahlungsschein oder Informationsbeleg ausgegeben wurde. Dies ist wichtig, da ansonsten die bereits erstellte Datei einfach überschrieben wird, da sie den gleichen Namen tragen. Damit dies gelöst wird, wird verglichen, ob der übergebene String in irgendeiner anderen Datei vorkommt, falls ja, wird zuerst geschaut welche Zahl dort steht, diese wird dann einfach um eins erhöht. Dieser Filter wird mithilfe einer Regex umgesetzt.

```
public void fileAlreadyExistsReader() {
    try {
        List<File> filesInFolder;
        filesInFolder = Files.walk(Paths.get(System.getProperty("user.dir")))
            .filter(Files::isRegularFile)
            .map(Path::toFile)
            .collect(Collectors.toList());
        for (File iterable_element : filesInFolder) {
            if (iterable_element.getName().contains(fileName)) {
                if (isWithdraw) {
                    counter_for_files_W = Integer.parseInt(iterable_element.getName().replaceAll(regex, "[^0-9]"),
                        counter_for_files_W += 1;
                } else {
                    counter_for_files_I = Integer.parseInt(iterable_element.getName().replaceAll(regex, "[^0-9]"),
                        counter_for_files_I += 1;
                }
            }
        }
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Dies ist die `createPDF()` Methode. Sie sorgt dafür, dass der Name der aktuellen Datei angepasst wird, sie setzt ihn aus dem, als Parameter übergebenen Namen (übergeben an eine andere Methode), und dem Zähler, welcher in der `fileAlreadyExistsReader()` stetig angepasst wird. Anschliessend wird einfach die Dateiendung `.pdf` angefügt, was aus dieser Datei ein PDF macht.

```
private void createPDF() {
    try {
        if (isWithdraw) {
            fileNameFull = fileName + "_" + counter_for_files_W + ".pdf";
        } else {
            fileNameFull = fileName + "_" + counter_for_files_I + ".pdf";
        }
        document = new Document();
        PdfWriter.getInstance(document, new FileOutputStream(fileNameFull));
    } catch (FileNotFoundException | DocumentException e) {
        e.printStackTrace();
    }
}
```

Die `OpenPDF` Klasse, wird im PDF-Maker aufgerufen, sobald die gewünschte PDF-Datei erstellt wurde. Zuerst wird geprüft, ob der Desktop, auf welchem das Programm läuft, unterstützt wird. Mit der `getDesktop` wird eine Instanz des Desktops, zum Öffnen des spezifischen Pfades, genutzt.

```
public class OpenPDF {

    public OpenPDF(PDFFile f) {
        try {
            File pdfFile = PDFFile.getFileName();
            if (pdfFile.exists()) {
                if (Desktop.isDesktopSupported()) {
                    Desktop.getDesktop().open(pdfFile);
                } else {
                    System.out.println("Awt Desktop is not supported!");
                }
            } else {
                System.out.println("File does not exist");
            }
            System.out.println("Done");
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

### 5.3 Implementierung Exchange-Rate API

Die Methode `getExRate()` gibt einen `double` als Rückgabewert zurück, dieser Rückgabewert soll der aktuelle Wechselkurs sein. An die Daten der API gelangen wir mit einer GET-Request an die Https-Adresse der entsprechenden API, das Resultat wird in ein JSON-Objekt umgewandelt, für das wir extra eine Bibliothek ins Projekt einbauen mussten. Aus diesem Objekt wird dann genau der Wert herausgelesen, der uns interessiert. Nämlich der Wechselkurs zwischen Euro und CHF. Wie im Code zu sehen ist, haben wir allfällige Fehlermeldungen abgefangen und geben dann einen Kurs von 1.11, da dieser momentan aktuell ist, zurück. Dies ist natürlich nicht ideal und falls wir dieses Projekt kommerziell vertreiben würden, wären wir auch bereit, die API-Kosten zu bezahlen. Da es sich aber «nur» um ein Schulprojekt handelt, geben wir, für den äusserst unwahrscheinlichen Fall, dass alle 1000 kostenlosen Abfragen in einem Monat verbraucht würden, diesen Pauschalkurs von 1.11 zurück, damit das Programm trotzdem weiterlaufen kann, ohne das alles einfriert oder abstürzt.

```
public static Double getExRate() {
    try {
        StringBuilder res = new StringBuilder();
        URL url = new URL( spec: "https://api.exchangeratesapi.io/v1/latest?access_key=b11c07a5c6d7fde5ea8c69e1f0f8676f&symbol");
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        try (var reader = new BufferedReader(new InputStreamReader(conn.getInputStream()))) {
            for (String line; (line = reader.readLine()) != null; ) {
                res.append(line);
            }
            String result = String.valueOf(res);
            JSONObject obj = new JSONObject(result);
            return obj.getJSONObject("rates").getDouble( key: "CHF");
        }
    } catch (Exception e) {
        System.out.println("Aktueller Kurs kann nicht abgerufen werden, es wird mit einem CHF-Euro Kurs von 1.11 gerechnet.");
        return 1.11;
    }
}
```

In diesem Screenshot ist ein Beispieldatensatz aus der API zu sehen, den wir mit dem Link abgefragt haben. Ein JSON-Objekt mit den angeforderten Daten wird an unser Programm zurückgeschickt, und den erhaltenen Wechselkurs können wir dann für diverse Umrechnungen verwenden.

```
{
  "success": true,
  "timestamp": 1622555044,
  "base": "EUR",
  "date": "2021-06-01",
  "rates": {
    "CHF": 1.098173,
    "EUR": 1
  }
}
```

## 5.4 Datenbankverbindung

In der `Database`-Klasse wird in der `connectToDatabase()`-Methode zuerst die JDBC-URL in eine Konstante gespeichert, in der die öffentliche IP-Adresse des Servernetzwerks, der MySQL Port und die Datenbank angegeben sind. Dann werden auch noch der Benutzername und das Passwort gespeichert. Diese Angaben werden nach der Registrierung des JDBC Treibers benutzt, um eine Verbindung zur Datenbank herzustellen.

```
public static void connectToDatabase() throws ClassNotFoundException, SQLException {  
    final String jdbcURL = "jdbc:mysql://[REDACTED]:3306/bancomax";  
    final String username = "BancoAdmin";  
    final String password = "[REDACTED]";  
    System.out.println("Connecting to database ...");  
    Class.forName("com.mysql.cj.jdbc.Driver"); // Register JDBC Driver  
    conn = DriverManager.getConnection(jdbcURL, username, password);  
    System.out.println(ConsoleColors.GREEN + "Connection successful!" + ConsoleColors.RESET);  
}
```

## 5.5 Interaktion mit der Datenbank

In den folgenden drei Unterkapiteln werden ein paar wichtige Methoden beschrieben, an denen die Technische Umsetzung von Interaktionen mit der Datenbank gut erkennbar ist.

### 5.5.1 Beispiel: SELECT-Befehl

Die Methode `getBalance()` befindet sich in der Klasse `Database`. Als Parameter wird die ID des Benutzers übergeben, von dem man den Kontostand haben möchte. Die Variable `query` wird mit dem SQL-Befehl beschrieben, der oder mehrere Datensätze von Datenbank anfordert. Nun wird die Methode `createStatement()` ausgeführt. Die aus der Abfrage resultierenden Datensätze werden als `Double` ausgelesen und als `return`-Wert zurückgegeben. Falls bei der Verbindung zur Datenbank, oder beim Umwandeln des Resultats ein Fehler entsteht, wird dieser auf die Konsole ausgegeben.

```
public static Double getBalance(int accountID) {
    String query = "SELECT a.balanceInCHF FROM bancomax.card as c JOIN account as a ON c.FK_accountID = a.accountID WHERE a.accountID = '"+accountID+"'";
    try (Statement stmt = conn.createStatement()) {
        ResultSet rs = stmt.executeQuery(query);
        if (rs.next()) {
            return rs.getDouble(1);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

### 5.5.2 Beispiel: INSERT-Befehl

Die Methode `insertUser()` fügt einen neuen Datensatz in die Tabelle `user` in der Datenbank ein. Informationen über den gewünschten Benutzer werden durch Übergabeparameter mitgeteilt und mit dem MySQL `INSERT`-Befehl in die Datenbank übertragen.

```
public static void insertUser(String firstName, String lastName, Salutation salutation) {
    try {
        String query = "INSERT INTO `bancomax`.`user` (`firstName`, `lastName`, `salutation`) " +
            "VALUES ('"+firstName+"', '"+lastName+"', '"+salutation+"')";
        PreparedStatement pstmt = conn.prepareStatement(query);
        pstmt.execute();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

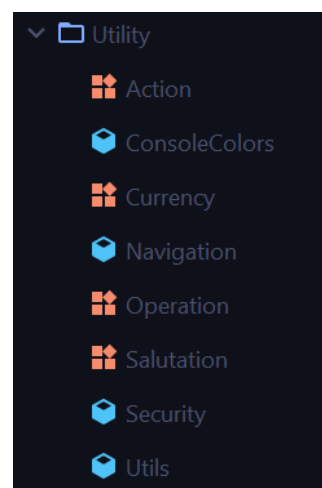
### 5.5.3 Beispiel: UPDATE-Befehl

```
public static void updateBalance(Operation operation, double amountInCHF, int accountID) {  
    try {  
        double newBalance;  
        if (operation == Operation.deposit) newBalance = getBalance(accountID) + amountInCHF; // Deposit  
        else newBalance = getBalance(accountID) - amountInCHF; // Withdraw  
        String query = "UPDATE bancomax.account SET balanceInCHF = '" + newBalance + "' " +  
            "WHERE `accountID` = '" + accountID + "';";  
        PreparedStatement pstmt = conn.prepareStatement(query);  
        pstmt.execute();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

Die Methode `updateBalance()` hat drei Übergabeparameter. Diese sind folgende: `operation`, diese gibt an, ob es eine Ein- oder Auszahlung ist; `amountInCHF` gibt den ein-/ausbezahlten Betrag an; schlussendlich die `accountID`, die für die Datensätze des Kunden zuständig ist. Zuerst wird abgefragt, ob die Operation eine Ein- oder Auszahlung ist, je nachdem wird entweder zum Kontostand des Kunden den Betrag addiert/subtrahiert. Nun wird der UPDATE-Befehl für die Datenbank mit den neuen Datensätzen vorbereitet und anschliessend ausgeführt. Falls bei der Verbindung zur Datenbank ein Fehler passiert, wird dieser auf die Konsole ausgegeben.

## 5.6 Utility

Das Package `Utility` dient uns als eine Art Werkzeugkiste, darin haben wir verschiedene Klassen erstellt, die über das Projekt hinweg ganz oft an verschiedenen Orten verwendet werden. Ein Beispiel dafür sind die Enum-Typen für `Action`, `Salutation`, `Currency` und `Operation`, die vor allem zur Typensicherheit bei Übergabeparametern verwendet werden. Was sie genau machen wird im Folgenden am Beispiel von `Salutation` erklärt. Ebenfalls werden zwei nützliche Methoden aus der `Utils`-Klasse erklärt.



### 5.6.1 Enum-Beispiel: Salutation

Der Typ `Salutation` ist ein von uns erstelltes `Enum`, welches entweder «Herr» oder «Frau» sein kann. Solch eine Information könnte auch in einem `String` gespeichert werden, dies wäre aber bei Vergleichsabfragen wesentlich langsamer und würde unnötigen Speicherplatz und Ressourcen verbrauchen. Da es nur zwei Werte sind könnte auch ein `boolean` verwendet werden, dies würde allerdings eindeutig gegen die Clean-Code-Prinzipien verstossen, und wäre nur schwer lesbar. Aus diesen Gründen haben wir uns entschieden, ein `Enum` zu verwenden.

```
public enum Salutation {  
    Herr, Frau  
}
```

### 5.6.2 isNumeric()-Methode

In der `Utils`-Klasse sind diverse Funktionen implementiert, die eine Anwendung in verschiedenen Klassen finden. Diese Methoden sind `static` deklariert. Eine davon ist die `isNumeric()`-Methode.

Diese überprüft durch eine `null`-Abfrage und den Versuch, die Übergabe in einen `Double` zu konvertieren, ob es sich

```
public static boolean isNumeric(String strNum) {  
    if (strNum == null) {  
        return false;  
    }  
    try {  
        Double number = Double.parseDouble(strNum);  
    } catch (Exception e) {  
        return false;  
    }  
    return true;  
}
```

beim übergebenen `String` um eine gültige Zahl handelt. Dementsprechend wird dann `true` oder `false` zurückgegeben.

### 5.6.3 formatMoney()-Methode

Diese Methode gibt in Form eines `String`s eine formatierte Zahl zurück. Als Beispiel: Wenn der Input, welcher als `double` Parameter übergeben wird, folgendermassen aussieht: 42000.1234, gibt diese Methode die gleiche Zahl als `String` mit Apostrophen und Dezimalpunkt formatiert zurück. Die Rückgabe würde folgendermassen aussehen: 42'000.12

```
public static String formatMoney(double num) {  
    DecimalFormat formatter = new DecimalFormat( pattern: "#,###.00");  
    return formatter.format(num);  
}
```

## 5.7 Security: Implementierung

### 5.7.1 Hashing

Um die Sicherheit zu erhöhen, wenden wir nicht nur die PIN, sondern auch einen `Salt` dazu in der Hash-Funktion an, dieser macht Brute-Force Attacken beinahe Unmöglich, ausser man hat bereits Zugriff auf die Datenbank. Für jeden Benutzer wird beim Erstellen seines Kontos ein `Salt` abgespeichert, dieser wird jeweils aus der Datenbank ausgelesen und bei einem neuen Login zusammen mit der eingegebenen PIN in den Hashing-Algorithmus geworfen, um herauszufinden, ob die richtige PIN eingegeben wurde. Der `Salt` wird mit dem Code rechts erstellt.

```
public static byte[] createSalt() {  
    SecureRandom random = new SecureRandom();  
    byte[] salt = new byte[10];  
    random.nextBytes(salt);  
    return salt;  
}
```

Unten ist noch die Implementierung der `hash()`-Methode zu sehen. Darin werden die eingegebene PIN und der `Salt` aus der Datenbank in den SHA-Algorithmus mit 50'000 Durchläufen und einer Schlüssellänge von 50 gegeben. Der zurückgegebene `Byte-Array` kann dann mit dem aus der Datenbank abgeglichen werden, um die Korrektheit der PIN zu überprüfen.

```
public static byte[] hash(String toBeHashed, byte[] salt) throws NoSuchAlgorithmException, InvalidKeySpecException {  
    KeySpec spec = new PBEKeySpec(toBeHashed.toCharArray(), salt, iterationCount: 50000, keyLength: 50);  
    SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");  
    return factory.generateSecret(spec).getEncoded();  
}
```

### 5.7.2 Kartennummer-Verschlüsselung

Die `encrypt()`-Methode verschlüsselt einen übergebenen String, in diesem Anwendungsfall die Kartennummer, auf eine rückgängig-machbare Weise. Dazu werden die Klassen `Key` und `Cipher` verwendet und mehrere Schritte zur «Verunstaltung» des Wertes durchgeführt. Bei der Entschlüsselung (Decryption) werden alle diese Schritte in umgekehrter Reihenfolge abgearbeitet, um wieder den ursprünglichen `String`-Wert zu erhalten.

```
public static String encrypt(String valueToEnc) {  
    try {  
        Key key = generateKey();  
        Cipher c = Cipher.getInstance(ALGORITHM);  
        c.init(Cipher.ENCRYPT_MODE, key);  
        byte[] encValue = c.doFinal(valueToEnc.getBytes());  
        return Base64.getEncoder().encodeToString(encValue);  
    } catch (Exception e) {  
        return "";  
    }  
}
```

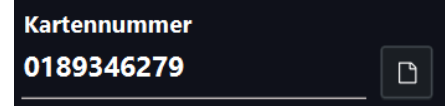
Als Beispiel: Aus der Kartennummer «5678» wird, mit Hilfe von diesem Algorithmus, die Zeichenfolge «P2NlxTfzq40Wir0Bi85wPg==».



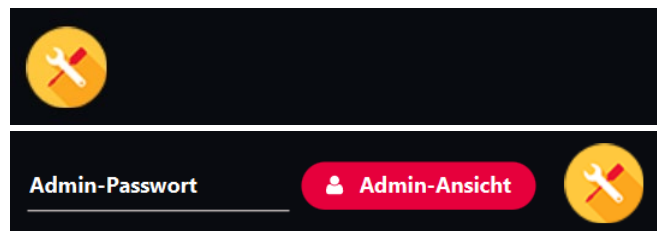
## 6 Abweichungen vom UI-Konzept

Wir werden in dieser Dokumentation die Funktionsweise des Programmes nicht nochmal erklären. Da wir uns sehr nach dem UI-Konzept gehalten haben, welches in [Kapitel 2.3](#) schon ausführlich erklärt und beschrieben ist. Auf die Abweichungen vom fertigen Programm zum UI-Konzept, die es gegeben hat, werden wir in diesem Kapitel noch eingehen. Behandelt werden grössere Abweichungen der Funktionalität und des Designs.

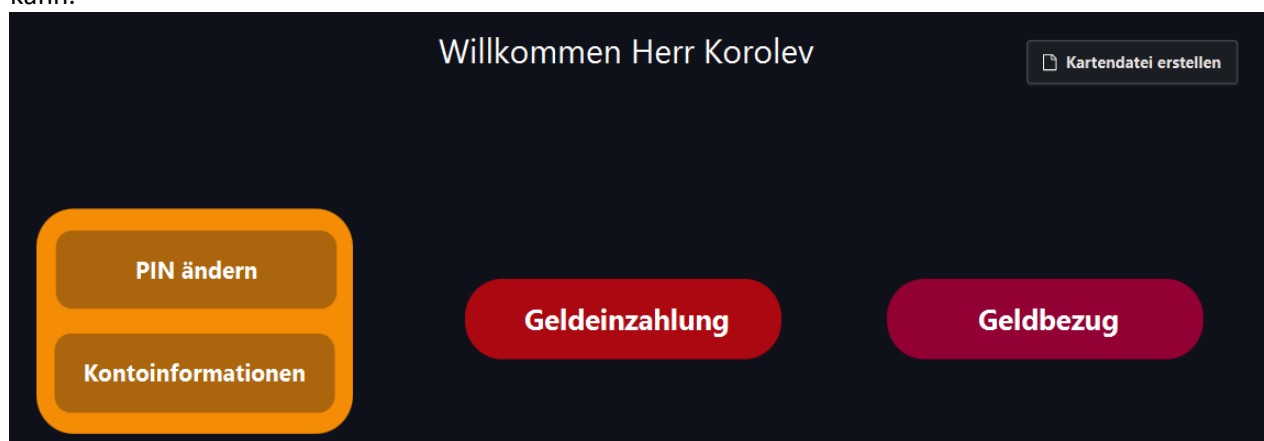
Bei der Login-View musste noch eine Option für die Auswahl einer Kartendatei als Alternative Eingabemöglichkeit angezeigt werden. Dies haben wir mit einem kleinen Button neben dem Feld für die Kartennummer gelöst.



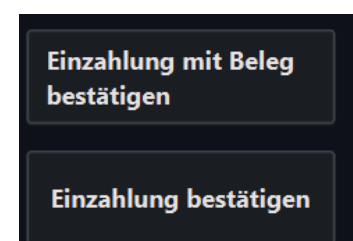
Das Admin-Login betrifft den grössten Teil der Nutzer nicht, deshalb ist es normalerweise ausgeblendet und erscheint nur, wenn über das «Settings» Icon unten links gefahren wird.



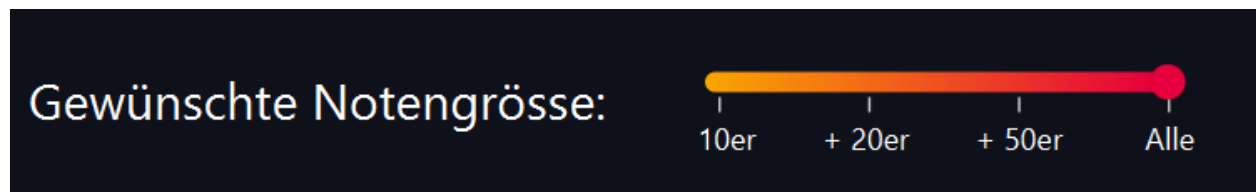
Die Master-View funktioniert genauso wie im Konzept beschrieben, nur dass die Hauptoptionen jeweils zu zweit unter einem Übertitel zusammengefasst werden, um den Nutzer nicht mit Buttons zu bombardieren. Zusätzlich erscheint auch eine Nachricht in der Mitte, die den Nutzer mit dem Namen begrüsst. Zudem musste oben rechts ein Button hinzugefügt werden, mit dem man eine neue Kartendatei erstellen kann.



Die DepositConfirm-View wurde um eine Einzahlungsbestätigungsfunktion mit Ausgabe eines Belegs ergänzt.



Da wir gegen Ende des Projekts noch Zeit und Lust hatten, haben wir in der WithdrawalConfirm-View einen Regler implementiert, mit welchem sich die gewünschte Notengrösse festlegen lässt. Man kann dabei auswählen, welche Noten ausgegeben werden sollen. «10er» heisst also, dass der ganze Betrag in Zehnernoten ausgegeben wird. Eine Auswahl von «+ 20er» führt dazu, dass der ganze Betrag in Zwanziger- und Zehnernoten berechnet und ausgegeben wird. Dasselbe gilt für «50er», dort kommen jedoch auch noch Fünfigernoten hinzu. Standardmässig ist der Regler auf «Alle» gesetzt, bei dieser Option berechnet ein Algorithmus, wie der Betrag mit der kleinstmöglichen Anzahl Noten ausgegeben werden kann und speichert die entsprechenden Noten.



Der Rest des Programmes funktioniert exakt so, wie es im UI-Konzept beschrieben ist.

## 7 Schlussbericht

### 7.1 Reflexion

Dies war unser bisher grösstes Projekt. Wir haben dementsprechend jede Woche viel daran gearbeitet. Am Anfang haben wir viel Zeit in die Planung und die Diagramme investiert. Im Verlauf des Projekts haben sich unsere Vorstellungen mehrmals verändert und die Diagramme mussten angepasst werden. Dementsprechend haben wir bei einigen Diagrammen etwa 10 verschiedene Versionen. Dies ist definitiv etwas, was es besser zu machen gilt. Ausserdem hat es beim Einrichten der Datenbank auf dem Raspberry Pi viele Probleme gegeben. Später, damit wir das Problem beheben konnten, haben wir einen eigenen Server aus einem alten PC gemacht. Das Ganze hat zwar sehr viel Zeit in Anspruch genommen, jedoch konnten wir auch daraus lernen und wissen jetzt, wie wir es beim nächsten Projekt machen. Wir haben aber auch aus anderen Dingen in unserem Projekt überdurchschnittlich viele neue Eindrücke gewonnen sowie Erfahrungen mit neuen Technologien und Herangehensweisen gemacht. Als Beispiel hierfür kann man das Abfragen von Daten durch eine API nehmen. Und auch mit dem Erstellen eines PDFs durch Code hatten wir bisher noch gar nichts zu tun. Durch das Verwenden von Ubuntu Server haben wir auch Erfahrungen in der Verwendung von Linux Distros sammeln können. Die gemeinsame Nutzung von Git haben wir uns seit einem der letzten Projekte angeeignet und es hat sich auch hier wieder als äusserst hilfreich erwiesen.

### 7.2 Fazit

Die Durchführung dieses Projekts war für uns eine sehr interessante Erfahrung. Speziell die Anwendung von Kenntnissen, die wir in mehreren Modulen erworben haben, wie zum Beispiel das Schreiben von Diagrammen, das Verwalten von SQL-Datenbanken, das Objektorientierte Programmieren und das Aufsetzen eines Servers, hat uns auch wirklich Spass gemacht. Wir erkannten durch dieses Projekt, wie sehr eben alle diese Themen zusammenhängen können und gemeinsam zu etwas Grösserem verbunden werden können. Wenn wir ehrlich sind, müssen wir zugeben, dass wir stolz auf unser fertiges Projekt sind. Wir haben alle unsere Muss, Soll und Kann-Ziele erreicht, und noch wesentlich mehr gemacht, als wir ursprünglich geplant hatten. Das Programm wurde zwar deswegen nicht schlechter, aber wir sind wohl ein wenig über das Ziel hinausgeschossen. Diese Projektarbeit wird uns auf jeden Fall in positiver Erinnerung bleiben als das wahrscheinlich lehrreichste und vielseitigste Projekt, das wir bisher durchgeführt haben.