Module 2 Capstone - TEnmo

Congratulations—you've landed a job with TEnmo, whose product is an online payment service for transferring "TE bucks" between friends. However, they don't have a product yet. You've been tasked with writing a RESTful API server and command-line application.

Use cases

Required use cases

You should attempt to complete all of the following required use cases.

1. **[COMPLETE]** As a user of the system, I need to be able to register myself with a username and password.
    1. A new registered user starts with an initial balance of 1,000 TE Bucks.
    2. The ability to register has been provided in your starter code.
2. **[COMPLETE]** As a user of the system, I need to be able to log in using my registered username and password.
    1. Logging in returns an Authentication Token. I need to include this token with all my subsequent interactions with the system outside of registering and logging in.
    2. The ability to log in has been provided in your starter code.
3. As an authenticated user of the system, I need to be able to see my Account Balance.
4. As an authenticated user of the system, I need to be able to *send* a transfer of a specific amount of TE Bucks to a registered user.
    1. I should be able to choose from a list of users to send TE Bucks to.
    2. I must not be allowed to send money to myself.
    3. A transfer includes the User IDs of the from and to users and the amount of TE Bucks.
    4. The receiver's account balance is increased by the amount of the transfer.
    5. The sender's account balance is decreased by the amount of the transfer.
    6. I can't send more TE Bucks than I have in my account.
    7. I can't send a zero or negative amount.
    8. A Sending Transfer has an initial status of *Approved*.
5. As an authenticated user of the system, I need to be able to see transfers I have sent or received.
6. As an authenticated user of the system, I need to be able to retrieve the details of any transfer based upon the transfer ID.

Optional use cases

If you complete all of the required use cases and are looking for additional challenge, complete as many of the following optional use cases as you can.

7. As an authenticated user of the system, I need to be able to *request* a transfer of a specific amount of TE Bucks from another registered user.
    1. I should be able to choose from a list of users to request TE Bucks from.
    2. I must not be allowed to request money from myself.
    3. I can't request a zero or negative amount.
    4. A transfer includes the User IDs of the from and to users and the amount of TE Bucks.

    5. A Request Transfer has an initial status of *Pending*.
    6. No account balance changes until the request is approved.
    7. The transfer request should appear in both users' list of transfers (use case #5).
8. As an authenticated user of the system, I need to be able to see my *Pending* transfers.
9. As an authenticated user of the system, I need to be able to either approve or reject a Request Transfer.
    1. I can't "approve" a given Request Transfer for more TE Bucks than I have in my account.
    2. The Request Transfer status is *Approved* if I approve, or *Rejected* if I reject the request.
    3. If the transfer is approved, the requester's account balance is increased by the amount of the request.
    4. If the transfer is approved, the requestee's account balance is decreased by the amount of the request.
    5. If the transfer is rejected, no account balance changes.

Sample screens

Use case 3: Current balance

```
Your current account balance is: $9999.99
```

Use case 4: Send TE Bucks

```
-------------------------------------------
Users
ID          Name
-------------------------------------------
313         Bernice
54          Larry
---------

Enter ID of user you are sending to (0 to cancel):
Enter amount:
```

Use case 5: View transfers

```
-------------------------------------------
Transfers
ID          From/To                 Amount
-------------------------------------------
23          From: Bernice          $ 903.14
79          To:    Larry           $  12.55
---------
Please enter transfer ID to view details (0 to cancel): "
```

Use case 6: Transfer details

```
-------------------------------------------
Transfer Details
-------------------------------------------
 Id: 23
 From: Bernice
 To: Me Myselfandi
 Type: Send
 Status: Approved
 Amount: $903.14
```

Use case 7: Requesting TE Bucks

```
-------------------------------------------
Users
ID          Name
-------------------------------------------
313         Bernice
54          Larry
---------

Enter ID of user you are requesting from (0 to cancel):
Enter amount:
```

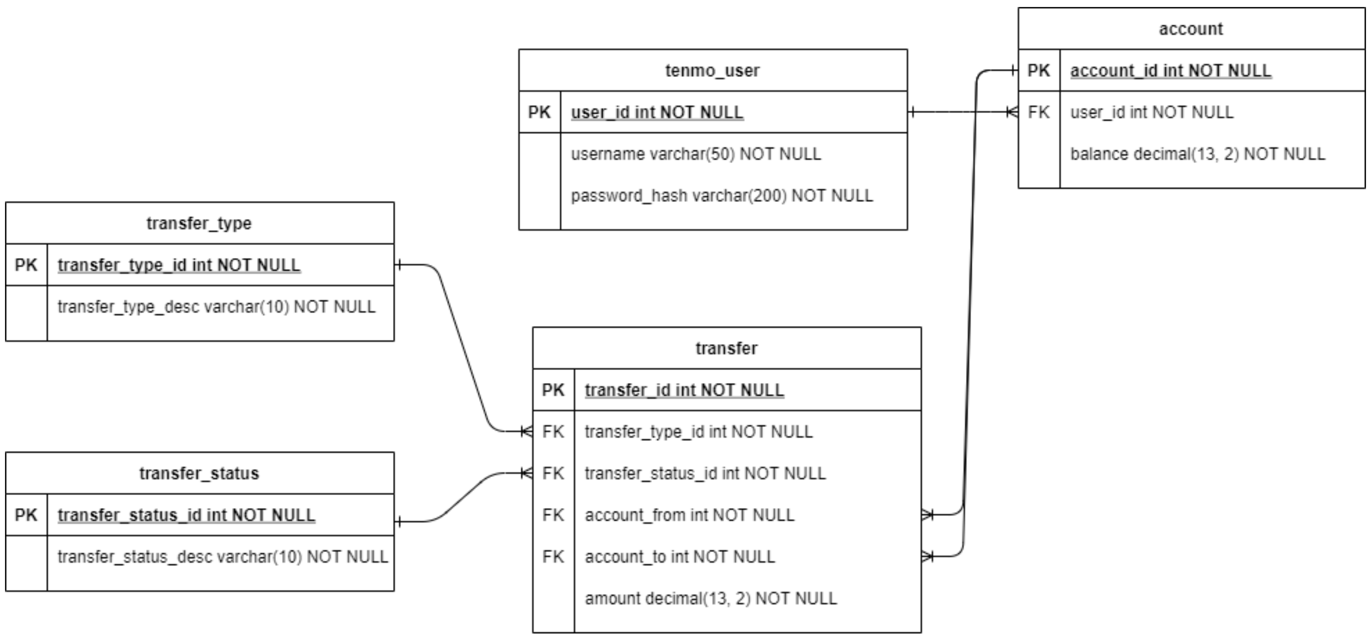
Use case 8: Pending requests

```
-------------------------------------------
Pending Transfers
ID          To                     Amount
-------------------------------------------
88          Bernice                $ 142.56
147         Larry                  $  10.17
---------
Please enter transfer ID to approve/reject (0 to cancel): "
```

Use case 9: Approve or reject pending transfer

```
1: Approve
2: Reject
0: Don't approve or reject
---------
Please choose an option:
```

Database schema


tenmo_user
PK user_id int NOT NULL
username varchar(50) NOT NULL
password_hash varchar(200) NOT NULL
account
PK account_id int NOT NULL
FK user_id int NOT NULL
balance decimal(13, 2) NOT NULL
transfer_type
PK transfer_type_id int NOT NULL
transfer_type_desc varchar(10) NOT NULL
transfer_status
PK transfer_status_id int NOT NULL
transfer_status_desc varchar(10) NOT NULL
transfer
PK transfer_id int NOT NULL
FK transfer_type_id int NOT NULL
FK transfer_status_id int NOT NULL
FK account_from int NOT NULL
FK account_to int NOT NULL
amount decimal(13, 2) NOT NULL

`tenmo_user` table

Stores the login information for users of the system.

Field	Description
`user_id`	Unique identifier of the user
`username`	String that identifies the name of the user; used as part of the login process
`password_hash`	Hashed version of the user's password
`role`	Name of the user's role

`account` table

Stores the accounts of users in the system.

Field	Description
`account_id`	Unique identifier of the account
`user_id`	Foreign key to the `users` table; identifies user who owns account
`balance`	The amount of TE bucks currently in the account

`transfer_type` table

Stores the types of transfers that are possible.

Field	Description
`transfer_type_id`	Unique identifier of the transfer type
`transfer_type_desc`	String description of the transfer type

There are two types of transfers:

`transfer_type_id`	`transfer_type_desc`	Purpose
1	Request	Identifies transfer where a user requests money from another user
2	Send	Identifies transfer where a user sends money to another user

`transfer_status` table

Stores the statuses of transfers that are possible.

Field	Description
`transfer_status_id`	Unique identifier of the transfer status
`transfer_status_desc`	String description of the transfer status

There are three statuses of transfers:

`transfer_status_id`	`transfer_status_desc`	Purpose
1	Pending	Identifies transfer that hasn't occurred yet and requires approval from the other user
2	Approved	Identifies transfer that has been approved and occurred
3	Rejected	Identifies transfer that wasn't approved

`transfer` table

Stores the transfers of TE bucks.

Field	Description
`transfer_id`	Unique identifier of the transfer
`transfer_type_id`	Foreign key to the `transfer_types` table; identifies type of transfer
`transfer_status_id`	Foreign key to the `transfer_statuses` table; identifies status of transfer
`account_from`	Foreign key to the `accounts` table; identifies the account that the funds are being taken from
`account_to`	Foreign key to the `accounts` table; identifies the account that the funds are going to
`amount`	Amount of the transfer

> Note: there are two check constraints in the DDL that creates the `transfer` table. Be sure to take a look at `tenmo.sql` to understand these constraints.

How to set up the database

Create a new Postgres database called `tenmo`. Run the `database/tenmo.sql` script in pgAdmin to set up the database.

Datasource

A Datasource has been configured for you in `/src/resources/application.properties`.

```
# datasource connection properties
spring.datasource.url=jdbc:postgresql://localhost:5432/tenmo
spring.datasource.name=tenmo
spring.datasource.username=postgres
spring.datasource.password=postgres1
```

JdbcTemplate

If you look in `/src/main/java/com/techelevator/dao`, you'll see `JdbcUserDao`. This is an example of how to get an instance of `JdbcTemplate` in your DAOs. If you declare a field of type `JdbcTemplate` and add it as an argument to the constructor, Spring automatically injects an instance for you:

```
@Service
public class JdbcUserDao implements UserDao {

    private JdbcTemplate jdbcTemplate;

    public JdbcUserDao(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }
}
```

Testing

DAO integration tests

`com.techelevator.dao.BaseDaoTests` has been provided for you to use as a base class for any DAO integration test. It initializes a Datasource for testing and manages rollback of database changes between tests.

`com.techelevator.dao.JdbUserDaoTests` has been provided for you as an example for writing your own DAO integration tests.

Remember that when testing, you're using a copy of the real database. The schema and data for the test database are defined in `/src/test/resources/test-data.sql`. The schema in this file matches the schema defined in `database/tenmo.sql`.

Authentication

The user registration and authentication functionality for the system has already been implemented. If you review the login code, you'll notice that after successful authentication, an instance of `AuthenticatedUser` is stored in the `currentUser` member variable of `App`. The user's authorization token—meaning JWT—can be accessed from `App` as `currentUser.getToken()`.

When the use cases refer to an "authenticated user", this means a request that includes the token as a header. You can also reference other information about the current user by using the `User` object retrieved from `currentUser.getUser()`.