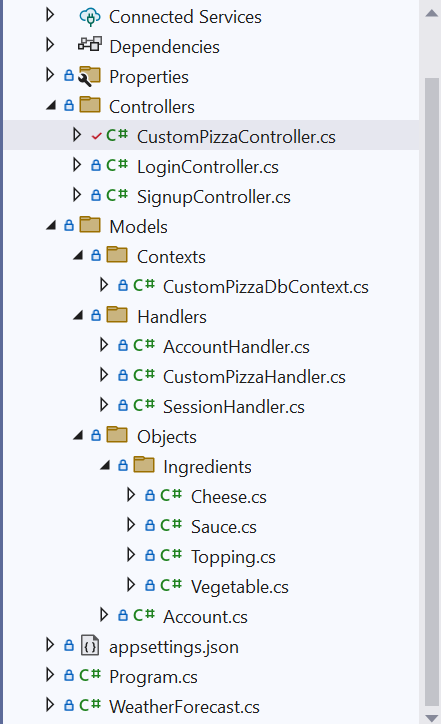
\*Note: For clarification purposes, some parts of this documentation will be written in Vietnamese.

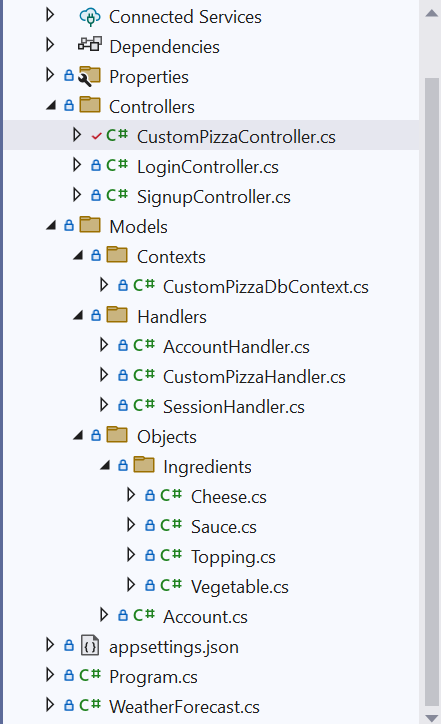
**PizzaWeb API - Documentation - Ver 1.0**

1. **Overall Structure**
   1. **Controllers: Classses that handles client requests** (e.g. HttpPost, HttpGet...), and in some cases, **returns appropriate data** to client in JSON format.

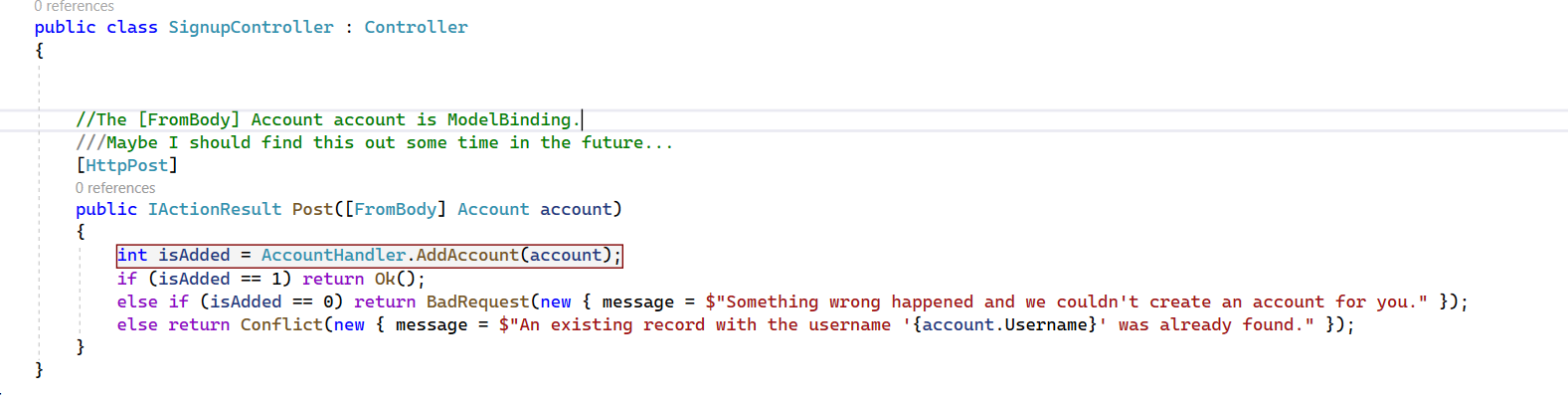
**CustomPizzaController:** Provides the client with lists of Pizza ingredients (e.g. Sauces, Cheeses,...), depending on the parameter.

**LoginController:** Checks if the requested account is available in the database. If logged in successfully, a SessionID will be provided to the client.

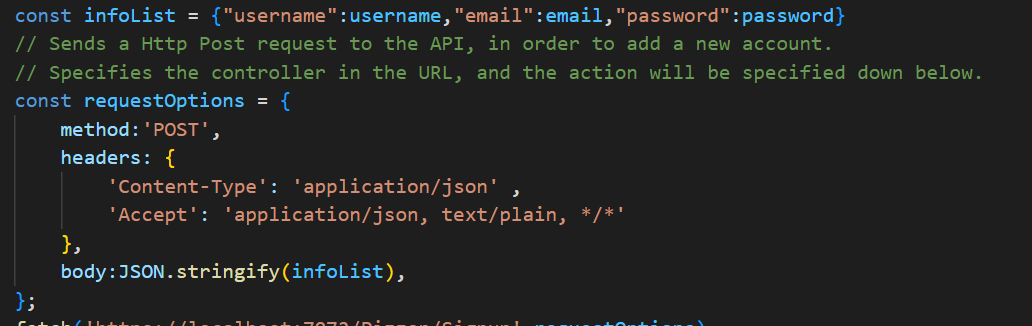
**SignupController:** Handles the creation of new accounts (customer, admins (to be added))

****

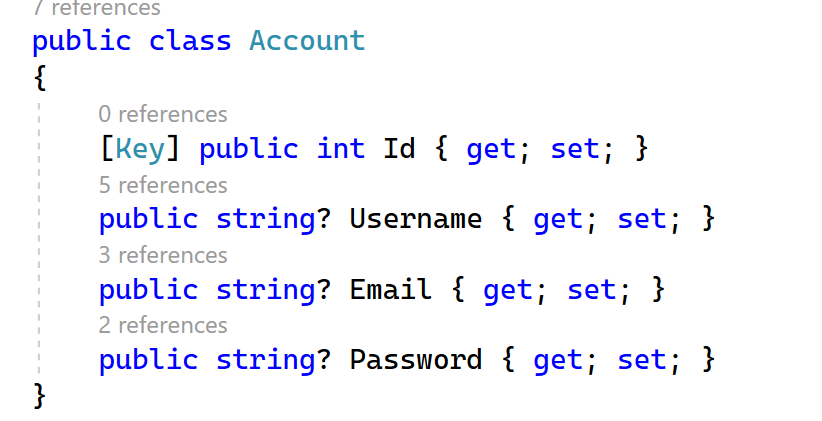
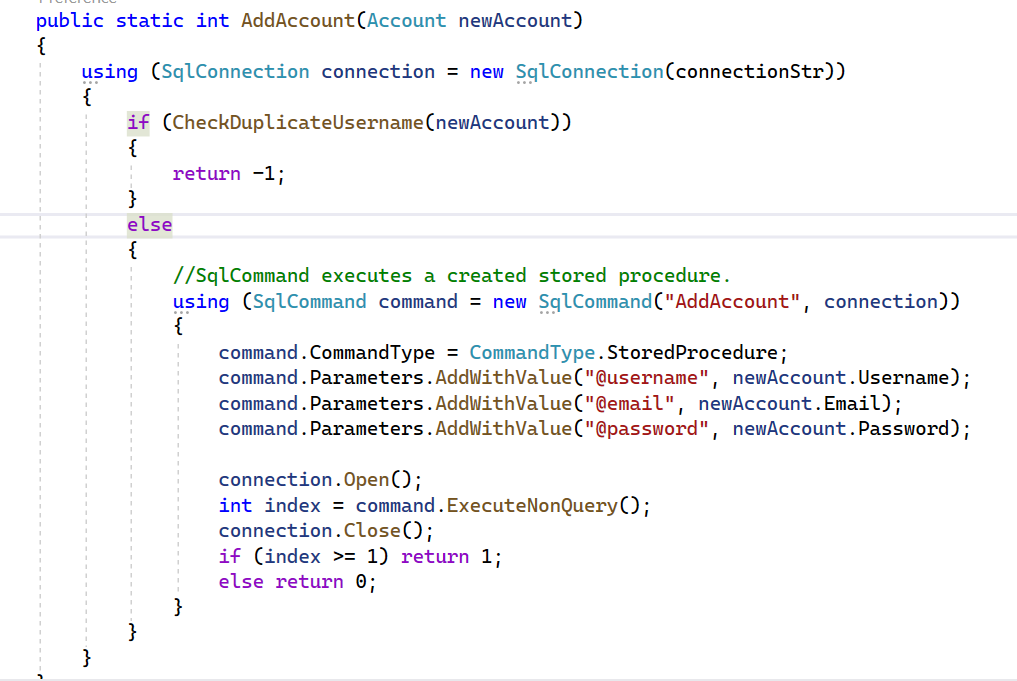
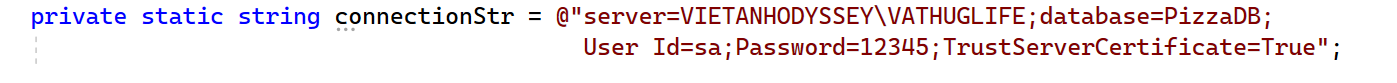
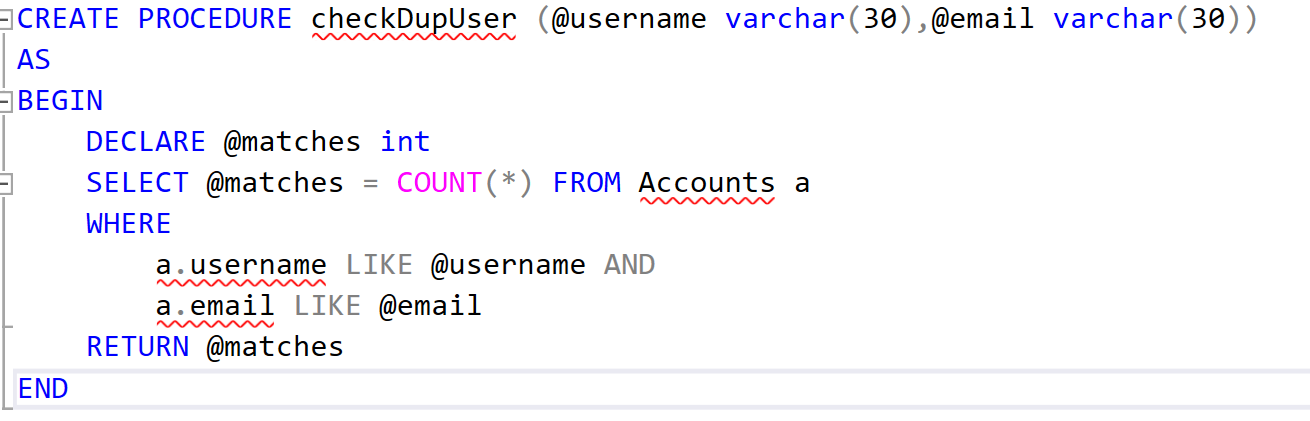
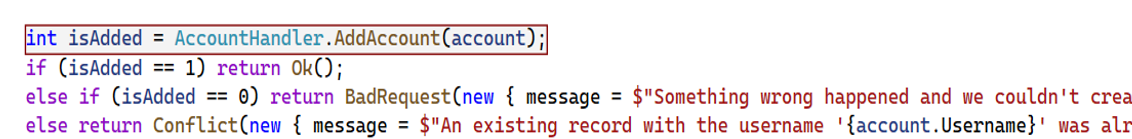
* 1. **Models: Classes that handles the data logic** (e.g. add new account to database, queries the database to find the appropriate data...)
     + 1. **Contexts:** Creates an **INSTANCE** of a database. Simply understood, **an instance is an object**, created from a blueprint (or a class).
          1. **CustomPizzaDbContext:** A **class** that **creates a Database Instance** (or an ENVIRONMENT) for this API (PizzaWeb\_API) **to interact with the tables** (entities) inside the PizzaDB.
       2. **Handlers:** Contains classes that ACTUALLY handles the data logic.
       3. **Objects:** They are basically real-world entities throughout the app (Pizza, Ingredients, Accounts…). Also responsible for model binding (will be explained later).

1. **Deep dive – the Sign-up procedure** 
   1. **Stage 1: The SignupController**
      1. **Overview**
         1. ****
      2. **What it does: Handles client requests (React)** when **creating** **a new user account**, then **sends a Success/Failure response** back.
      3. **How it works:** 
         1. User clicks on the Signup button on the Webpage, after filling in necessary information. This will create a HttpPost request, asking the Server to check and create a new account, if valid.
         2. **The IActionResult Post function,** of **SignupController** will receive this request. The Sign-up details (e.g. username, password, email…) will be passed via the **[FromBody] Account account parameters** **(1).**
         3. After receiving the account details, **SignupController will call the AddAccount function**, from the AccountHandler library, to check if the details are valid.
            1. If yes, a new account will be created, and a success message will be returned to the client.
            2. If no, no new accounts will be created, and a failure message will be returned to the client.

Sidenotes:

* **[FromBody] Account account:** This is what’s called **Model Binding in C#.** Simply put, Model Binding is the act of **mapping (or binding, attaching)** the **data** from the Body of HttpRequest, **with the attributes stated inside the class of the Objects** folder. For example:
  + **First**, the user will send a HttpPost from React. The Post Request should look something like this:
    - 
      * **infoList:** a list of key-value pairs, containing the signup information.
      * **RequestOptions:** this is basically the HTTP Post request. It contains 3 parts:
        + **The request type (GET, POST, …)**
        + **Headers:** containing additional information
        + **Body:** contains the information to be sent. Before being sent, it will be converted into a JSON string, and look something like this:



* + Second, after the POST request from the Client (React), the signup information is received in the Back-end (ASP.Net Core), through the [FromBody] Account account. What this line means is:
    - **[FromBody]:** Tells C# that **the information (username, email, password)** we are going to get is **in the BODY** **section** of the Http Post Request.
    - Account account: Tells C# which model (class) to bind the information to. This is how the class looks like:
    - 
  + Third, when we do so, in the background, ASP.Net Core will automatically **get the value of the key** **from the JSON String**, then binds it to the attribute with the same name of the Model class (Account account). For example:
    - **First, it goes to the key “username” in the JSON String, and gets its value.** Now, we have the value “abc”.
      * 
    - Second, it then checks if there is any attribute called “username” in the Account class. In this case, there is 1 attribute found, and this is it:
      * 
    - Third, it gets the value “abc” from the JSON String, then binds it to the Username attribute. This image should explain the procedure:
      * 
      * 
  + And now, the attribute Username, of class Account, has the value “abc”. The procedure now goes the same for the remaining attributes (email, password)
  + **Dành cho những thím nào lười đọc** 😠**:** Model Bindingcơ bản là gài/gắn dữ liệu nhận được từ bên Client (React) vào thuộc tính của Class (Account Class – ASP.Net Core), dựa trên key của JSON và tên thuộc tính của Class.
* **Stage 2: The AccountHandler class**
  + **Overview:**
    - ****
  + **What it does:**  handles logic of **creating a new account to the SQL Database,** and checking if the **logged in user has existed**.
  + **How it works:**
    - After being called from the SignupController, the AddAccount function will start by **opening a connection to the SQL** **Database**, using a **connection string.**
      * **Connection string:** A string **that contains necessary information (e.g. Instance name, target Database, username, password…)** for ASP.NET Core to connect to a database (in this case, it is SQL Server). Here is the example:
      * ****
    - After successfully connected to the database, the function will check if there is a duplicated account, using the CheckDuplicateUsername function.
    - **If no duplicated user is found,** the function will continue by calling a stored procedure in SQL called AddAccount.
      * A stored procedure is basically a function like in many programming languages, but used in SQL Server. Here is an example:
      * 
    - Next, we pass in that Stored Procedure the username, email, password that we received from the Frontend earlier on, using command.Parameters.AddWithValue.
    - After setting things up, we open the connection between the ASP.NET Core Web API (PizzaWeb\_API) with the SQL Database, using the connection.Open(), then execute the command with ExecuteNonQuery().
    - Index means: returns the result after executing the command. An 1 means the operation was successful, and vice versa.
* **Stage 3: Returning the result to the SignupController**
  + After running the AddAccount function, the result is returned to the SignupController.
  + ****
  + **If an account is created successfully,** a Http Response with Code 200 will be returned to the Client (return Ok()).
  + **If an account has not been created,** an error message will be returned.