Java Banking App

Maximilian Schüller
Justus Siegert
Fynn Thierling
Egzon Zenuni
Joscha Dierks

28. Juni 2023

Inhaltsverzeichnis

1 Einleitung

Das Projekt » Java Banking App « ist eine Java Anwendung mit einer Datenbankschnittstelle und einem User Interface (UI), genauer ein Graphical User Interface (GUI). Ziel ist es, eine intuitiv bedienbare Anwendung zu schaffen, die die vorgegebenen Funktionsparameter erfüllt.

2 Anforderungen an das Projekt

Das Projekt sollte ein UI umgesetzt mit der Technologie von `Java Swing` besitzen. Alternativ stand `Java AWT` zur Auswahl, jedoch haben wir dies nicht direkt genutzt. Zu erwähnen ist jedoch, dass `Java Swing` ein Nachfolger von `Java AWT` ist und darauf basiert, jedoch weitaus mehr Funktionen bietet. Außerdem ist `Java Swing` deutlich performanceschonender und unterstützt das native Design der einzelenen Betriebssysteme, ohne dass dafür eine direkte Spezifikation erforderlich ist. Dies macht die plattformunabhängige Programmierung deutlich einfacher.

Weiterhin ist es erforderlich, dass das Programm seine Daten dauerhaft speichert, also auch bei einem Neustart die Daten aus der letzten Sitzung wieder geladen werden.

Ein realistisches Klassenmodell mit sinnvoll verteilten Attributen und Methoden, einer Vererbunghierarchie, und abstrakten Klassen muss ebenso vorhanden sein. Eventuell wird hierbei auf Polymorphie und Datenkapselung gesetzt.

Eine durchgehende und nutzerfreundliche Fehlerbehandlung wird vorausgesetzt, besonders Nutzereingaben, Konvertierungen und Überläufe sind hierbei zu berücksichtigen.

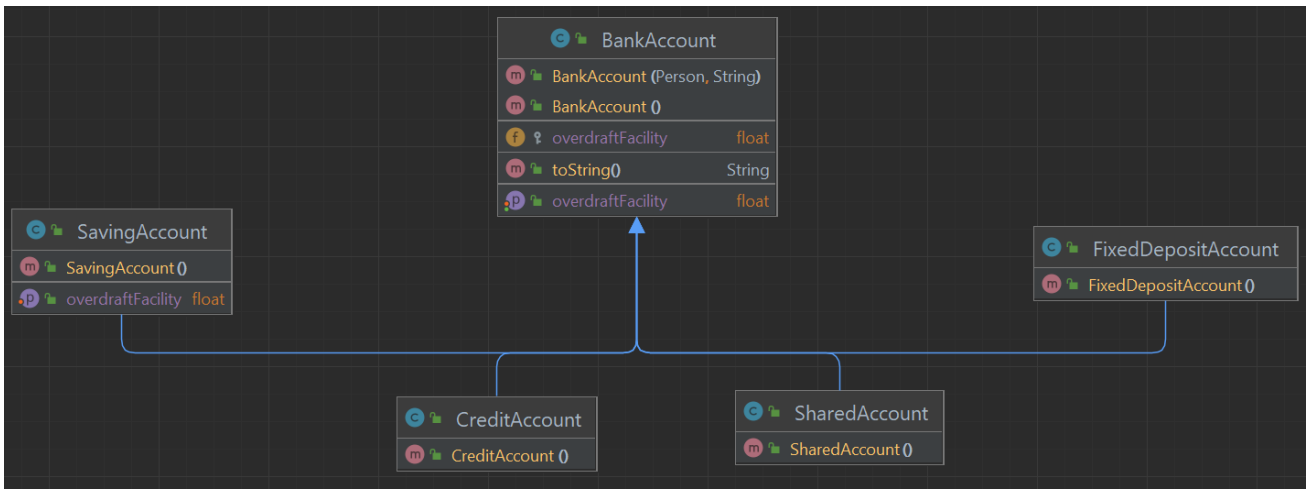
Der Programmcode folgt keinen explizieten Vorgaben, jedoch wird auf eine konsistente Formatierung und sinnvolle Bezeichnungen geachtet.

3 Projektplanung

Die vorausgehende Projektplanung ist spartanisch ausgefallen, um schnellstmöglich das Produkt fertig zu stellen. Daher beschränkt sich diese auf einen groben Aufbau des GUI und die nötigen Funktionsweisen. Wir einigten uns darauf, uns auf ein Hauptfenster zu beschränken, dass entweder die Transaktionsübersicht oder das Administrator-Dashboard anzeigt. Darüber hinaus sollen alle Funktionen ein eigenes Popup Fenster erhalten.

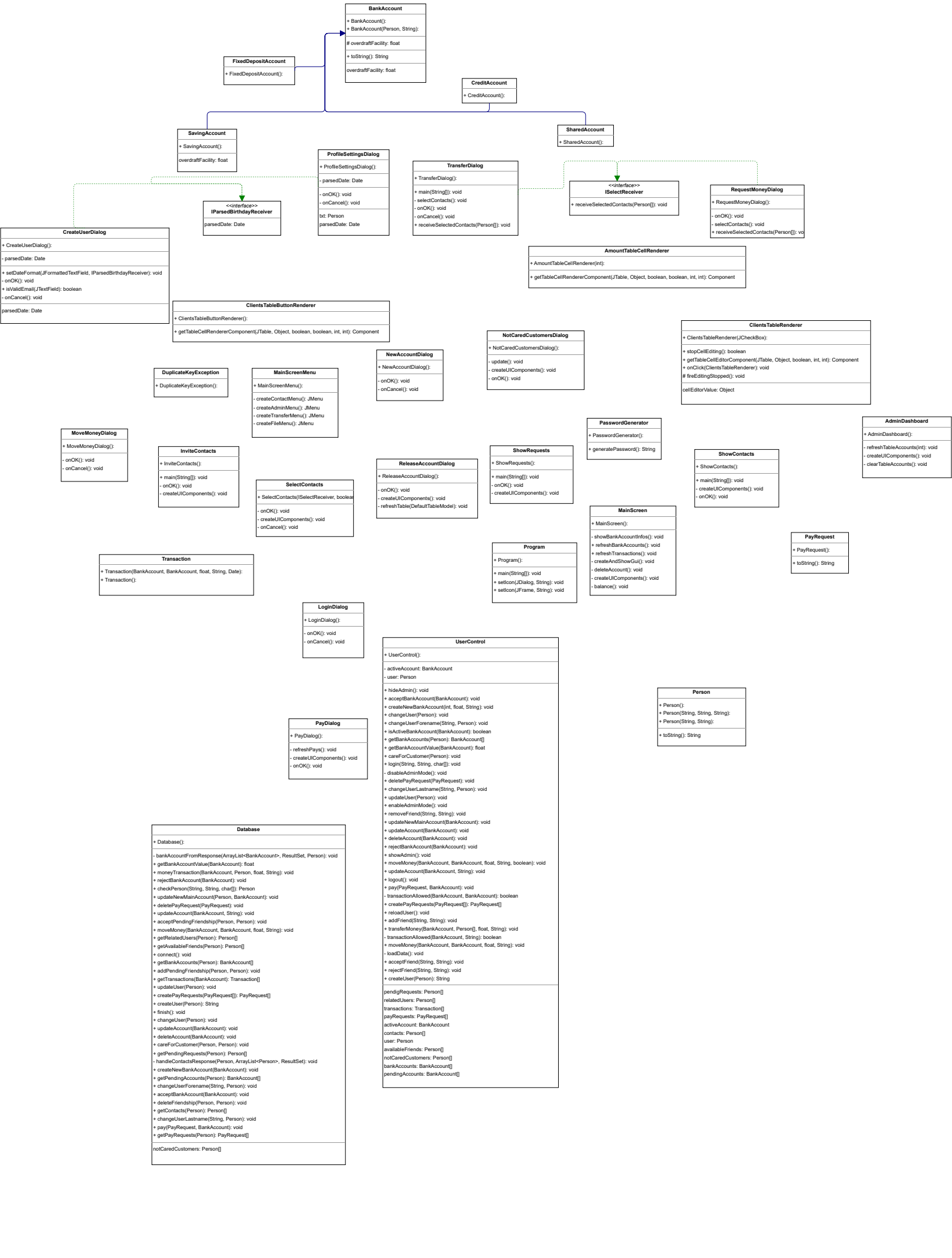
Der Zeitplan beinhaltet das Ziel, alle Module und Funktionen bis zum 23. Juni 2023 fertigzustellen, um die darauffolgenden Tage die Zusammenführung derselben und das finale Debuggen vornehmen zu können.

3.1 Klassenmodell



Das Klassenmodell leitet die verschiedenen Bankkonten von einem Standartkonto ab. Die Unterklassen ergänzen die Felder des Standartkontos. Die Funktionalität des Standartkontos wird durch die abgeleiteten Klassen tweilweise erweitert oder verändert. Ein Beispiel hierfür ist, dass ein Sparkonto die Funktion `setOverdraftFacility()` überschreibt, da ein Sparkonto nicht überzogen werden darf.

Die beiliegende vollständige Projektdokumentation, die mithilfe des Tools `Javadoc` erstellt wurde, liefert nähere Informationen über die einzelenen Felder und Methoden. Auf der folgenden Seite ist das Vollständige Klassendiagramm des Projekts zu sehen.


BankAccount
+ BankAccount():
+ BankAccount(Person, String):
# overdraftFacility: float
+ toString(): String
overdraftFacility: float
FixedDepositAccount
+ FixedDepositAccount():
CreditAccount
+ CreditAccount():
SavingAccount
+ SavingAccount():
overdraftFacility: float
SharedAccount
+ SharedAccount():
ProfileSettingsDialog
+ ProfileSettingsDialog():
- parsedDate: Date
- onOK(): void
- onCancel(): void
txt: Person
parsedDate: Date
TransferDialog
+ TransferDialog():
+ main(String[]): void
- selectContacts(): void
- onOK(): void
- onCancel(): void
+ receiveSelectedContacts(Person[]): void
<<interface>>
ISelectReceiver
+ receiveSelectedContacts(Person[]): void
RequestMoneyDialog
+ RequestMoneyDialog():
- onOK(): void
- selectContacts(): void
+ receiveSelectedContacts(Person[]): vo
<<interface>>
IParsedBirthdayReceiver
parsedDate: Date
CreateUserDialog
+ CreateUserDialog():
- parsedDate: Date
+ setDateFormat(JFormattedTextField, IParsedBirthdayReceiver): void
- onOK(): void
+ isValidEmail(JTextField): boolean
- onCancel(): void
parsedDate: Date
AmountTableCellRenderer
+ AmountTableCellRenderer(int):
+ getTableCellRendererComponent(JTable, Object, boolean, boolean, int, int): Component
ClientsTableButtonRenderer
+ ClientsTableButtonRenderer():
+ getTableCellRendererComponent(JTable, Object, boolean, boolean, int, int): Component
NotCaredCustomersDialog
+ NotCaredCustomersDialog():
- update(): void
- createUIComponents(): void
- onOK(): void
ClientsTableRenderer
+ ClientsTableRenderer(JCheckBox):
+ stopCellEditing(): boolean
+ getTableCellEditorComponent(JTable, Object, boolean, int, int): Component
+ onClick(ClientsTableRenderer): void
# fireEditingStopped(): void
cellEditorValue: Object
NewAccountDialog
+ NewAccountDialog():
- onOK(): void
- onCancel(): void
DuplicateKeyException
+ DuplicateKeyException():
MainScreenMenu
+ MainScreenMenu():
- createContactMenu(): JMenu
- createAdminMenu(): JMenu
- createTransferMenu(): JMenu
- createFileMenu(): JMenu
PasswordGenerator
+ PasswordGenerator():
+ generatePassword(): String
AdminDashboard
+ AdminDashboard():
- refreshTableAccounts(int): void
- createUIComponents(): void
- clearTableAccounts(): void
MoveMoneyDialog
+ MoveMoneyDialog():
- onOK(): void
- onCancel(): void
InviteContacts
+ InviteContacts():
+ main(String[]): void
- onOK(): void
- createUIComponents(): void
ReleaseAccountDialog
+ ReleaseAccountDialog():
- onOK(): void
- createUIComponents(): void
- refreshTable(DefaultTableModel): void
ShowRequests
+ ShowRequests():
+ main(String[]): void
- onOK(): void
- createUIComponents(): void
ShowContacts
+ ShowContacts():
+ main(String[]): void
- createUIComponents(): void
- onOK(): void
SelectContacts
+ SelectContacts(ISelectReceiver, boolean
- onOK(): void
- createUIComponents(): void
- onCancel(): void
MainScreen
+ MainScreen():
- showBankAccountInfos(): void
+ refreshBankAccounts(): void
+ refreshTransactions(): void
- createAndShowGui(): void
- deleteAccount(): void
- createUIComponents(): void
- balance(): void
PayRequest
+ PayRequest():
+ toString(): String
Transaction
+ Transaction(BankAccount, BankAccount, float, String, Date):
+ Transaction():
Program
+ Program():
+ main(String[]): void
+ setIcon(JDialog, String): void
+ setIcon(JFrame, String): void
LoginDialog
+ LoginDialog():
- onOK(): void
- onCancel(): void
UserControl
+ UserControl():
- activeAccount: BankAccount
- user: Person
+ hideAdmin(): void
+ acceptBankAccount(BankAccount): void
+ createNewBankAccount(int, float, String): void
+ changeUser(Person): void
+ changeUserForename(String, Person): void
+ isActiveBankAccount(BankAccount): boolean
+ getBankAccounts(Person): BankAccount[]
+ getBankAccountValue(BankAccount): float
+ careForCustomer(Person): void
+ login(String, String, char[]): void
- disableAdminMode(): void
+ deletePayRequest(PayRequest): void
+ changeUserLastname(String, Person): void
+ updateUser(Person): void
+ enableAdminMode(): void
+ removeFriend(String, String): void
+ updateNewMainAccount(BankAccount): void
+ updateAccount(BankAccount): void
+ deleteAccount(BankAccount): void
+ rejectBankAccount(BankAccount): void
+ showAdmin(): void
+ moveMoney(BankAccount, BankAccount, float, String, boolean): void
+ updateAccount(BankAccount, String): void
+ logout(): void
+ pay(PayRequest, BankAccount): void
- transactionAllowed(BankAccount, BankAccount): boolean
+ createPayRequests(PayRequest[]): PayRequest[]
+ reloadUser(): void
+ addFriend(String, String): void
+ transferMoney(BankAccount, Person[], float, String): void
- transactionAllowed(BankAccount, String): boolean
+ moveMoney(BankAccount, BankAccount, float, String): void
- loadData(): void
+ acceptFriend(String, String): void
+ rejectFriend(String, String): void
+ createUser(Person): String
pendigRequests: Person[]
relatedUsers: Person[]
transactions: Transaction[]
payRequests: PayRequest[]
activeAccount: BankAccount
contacts: Person[]
user: Person
availableFriends: Person[]
notCaredCustomers: Person[]
bankAccounts: BankAccount[]
pendingAccounts: BankAccount[]
Person
+ Person():
+ Person(String, String, String):
+ Person(String, String):
+ toString(): String
PayDialog
+ PayDialog():
- refreshPays(): void
- createUIComponents(): void
- onOK(): void
Database
+ Database():
- bankAccountFromResponse(ArrayList<BankAccount>, ResultSet, Person): void
+ getBankAccountValue(BankAccount): float
+ moneyTransaction(BankAccount, Person, float, String): void
+ rejectBankAccount(BankAccount): void
+ checkPerson(String, String, char[]): Person
+ updateNewMainAccount(Person, BankAccount): void
+ deletePayRequest(PayRequest): void
+ updateAccount(BankAccount, String): void
+ acceptPendingFriendship(Person, Person): void
+ moveMoney(BankAccount, BankAccount, float, String): void
+ getRelatedUsers(Person): Person[]
+ getAvailableFriends(Person): Person[]
+ connect(): void
+ getBankAccounts(Person): BankAccount[]
+ addPendingFriendship(Person, Person): void
+ getTransactions(BankAccount): Transaction[]
+ updateUser(Person): void
+ createPayRequests(PayRequest[]): PayRequest[]
+ createUser(Person): String
+ finish(): void
+ changeUser(Person): void
+ updateAccount(BankAccount): void
+ deleteAccount(BankAccount): void
+ careForCustomer(Person, Person): void
+ getPendingRequests(Person): Person[]
- handleContactsResponse(Person, ArrayList<Person>, ResultSet): void
+ createNewBankAccount(BankAccount): void
+ getPendingAccounts(Person): BankAccount[]
+ changeUserForename(String, Person): void
+ acceptBankAccount(BankAccount): void
+ deleteFriendship(Person, Person): void
+ getContacts(Person): Person[]
+ changeUserLastname(String, Person): void
+ pay(PayRequest, BankAccount): void
+ getPayRequests(Person): PayRequest[]
notCaredCustomers: Person[]

4 Installationsanleitung

Zum ausführen des Programms kann das beiliegende `jav` Archiv über die Konsole gestartet werden. Dazu wird der Befehl `$ java -jar Banking.jar` im Verzeichnis ausgeführt, welches das Archiv enthält.

Voraussetzung, um diesen Befehl ausführen zu können, ist die vollständige Installation des Java Runtime Environment (JRE) der Version 19 oder höher. Sollte das Java Development Kit (JDK) bereits vorhanden sein, kann auch dieses genutzt werden, um das JAR-Archiv auszuführen. Hier kann das JRE heruntergeladen werden `https://www.oracle.com/java/technologies/downloads/#jdk20-windows`. Es sollte sichergestellt werden, dass der Ordner `bin` im Installationsverzeichnis des JRE den Umgebungsvariabeln hinzugefügt wurde, bevor der Befehl in der Konsole ausgeführt wird.

Damit die Anwendung korrekt funktioniert, ist eine aktive Internetverbindung erforderlich!

5 Externe Software

Das Programm verwendet eine externe zentrale Datenbank, sodass jeder Client von überall auf der Welt mit einer Internetverbindung auf unseren Service zugreifen kann. Unser Applikation greift direkt vom Client auf die Datenbank zu. Dies ist in einer produktiven Umgebung **nicht** zu empfehlen! Die Zugangsdaten zur Datenbank stehen im Code des Clients und können mit verhältnismäßig wenig Aufwand ausgelesen werden. Ein Angreifer kann sich so leicht Zugriff zu Datenbank verschaffen.

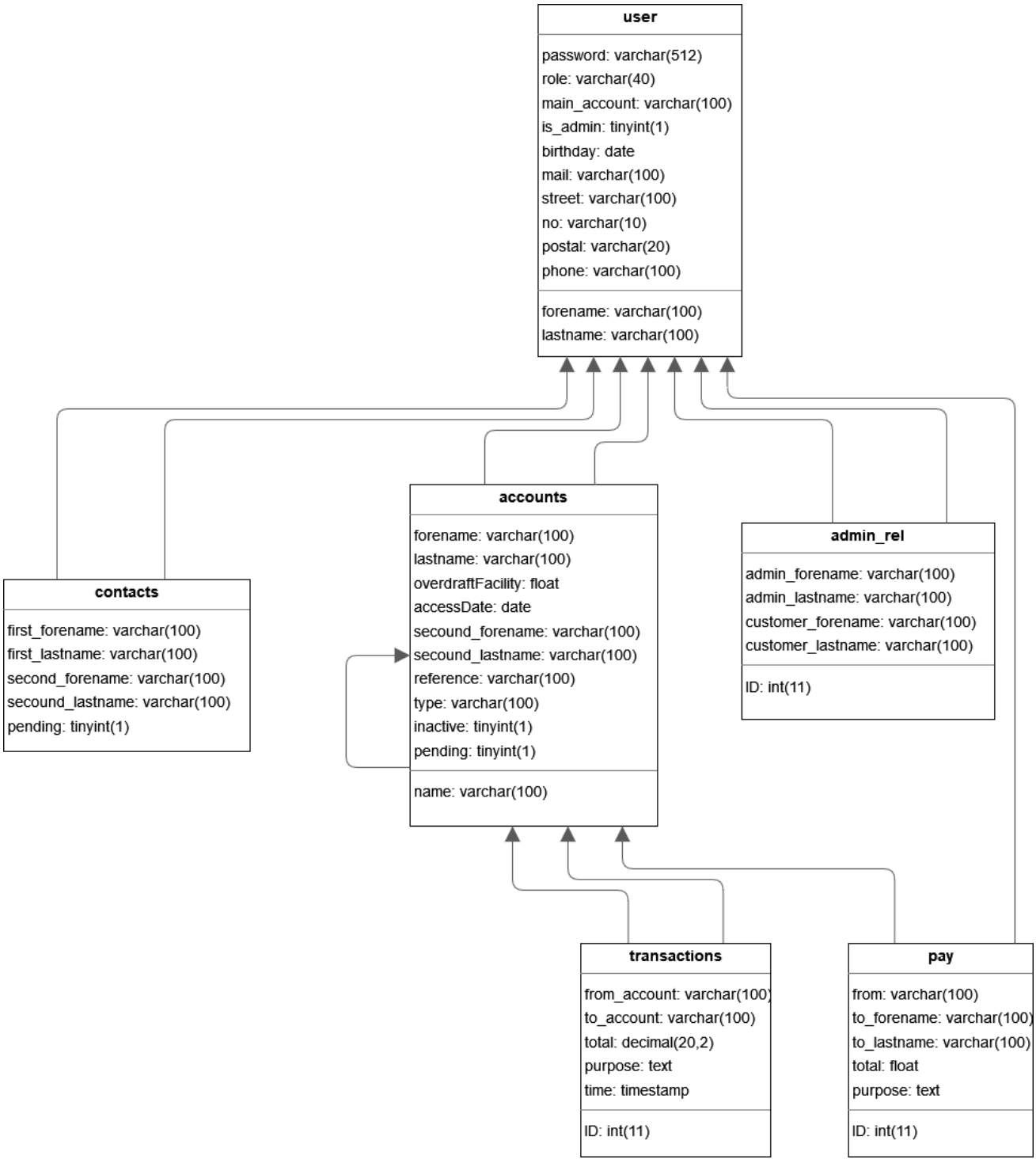
Zu empfehlen ist es, ein Application Interface (API) zwischen Client und Datenbank zu stellen, der die Authentifizierung der Nutzer übernimmt. So wird verhindert, dass ein Nutzer direkt auf die Datenbank zugreifen kann, er erhält also nur seine Anmeldedaten für sein Konto, aber niemals die Anmeldedaten für weiter Infrastrucktur.

Wir haben uns gegen dieses bewährte Modell entschieden, da der Arbeitsaufwand sich effektiv verdoppelt hätte, hätten wir ein API zu unserer Anwendung geschrieben. Da unsere Applikation nicht produktiv eingesetzt wird, sondern nur ausgewählten Personen zur Verfügung steht, ist das Risiko akzeptabel. Uns ist jedoch bewusst, dass in einer produktiven Umgebung ein solches Modell extrem unsicher ist.

5.1 Datenbank

Als Datenbank verwenden wir eine MariaDB `Ver 15.1 Distrib 10.6.12-MariaDB` auf einem `Ubuntu 22.04.2 LTS` V-Server.

Zum Verbinden mit der Datenbank verwenden wir die externe Bibliothek `org.mariadb.jdbc.Driver` der Version `3.1.4`, die hier heruntergeladen werden kann: `https://mariadb.com/downloads/connectors/connectors-data-access/java8-connector`



Abkürzungen

API Application Interface
GUI Graphical User Interface
JDK Java Development Kit
JRE Java Runtime Environment
UI User Interface