Mandatory Assignment II – SD20w2

General Information

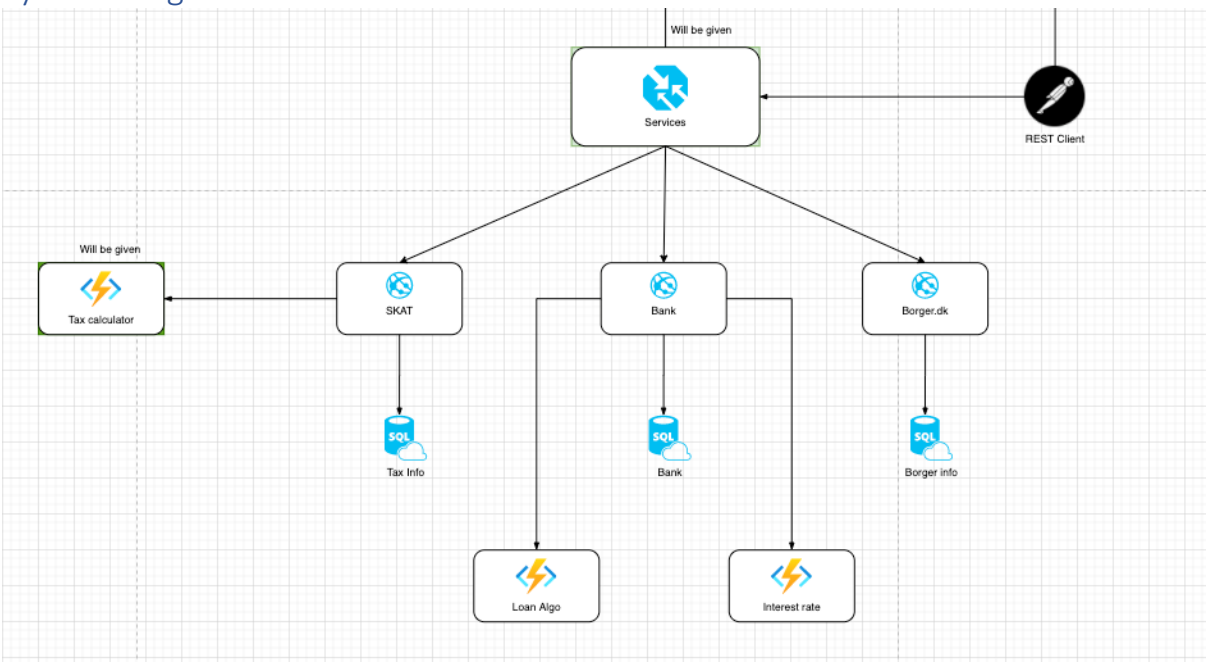
The time has come for you to apply the knowledge that was taught thus far and complete your second mandatory assignment.

The purpose of this assignment is to showcase different scenarios and to make you develop different main/supporting systems that overall will contribute to the creation of a distributed system – following an architecture based on both microservices and serverless architecture. You may use any programming language, database provider and framework to fulfil this mandatory. As long as it works, it is RESTful and the systems are well built and return the appropriate status codes.

You can find the starting code base here:
https://bitbucket.org/_bogz_/si_mandatory_assignment_2/src/master/

It is expected of you to work together, learn something and of course have fun. Please be aware of all the rules and guidelines regarding working together in terms of COVID-19. While I cannot ensure that you will not be in close contact with each other (Less than 1 meter distance) outside of school, while working in a group in KEA you have to wear a mask, should you be in a less than 1 meter proximity to one of your group mates and under no circumstances you are to use another person's computer.

System Diagram:



The tax calculator and the base gateway will be given. It is your responsibility to develop the rest. A description of each system will be given and since you will also get the source code, it should be fairly simple to implement these systems based on the code snippets and example from your group's repository: https://bitbucket.org/_bogz_/sd20w2/src/master/

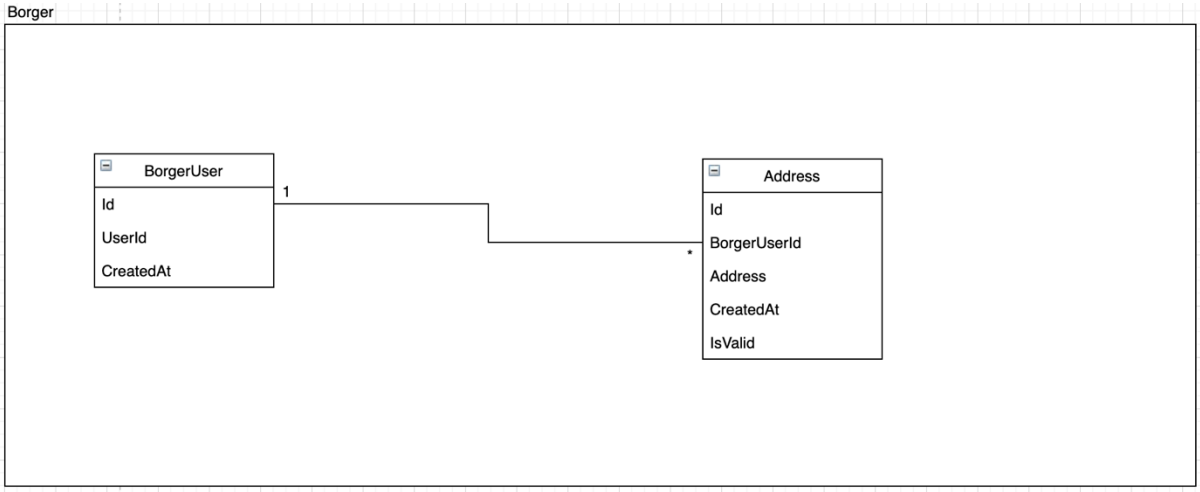
Task and System description:

1. Borger System (To be developed):

- Description:

This system will simply register the address of a user. It will communicate only directly with the gateway. It should be connected to a relational database that has two tables:

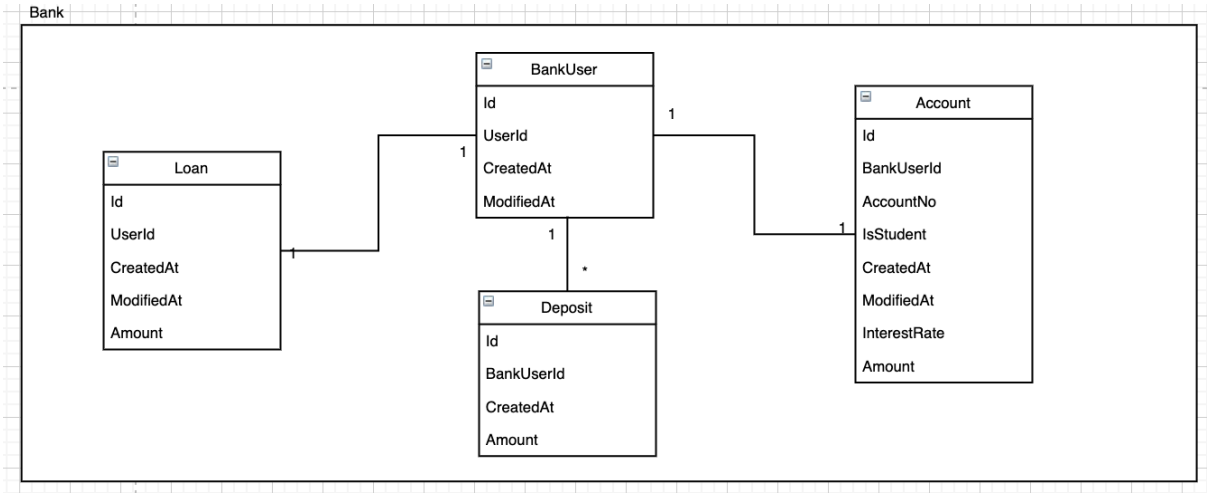
- Requirements:
  - Implement CRUD operations for BorgerUser and Address entities
- Considerations:
  - An Address cannot be created unless the user already exists
  - A BorgerUser can only have one active Address at any given point in time
  - Updating the address will result in the old one being voided (IsValid = false), and the new one replacing it.
  - Deleting the user will delete the corresponding addresses



2. Bank (To be developed):

- Description

This system will contain information about a User's loans, deposits and account. The service will be comprised of three different systems: An API and 2 functions. The API will be connected to a relational database with the following data model:



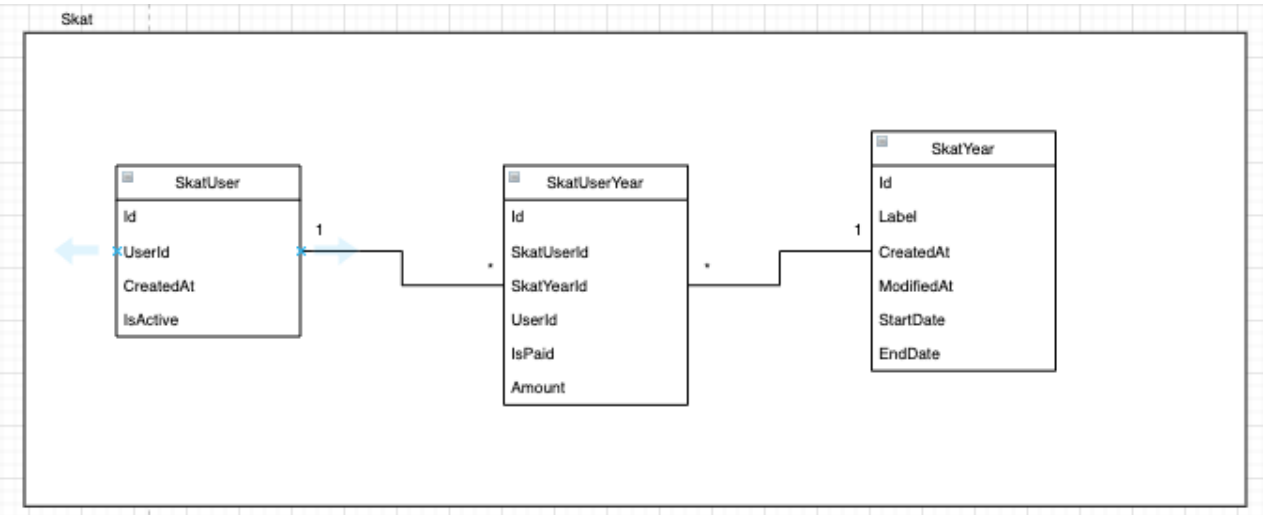
- Requirements:
  1. Bank API
     - Implement CRUD operations for: BankUser, Account
     - Implement /add-deposit endpoint:
       - Will receive a body with an amount and a BankUserId
       - Amount deposited cannot be null or negative
       - A deposit amount will first be sent to the interest rate function – the result will be saved in the database
       - A record will be inserted in the deposits table as well
     - Implement /list-deposits endpoint:
       - GET request that takes a BankUserId
       - Returns a list of all the deposits that were made by that user
     - Implement /create-loan endpoint:
       - Request will contain the BankUserId and LoanAmount.
       - A POST request will be made to the Loan Algorithm Function with an amount and the total account amount for that user
       - If the status code is 200, a loan record will be created
       - If the status code is 403 or similar, an error will be returned
     - Implement /pay-loan endpoint:
       - The loan can be paid integrally by using the /pay-loan endpoint. The request will contain the BankUserId and the LoanId as well. This will make the amount from a loan 0 and will subtract that amount from the account of that user. If there aren't enough money on the account, an error will be returned.
     - Implement /list-loans endpoint
       - GET request which will return a list of all the loans belonging to a user that are greater than 0 (not paid)
       - The request should have a BankUserId to retrieve the loans for a user.
     - Implement /withdrawl-money endpoint:
       - The body of that request should contain an amount and a UserId(Not BankUserId, not SkatUserId)
       - Subtract (if possible) the amount from that users account. Throw an error otherwise.

  2. Loan Algorithm Function:
     - Has a POST endpoint.
     - Will calculate if the loan exceeds 75% of the account amount. If it does, return 403, otherwise 200
  3. Interest Rate Function:
     - Has a POST endpoint
     - Upon a new deposit, this function will be called. It will return the amount + 2% of the deposited amount.

3. Skat (To be developed):

- Description:

This system accounts for the taxes that all the users need to pay on a yearly basis. The service is comprised of two systems: An API and a function (Tax calculator). This service will be connected to a database with the following data model:



- Requirements:

1. Skat API:
    - Implement CRUD operations for SkatUser
    - Implement CRUD operations for SkatYear. When creating a SkatYear entry, keep in mind the following:
        - It will create as many entries in the SkatUserYear Table as there are SkatUsers
    - Implement a /pay-taxes endpoint (POST request):
        - Takes a body with a UserId (Not BankUserId, Not SkatUserId – UserId that comes from the supposed Main System developed in Mandatory I) and the total amount that is in that user's bank account.
        - An initial check will occur – if the user did not previously paid his taxes
        - A user is deemed to have paid his taxes if the value is greater than 0 in the SkatUserYear table.
        - A call will be made to the Tax Calculator and depending on the response, the SkatUserYear will be updated with the returned sum from the Tax Calculator and the IsPaid property will be set to true…
        - Make a call to an endpoint in Bank API to subtract money from account. The body of that request should contain an amount and a UserId(Not BankUserId, not SkatUserId)
2. Tax Calculator Function:
    - Receives a post request with an amount property to http://localhost:7071/api/Skat_Tax_Calculator
    - Returns the calculated amount that needs to be paid.
    - An error will be thrown if the value is negative.

4. Gateway (I will give you the shell – the port and endpoints that your system will use will be added by you)

Note: If you want, you can also just use a different gateway implementation. Do keep in mind that this will be the interaction point towards your suite of microservices/functions

Deadline and submission:

The deadline is Wednesday the 24th of November at 11:59 PM. The deliverables for this assignment are a document that contains a link to your repository, as well as who implemented what part of the system. Feel free to leave any additional feedback that you might have. For example, what went well and/or What could have been done better.

NOTES:
While I am not really interested in the used programming language/framework you should keep in mind the following considerations:

a. The system must work
b. The minimum requirements are ought to be implemented
c. The appropriate status codes and messages should be returned
d. Validation is a plus, but not 100% necessary – UNLESS where I have specified that you should handle a particular scenario
e. You are more than welcome to pick my brain when we have classes with questions, both regarding the mandatory as well as anything else that the course covers
f. Have fun xD

You will submit your assignment through fronter. Make sure that the repository is public. During week 47 (Thursday, the 26th of November) we will look at the mandatories and reflect upon them. I will again ask you questions as if we were in the examination. That way you will be well prepared for your exam and you can emphasize on the topics that you are not that sure about.