# VME Technical Test

The goal of this test is to evaluate your ability in enhancing existing code as well as debugging and fixing bugs.

# Instructions

The provided codebase is written in .NET Core and is structured as follows:

* A REST API which allows users to add, edit, delete and retrieve customers
* A BusinessLogic layer where any business rules and logic will live
* A DataTransferObjects layer where any DTOs are stored
* A repository layer where entities are saved

The following needs to be completed:

* **Implement the GET route to return all customers**

This is found in the CustomersController and is currently returning null. You could return either the whole list or a paginated list of customers. You can modify the route to fit whatever parameters you deem fit to accomplish this.

* **Create a new controller – AccountsController**

This will have the following routes:

1. GET /api/accounts/1

Returns account balance for customer 1

1. POST /api/accounts/1/deposit

{

“funds”: 50

}

Deposits €50 into the account of customer 1

1. POST /api/accounts/1/withdraw

{

“funds”: 50

}

Withdraws 50 from the account of customer 1

1. POST /api/accounts/transfer

{

“from”: 1

“to”: 2

“funds”: 50

}

Transfers €50 from the account of customer 1 to the account of customer 2.

Since balances are sensitive objects, we need to make sure that any operations done on them are thread safe. We also need to make sure that all operations done within one API call are all successful to maintain a proper state.

You are free to create any new classes / projects / methods /DTOs you deem best to implement the operations.

* **Implement a proper read-model**

The current read-model we have implemented is an in-memory database. We need to implement a persistent read-model, whilst still implementing the IReadModel interface. Our suggested framework would be either EventStore, SQL or MongoDB, but you are free to choose any other approach.

* **Audit any balance transactions occurring**

We basically need to know of any transactions which are occurring on a customer’s account. It should be possible to re-build a customer’s balance by going through the audits and re-applying all the transactions. Our suggested approach for this would be to use either EventStore or SQL.

* **Fix bug where customers are ending with a negative balance**

In the original implementation, it is possible for a customer to end up with a negative balance. You’ll need to find where the bug is and fix it.

* **Add some unit tests**

Build a simple unit testing project which would cover code in both the Repository and BusinessLogic layers. Our suggested framework is xUnit, but you are free to use whatever you are most comfortable with.