Coding Exercise

This exercise is centered around building API endpoints for a banking application. Ideally, it will be built in C# using the ASP.NET Core Web API template, but variations are allowed if it still shows us object-oriented programming over more than one architectural layer.

The API can be a monolith. It should use the repository pattern. It does not need to connect to an actual database; it's acceptable for the repositories to return results in code rather than from a database.

Complete at least the first endpoint documented below, and then do additional ones as time allows. Although this is obviously a small exercise and probably does not need all the things someone would do in an enterprise-level API, please do whatever you would normally do in code as if it were part of a much larger and complex API.

We will assume the code you produce reflects how you think an enterprise level API should be built.

A UI is not necessary, but feel free to build one if you wish. Use whatever resources you would like in completing this exercise except for other developers. This exercise is intended to determine what you would put together so working on a team for this exercise is not allowed.

Endpoint 1: Make a deposit

This endpoint should facilitate a request to deposit a dollar amount in a customer's account.

The endpoint will receive the following JSON:

```
{
      customerId: 5,
      accountId: 17,
      amount: 112.00
}
```

And should return:

```
{
      customerId: 5,
      accountId: 17,
      balance: 2287.13,
      succeeded: true
}
```

The values above are only an example.

Acceptance Criteria

- The balance returned should reflect the current balance after the operation
- The account must exist and belong to that customer.
- The deposit amount must be greater than 0.

Endpoint 2: Make a withdrawal

This endpoint should facilitate a request to withdraw a dollar amount from a customer's account.

The endpoint will receive the following JSON:

```
{
      customerId: 5,
      accountId: 17,
      amount: 112.00
}
```

And should return:

```
{
      customerId: 5,
      accountId: 17,
      balance: 2287.13,
      succeeded: true
}
```

The values above are only an example.

Acceptance Criteria

- The balance returned should reflect the current balance after the operation
- The account must exist and belong to that customer.
- The withdrawal amount must be greater than 0.
- The withdrawal cannot be done if it would bring the balance below 0.

Endpoint 3: Close an account

This endpoint should facilitate a request to close a customer's account.

The endpoint will receive the following JSON:

```
{
      customerId: 5,
      accountId: 17
}
```

And should return:

```
{
      customerId: 5,
      accountId: 17,
      succeeded: true
}
```

The values above are only an example.

Acceptance Criteria

- The account must exist and belong to that customer.
- The account can only be closed if the balance is exactly 0.
- The closure is a change of status, but the account should not actually be deleted.

Endpoint 4: Create an account

This endpoint should facilitate a request to create a new account for a customer.

The endpoint will receive the following JSON:

```
{
      customerId: 5,
      initialDeposit: 525.00,
      accountTypeId: 1
}
```

And should return:

```
{
      customerId: 5,
      accountId: 17,
      accountTypeId: 1,
      balance: 525.00,
      succeeded: true
}
```

The values above are only an example.

Acceptance Criteria

- The customer must exist.
- The initial deposit must be at least 100.
- The account type id should correspond to one of the following values:
    - 1 - Checking

    - 2 - Savings
- If this is the customer’s first account, it must be savings.