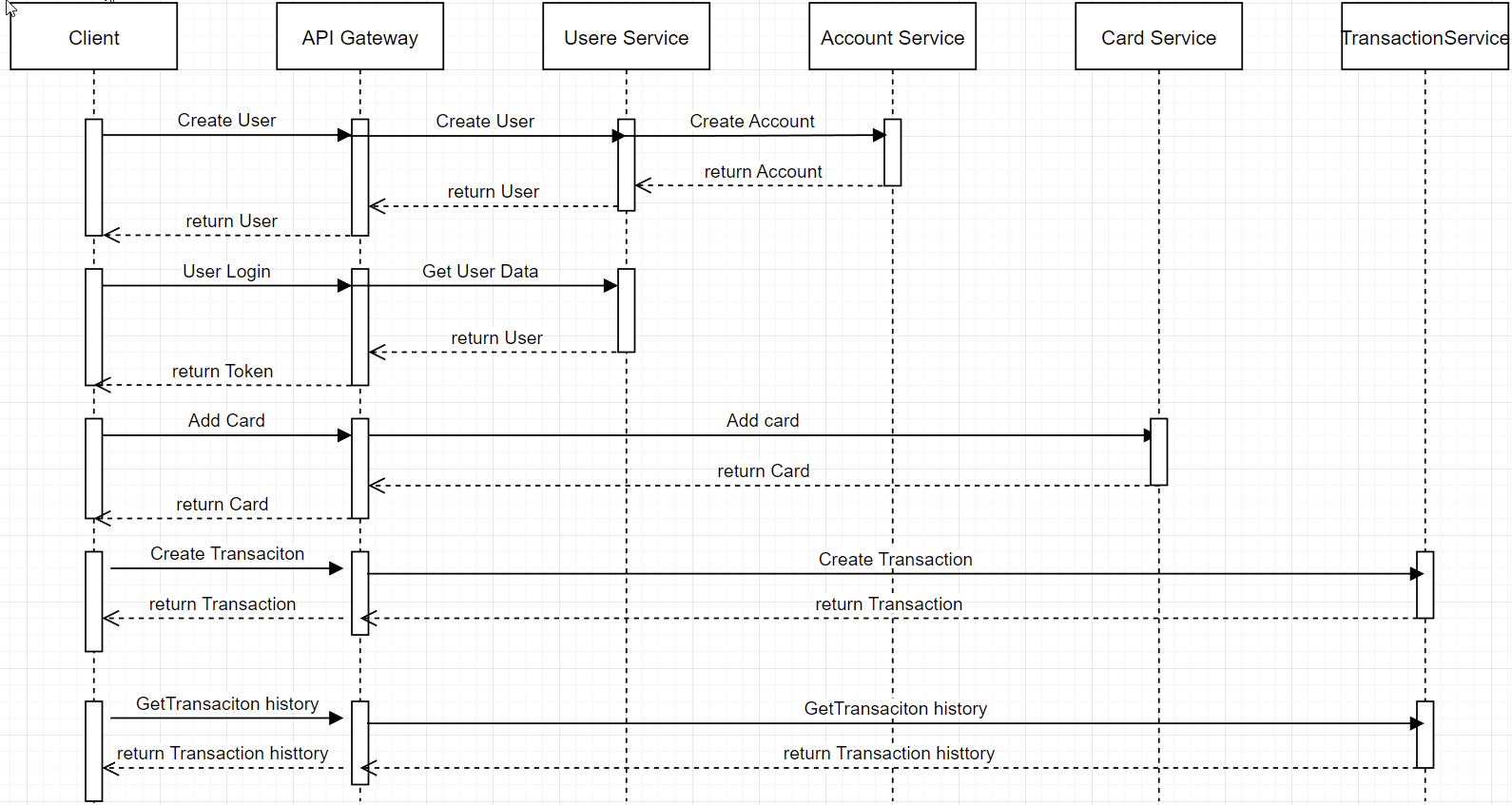
**Description of the accounting system**

**User perspective interactions:**

* Create a user
* Login (authenticate) user
* Add card to user account
* Receive transaction
* Make transaction
* Get account info
* View transaction history

**Backend components:**

Here is a diagram of the system

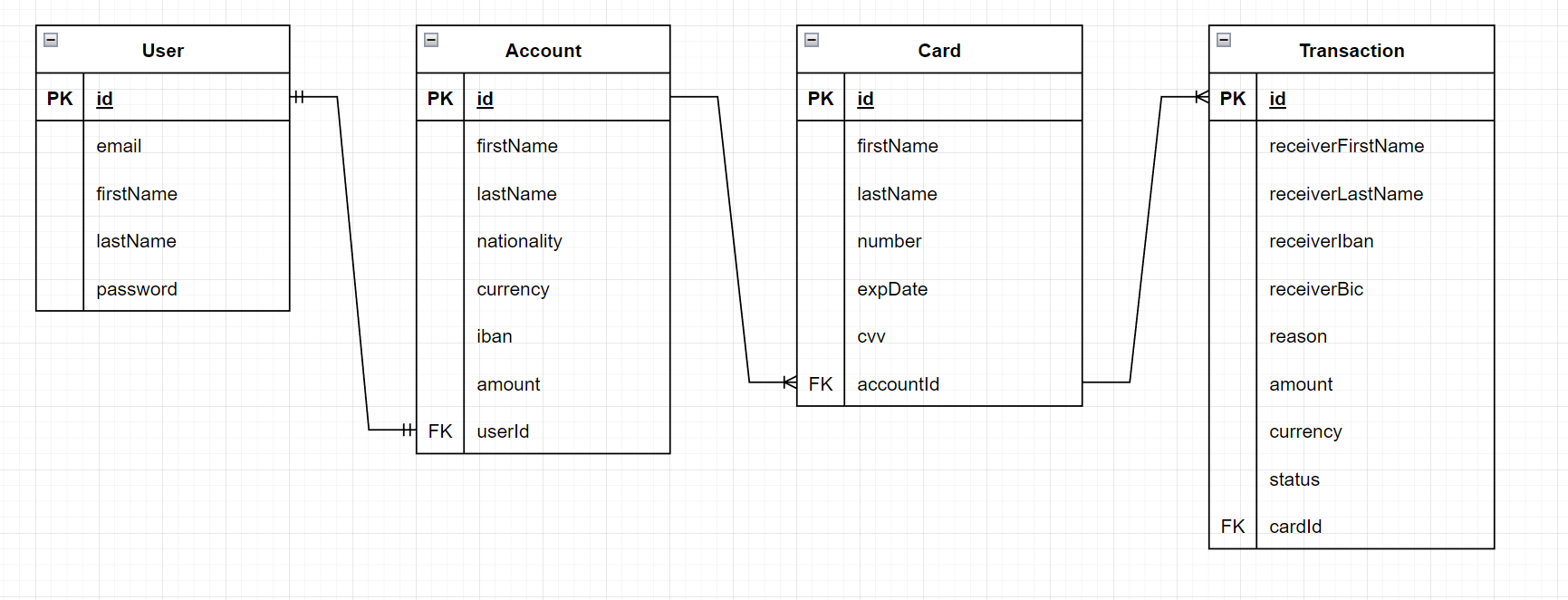


* API Gateway that will receive all the end user requests and will forward the request to the relevant service and also will be implementing authentication and authorization
* User service that will be managing the user operations and data
* Account service that will be managing the account operations and data
* Card service that will be managing the card operations and data
* Transaction service that will be managing the transaction operations and data

**Data Models and Storage:**

* For storage I used no relational database, although for most of the models I would have used relational database, but because I know you are using NoSQL DB, I wanted to implement the solution that way
* Models:
  + User:
    - email
    - password
    - firstName
    - lastName
    - isEnabled
  + Account
    - Iban
    - balance
    - currency
    - isEnabled
    - userId
  + Card
    - firstName
    - lastName
    - number
    - expData
    - cvv
    - userId
  + Transaction
    - fromFirstName
    - fromLastName
    - fromIban
    - toFirstName
    - toLastName
    - toIban
    - amount
    - currency
    - reason
    - status
    - cardId

I am attaching a model if I had to implement it with relation DB:



**Security**

The system will be with microservice architecture, that is ready to deploy on AWS cloud environment. We are having API gateway service that is the only service that is going to be accessible for user interactions and the rest of them will be in closed virtual network and will be allowed to be accessed only form the security groups of the needed microservices.

In order to have access to the API Gateway the user should have master API key which will be generated and give it to him (in our case is just env variable) that grad him access to create user in the system. I implemented a middleware to check for the requested header value.

After the user is created now he should login in order to interact with the rest of the system. The login will create JWT token containing user information and this token will be needed in order to interact with the rest of the routes, because the JWT middleware is attached to it.

Other security aspects that are handled are SQL injections and also, we are storing all sensitive data like password by hashing them.

**Infrastructure**

There are many ways that the services can be distributed. As the application is small, we can host it on one EC2 with small requirements running all the Docker images or Kubernetes pods in it. The first bottleneck will be the API Gateway service and for that we can have couple of replicas.

We will need AWS Mongo dB cluster for the database storage

We will need to configure secrets with secret manager, the security groups and VPC and finally to add the domain name to access the application from web.

**Testing**

Every service has unit tests that can be run locally and when we are submitting the code with Jenkins.

The test cases cover the business logic that is extracted in the services, so basically, we are not chasing 100% code coverage. The tests are generating report for the execution and coverage where you can check the results.

Also, there are integration test that can be run as sanity tests for most common cases, once all services are deployed on the testing environment. The collection is generated with Postman and it is run by Newman package. If you have all the services running you can run the collection from the zip file with the following commands:

npm install -g newman

newman run Integration tests.postman\_collection.json

You can check the Readme.md for further details about the application