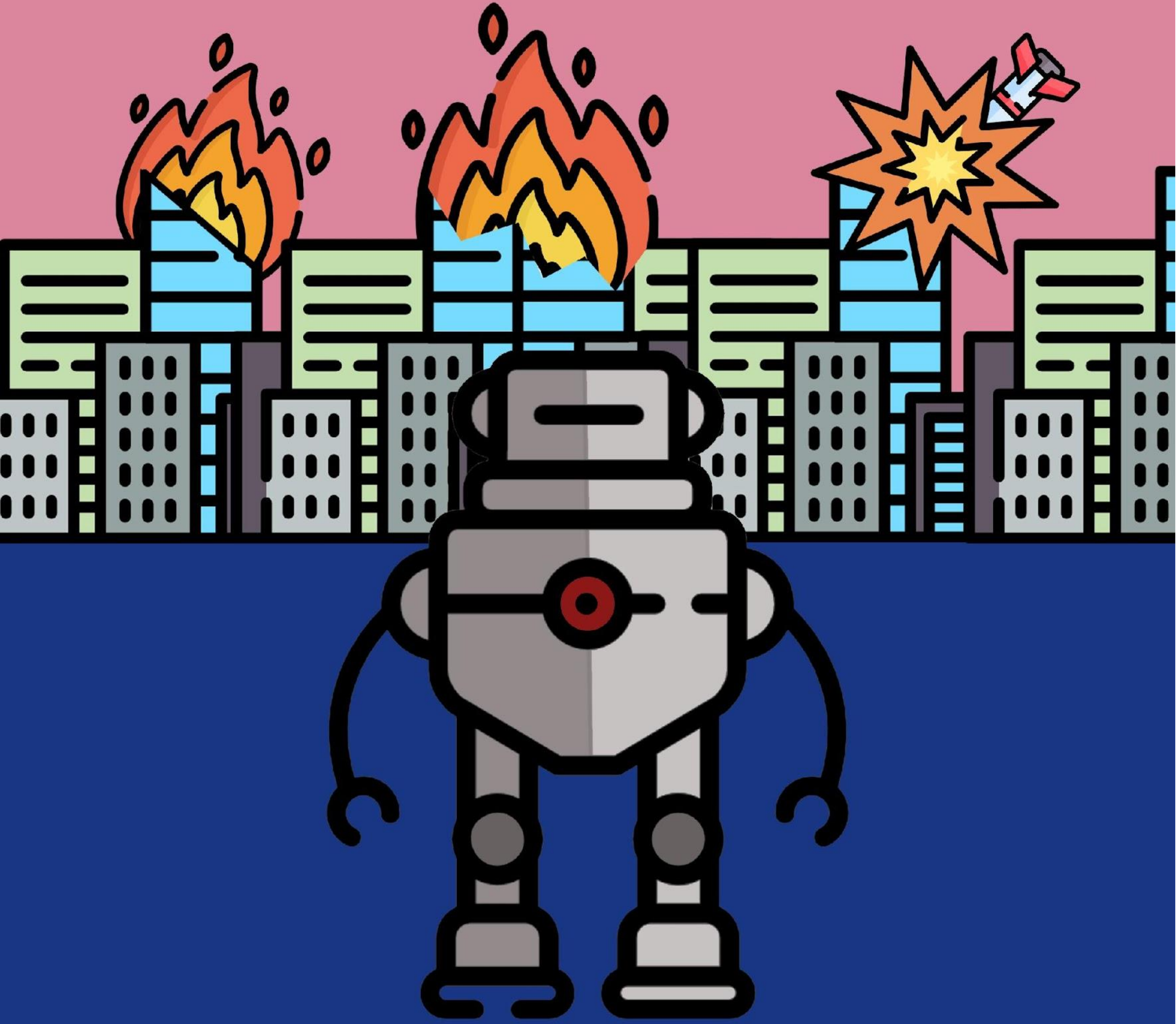


CRYPTOCOP

DESTROYER OF MODERN CURRENCIES

Below are all captions within the assignment description - pay attention to all of them as they may contain information useful for the assignment.

Cryptocop

You have received a message from an individual who calls himself “Droid” and he has been mining for cryptocurrency over the last years and is ready to start a cryptocurrency sales platform. You are going to fulfill his request and make him the cryptocurrency sales platform he needs!

Assignment description

In this assignment we are going to build a microservice structure which consists of four services. Below each service will be added to a specific section with all its requirements.

External APIs

An external API will be used to collect data regarding available cryptocurrencies and exchanges. The documentation for the external API can be found here: https://data.messari.io/docs. This is an open API which requires no authentication. Only the **Assets** and **Markets** sections within the API will be useful for this assignment. I advise you to read through the documentation and try the API out in order to fetch the correct data within the API.

Web API (70%)

The web API should be written in **.NET Core** and the requirements for the API are the following:

Authentication using JWT (10%)

- All endpoints should require authentication excluding: **(2%)**
  - **/api/account/register [POST]**
  - **/api/account/signin [POST]**
- The authentication scheme used is **JWT (8%)**
  - Configuration settings should be stored in **appsettings.json (1%)**
  - The authentication scheme should be setup using a middleware **(4%)**
  - The middleware should setup the **OnTokenValidated** event which checks whether the token is blacklisted or not **(2%)**
  - When the middleware has been implemented it should be registered within **Startup.cs (1%)**

Endpoints (10%)

Each controller should make use of a corresponding service in order to retrieve data from both the database and the external API

- AccountController **(3%)**
  - **/api/account/register [POST]** - Registers a user within the application, see **Models** section for reference
  - **/api/account/signin [POST]** - Signs the user in by checking the credentials provided and issuing a JWT token in return, see **Models** section for reference
  - **/api/account/signout [GET]** - Logs the user out by voiding the provided JWT token using the id found within the claim
- ExchangeController **(1%)**
  - **/api/exchanges [GET]** - Gets all exchanges in a paginated envelope. This routes accepts a single query parameter called **pageNumber** which is used to paginate the results
- CryptocurrencyController **(1%)**

    - **/api/cryptocurrencies [GET]** - Gets all available cryptocurrencies - the only available cryptocurrencies in this platform are BitCoin (*BTC*), Ethereum (*ETH*), Tether (*USDT*) and Monero (*XMR*)
- ShoppingCartController **(4%)**
    - **/api/cart [GET]** - Gets all items within the shopping cart, see **Models** section for reference
    - **/api/cart [POST]** - Adds an item to the shopping cart, see **Models** section for reference
    - **/api/cart/{id} [DELETE]** - Deletes an item from the shopping cart
    - **/api/cart/{id} [PATCH]** - Updates the quantity for a shopping cart item
    - **/api/cart [DELETE]** - Clears the cart - all items within the cart should be deleted
- AddressController **(2%)**
    - **/api/addresses [GET]** - Gets all addresses associated with authenticated user
    - **/api/addresses [POST]** - Adds a new address associated with authenticated user, see **Models** section for reference
    - **/api/addresses/{id} [DELETE]** - Deletes an address by id
- PaymentController **(2%)**
    - **/api/payments [GET]** - Gets all payment cards associated with the authenticated user
    - **/api/payments [POST]** - Adds a new payment card associated with the authenticated user, see **Models** section for reference
- OrderController **(2%)**
    - **/api/orders [GET]** - Gets all orders associated with the authenticated user
    - **/api/orders [POST]** - Adds a new order associated with the authenticated user, see **Models** section for reference

Service project (17.5%)

All service classes should make use of a corresponding repository class if it is fetching data from the database

- **AccountService.cs (1%)**
    - CreateUser
        - Creates the user using the appropriate repository class
    - AuthenticateUser
        - Authenticates the user using the appropriate repository class
    - Logout
        - Voids the JWT token using the appropriate repository class
- **CryptoCurrencyService.cs (2.5%)**
    - GetAvailableCryptocurrencies
        - Call the external API and get all cryptocurrencies with fields required for the **CryptoCurrencyDto** model
        - Deserializes the response to a list - I would advise to use the **HttpResponseMessageExtensions** which is located within **Helpers/** to deserialize and flatten the response.
        - Return a filtered list where only the available cryptocurrencies BitCoin (*BTC*), Ethereum (*ETH*), Tether (*USDT*) and Monero (*XMR*) are within the list
- **ExchangeService.cs (2.5%)**
    - GetExchanges
        - Call the external API with a paginated query and get all exchanges with fields required for the **ExchangeDto** model
        - Deserialize the response to a list - I would advise to use the **HttpResponseMessageExtensions** which is located within **Helpers/** to deserialize and flatten the response.

    - Create an envelope and add the list to the envelope and return that
- **JwtTokenService.cs (1%)**
  - IsTokenBlacklisted
    - Checks if token is blacklisted using the appropriate repository class
- **OrderService.cs (2%)**
  - GetOrders
    - Gets all orders using the appropriate repository class
  - CreateNewOrder
    - Create a new order using the appropriate repository class
    - Delete the current shopping cart
    - Publish a message to RabbitMQ with the routing key **‘create-order’** and include the newly created order
- **QueueService.cs (2%)**
  - PublishMessage
    - Serialize the object to JSON
    - Publish the message using a channel created with the RabbitMQ client
- **ShoppingCartService.cs (2.5%)**
  - GetCartItems
    - Gets all cart items using the appropriate repository class
  - AddCartItem
    - Call the external API using the product identifier as an URL parameter to receive the current price in USD for this particular cryptocurrency
    - Deserialize the response to a **CryptoCurrencyDto** model
    - Add it to the database using the appropriate repository class
  - RemoveCartItem
    - Removes the cart item by id using the appropriate repository class
  - UpdateCartItemQuantity
    - Updates the quantity of the cart item using the appropriate repository class
  - ClearCart
    - Clears the users cart using the appropriate repository class
- **TokenService.cs (2%)**
  - GenerateJwtToken
    - Creates a valid JWT token and assigns the information stored within the **UserDto** model as claims and returns the newly created token
- **PaymentService.cs (1%)**
  - AddPaymentCard
    - Adds the payment card using the appropriate repository class
  - GetStoredPaymentCards
    - Get all stored payment cards using the appropriate repository class
- **AddressService.cs (1%)**
  - GetAllAddresses
    - Gets all addresses using the appropriate repository class
  - AddAddress
    - Adds the address using the appropriate repository class
  - DeleteAddress
    - Deletes the address using the appropriate repository class

Repository project (17.5%)

All repository classes should make use of the **DbContext** in order to retrieve data from the database

- **AddressRepository (2.5%)**
  - GetAllAddresses
    - Gets all addresses from the database associated with the authenticated user
  - AddAddress
    - Add an address to the database
  - DeleteAddress
    - Delete an address from the database using the id and email. A user can only delete addresses associated with him
- **OrderRepository (4%)**
  - GetOrders
    - Gets all orders from the database associated with the authenticated user
  - CreateNewOrder
    - Retrieve information for the user with the email passed in
    - Retrieve information for the address with the address id passed in
    - Retrieve information for the payment card with the payment card id passed in
    - Create a new order where the credit card number has been masked, e.g. ************5555
    - Return the order but here the credit card number should not be masked
- **PaymentRepository (2%)**
  - AddPaymentCard
    - Add a payment card to the database
  - GetStoredPaymentCards
    - Gets all stored payment cards from the database associated with the authenticated user
- **ShoppingCartRepository (4%)**
  - GetCartItems
    - Gets all cart items from the database associated with the authenticated user
  - AddCartItem
    - Add a cart item to the database
  - RemoveCartItem
    - Remove a cart item from the database
  - UpdateCartItemQuantity
    - Update a cart items quantity within the database
  - ClearCart
    - Clear all cart items from the shopping cart in the database
- **TokenRepository (2%)**
  - CreateNewToken
    - Add a new token to the database
  - IsTokenBlacklisted
    - Check to see if the token is blacklisted within the database
  - VoidToken
    - Set the token to blacklisted within the database
- **UserRepository (3%)**
  - CreateUser
    - Check if user with same email exists within the database - if it does do not continue
    - Add a user to the database where the password has been hashed using the hashing function provided

    - Create a new token within the database
    - Return the user
- AuthenticateUser
    - Check if user has provided the correct credentials by comparing the email and password - if it is not correct do not continue
    - Create a new token within the database
    - Return the user

Models project (5%)

- Setup all **DTOs** (see Models for reference)
- Setup all **InputModels** (see Models for reference)
- Setup all **Entities** (*see* Database diagram *for reference*)

Database (5%)

- Navigate to https://www.elephantsql.com/ and register for a new PostgreSQL database
- Add the connection string to **appsettings.json** in the API project
- Setup a **DbContext** for the newly created database in the Repository project
- Register the **DbContext** within the API project
- Create your first migration and update the database according to those migrations (*this can be repeated every time the entity models change*)

Dockerfile (5%)

- A Dockerfile should be created in order to run this application in Docker
- This file should be located in the root of the application

Payment service (10%)

The payment service can be written a programming language of your choice and the requirements are the following:

- AMQP listener **(2.5%)**
    - Sets up a queue called **payment-queue** which is bound to the **create-order** routing key
- Validate the credit card received within the order **(2.5%)**
    - A third party tool can be used to validate the credit card number
- Print out the validation message in the console **(2.5%)**
- Dockerfile **(2.5%)**

Email service (10%)

The email service can be written a programming language of your choice and the requirements are the following:

- AMQP listener **(2.5%)**
    - Sets up a queue called **email-queue** which is bound to the **create-order** routing key
- Send an email using **Mailgun** stating that the order was successful **(5%)**
    - The email should be setup in a proper manner using HTML structure **(1%)**
    - The following information should be part of the email: **(4%)**
        - Name of customer
        - Street name and number
        - City
        - Zip code
        - Country
        - Date of order
        - Total price
        - Order items
- Dockerfile **(2.5%)**

RabbitMQ (5%)

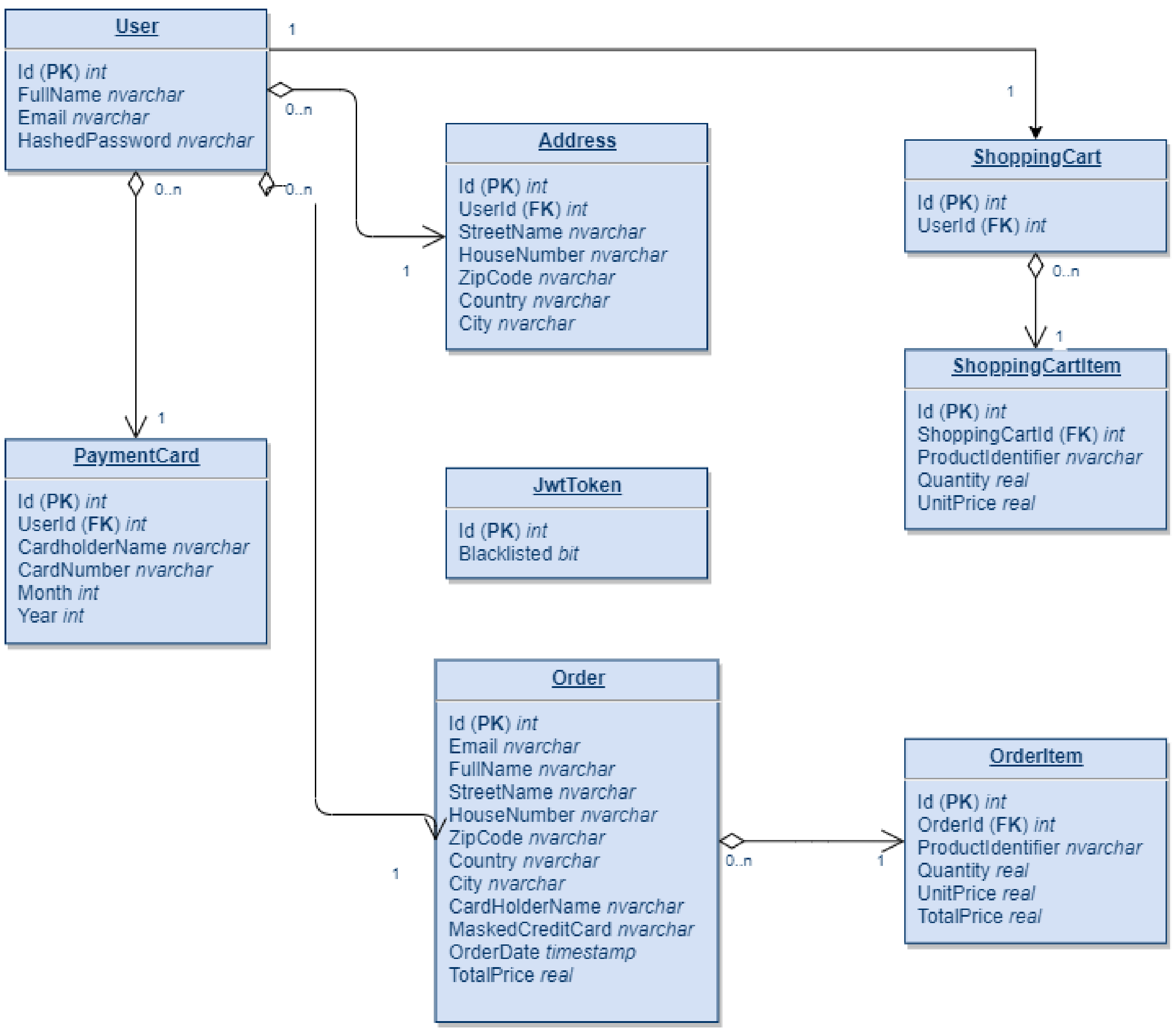
This service works as a glue between the other services to communicate with each other via AMQP.

Docker compose (5%)

A single docker-compose.yml should be a part of this structure in order to start and stop the microservice structure at will.

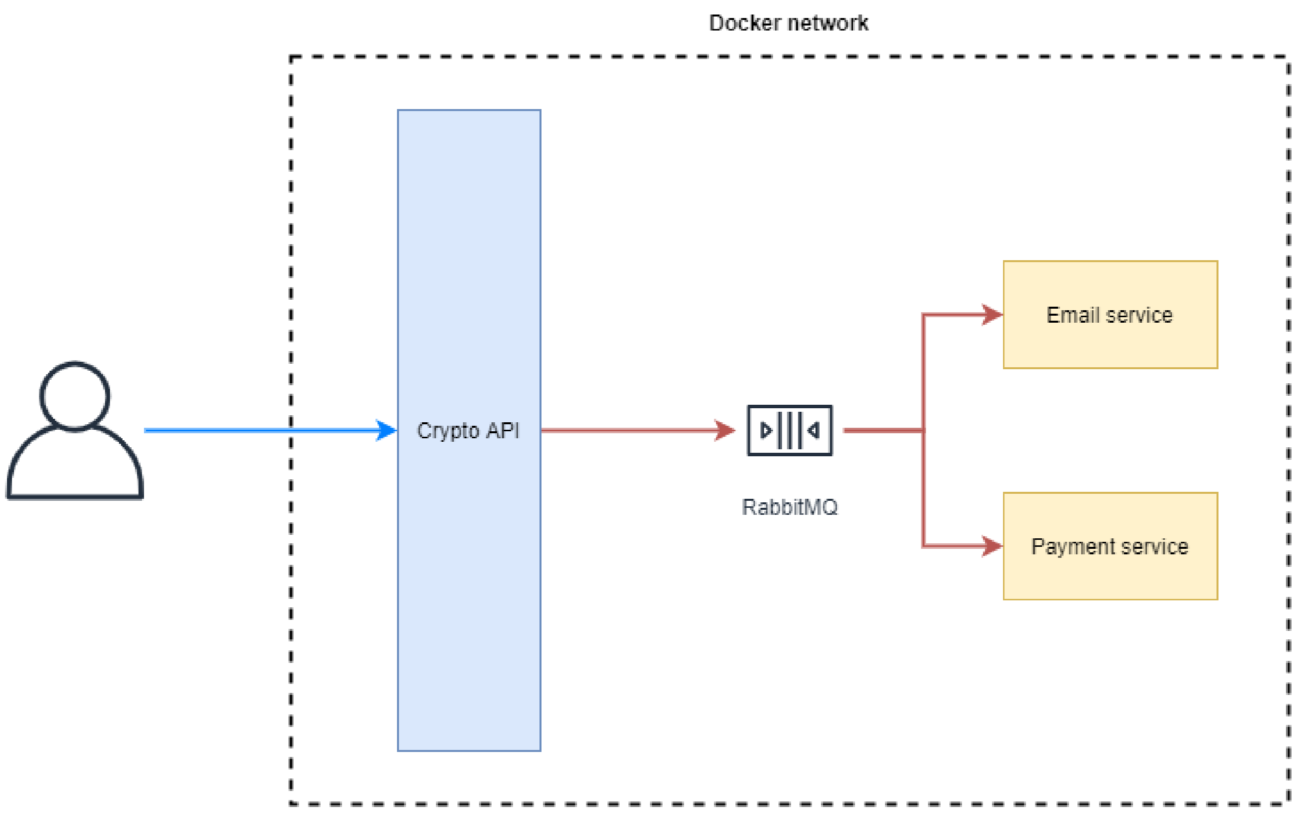
Database diagram

Here below is a diagram of how the database should look like.



Microservice structure

Here below is an overview of how the microservice structure should function as a whole.



Models

Below you can see the model structure for each model within the application, this includes: **Dtos** and **InputModels**. Entity models are excluded because they can be derived from the database diagram.

Data Transfer Objects (DTOs)

- **ExchangeDto**
    - Id (string)
    - Name (string)
    - Slug (string)
    - AssetSymbol (string)
    - PriceInUsd (nullable float)
    - LastTrade (nullable datetime)
- **CryptocurrencyDto**
    - Id (string)
    - Symbol (string)
    - Name (string)
    - Slug (string)
    - PriceInUsd (float)
    - ProjectDetails (string)
- **ShoppingCartItemDto**
    - Id (int)
    - ProductIdentifier (string)
    - Quantity (float)
    - UnitPrice (float)
    - TotalPrice (float)
- **AddressDto**
    - Id (int)
    - StreetName (string)
    - HouseNumber (string)
    - ZipCode (string)
    - Country (string)
    - City (string)
- **PaymentCardDto**
    - Id (int)
    - CardholderName (string)
    - CardNumber (string)
    - Month (int)
    - Year (int)
- **OrderDto**
    - Id (int)
    - Email (string)
    - FullName (string)
    - StreetName (string)
    - HouseNumber (string)
    - ZipCode (string)
    - Country (string)
    - City (string)
    - CardholderName (string)
    - CreditCard (string)

    - OrderDate (string)
        - Represented as **01.01.2020**
    - TotalPrice (float)
    - OrderItems (list of OrderItemDto)
- **OrderItemDto**
    - Id (int)
    - ProductIdentifier (string)
    - Quantity (float)
    - UnitPrice (float)
    - TotalPrice (float)
- **UserDto**
    - Id (int)
    - FullName (string)
    - Email (string)
    - TokenId (int)

Input models

In this section * means it is a required property.

- **AddressInputModel**
    - StreetName* (string)
    - HouseNumber* (string)
    - ZipCode* (string)
    - Country* (string)
    - City* (string)
- **LoginInputModel**
    - Email* (string)
        - Must be a valid email address
    - Password* (string)
        - A minimum length of 8 characters
- **OrderInputModel**
    - AddressId (int)
    - PaymentCardId (int)
- **PaymentCardInputModel**
    - CardholderName* (string)
        - A minimum length of 3 characters
    - CardNumber* (string)
        - Must be a valid credit card number
    - Month (int)
        - The range for this number is an inclusive 1 to 12
    - Year (int)
        - The range for this number is an inclusive 0 to 99

- **RegisterInputModel**
    - Email* (string)
        - Must be a valid email address
    - FullName* (string)
        - A minimum length of 3 characters
    - Password* (string)
        - A minimum length of 8 characters
    - PasswordConfirmation* (string)
        - A minimum length of 8 characters
        - Must be the same value as the property **Password**
- **ShoppingCartItemInputModel**
    - ProductIdentifier* (string)
    - Quantity* (nullable float)
        - The range for this number is an include 0.01 to the float type maximum value