

# **Fake bank**

AUTHOR  
Version  
Wed Jan 10 2024



# Table of Contents

Table of contents



# BankWebApp

This is a banking web application developed using C#, JavaScript, and SQL.

## Project Structure

The project is structured into several parts:

- `env/Envs.cs` : Contains the connection string for the MSSQL database.
- `database.sql` : Contains the SQL scripts for creating the necessary tables and indices in the database.
- `wwwroot/` : Contains the static files for the web application. (js css images)
- `docs/` : Contains the documentation for the project.
- `BankDB-data/` : Contains the data for the MSSQL database.

## Setup

1. Clone the repository.
2. Ensure that docker & docker-compose is installed.
3. Run `docker-compose up -d` to start the application.
4. Enjoy!.

## How to login to an admin account

1. Open the login page
2. Enter the following credentials:
  - Username: admin
  - Password: admin
3. Login.
4. Admin tools can be accessed by clicking the new buttons in the navbar.

## How to add admin access to a new user

1. Register a new user.
2. Open the database insert a new row into `UserRoles` table
3. Set the `UserId` to the `Id` of the user you want to make admin.
4. Set the `RoleName` to `Admin`.
5. Save the changes.
6. The user should now have admin access.
7. Admin tools can be accessed by clicking the new buttons in the navbar.

## Features

- User registration and login
- Bank account creation
- Transaction processing
- Transaction history
- Admin dashboard
- User dashboard
- Automatic health Checks
- Automatic docker health checks

## API Endpoints

- `/api/healthcheck/all` - Returns the health status of all services and also an overall health status.
- `/api/healthcheck/database` - Returns the health status of the database.
- `/api/healthcheck/disk` - Returns if the disk has enough space.
- `/api/healthcheck/ram` - Returns if the RAM has enough space.

## Database Diagram

## Custom User Components

- `Navbar` : The navigation bar at the top of the page. made with View Component.
- `Footer` : The footer at the bottom of the page. made with Partial View.

## Documentation

The documentation for this project is available in the `docs` folder. Files in the `docs` folder are generated using doxygen.

Available versions of the documentation:

- `PDF Documentation` (same as the rtf version)
- `RTF Documentation` (same as the pdf version, may give warnings about being unsafe)
- `HTML Documentation` (best version, but requires a browser to view)
- `Code Comments` (in the source code)

# The MIT License (MIT)

Copyright Jörn Zaefferer

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

# Namespace Index

## Package List

Here are the packages with brief descriptions (if available):

<b>BankWebApp</b> .....	9
<b>BankWebApp.Components</b> .....	10
<b>BankWebApp.Controllers</b> (HomeController class that inherits from Controller. This class is responsible for handling the requests related to the Home page of the application ) .....	11
<b>BankWebApp.env</b> .....	12
<b>BankWebApp.Models</b> .....	13
<b>BankWebApp.Services</b> (The DatabaseService class is responsible for managing the database connection. It implements the IDisposable interface to properly close the connection when it's no longer needed ) .....	14
<b>BankWebApp.Tools</b> .....	15



# Hierarchical Index

## Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

BankWebApp.Models.AccountHistoryModel .....	20
BankWebApp.Models.AccountIndexModel .....	21
BankWebApp.Models.AddFundsViewModel.....	22
BankWebApp.Models.AddressModel.....	23
BankWebApp.Models.BankAccountModel.....	24
BankWebApp.Models.ContactModel .....	25
Controller	
BankWebApp.Controllers.AccountController .....	16
BankWebApp.Controllers.HomeController .....	36
ControllerBase	
BankWebApp.Controllers.HealthCheckController .....	34
BankWebApp.Models.ErrorViewModel.....	33
IDisposable	
BankWebApp.Services.DatabaseService .....	27
IHealthCheck	
BankWebApp.Services.DatabaseHealthService .....	26
BankWebApp.Services.DiskHealthService .....	32
BankWebApp.Services.MemoryHealthService .....	42
BankWebApp.Models.ListUsersViewModel.....	40
BankWebApp.Models.LoginModel .....	41
BankWebApp.Services.MySignInManager .....	43
BankWebApp.Program .....	46
BankWebApp.Models.RegisterModel.....	47
BankWebApp.Models.RolesModel.....	49
BankWebApp.Models.TransactionModel .....	50
BankWebApp.Services.TransferService .....	52
BankWebApp.Models.TransferViewModel.....	54
BankWebApp.Models.UserModel .....	56
BankWebApp.Services.UserService .....	58
ViewComponent	
BankWebApp.Components.NavbarViewComponent .....	45

# Class Index

## Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<b>BankWebApp.Controllers.AccountController</b> (The AccountController class is responsible for handling requests related to the user's bank account. It includes actions for displaying the account index, transferring funds, adding funds (admin only), listing users (admin only), showing user details (admin only), deleting a user (admin only), and viewing transaction history ) .....	16
<b>BankWebApp.Models.AccountHistoryModel</b> .....	20
<b>BankWebApp.Models.AccountIndexModel</b> .....	21
<b>BankWebApp.Models.AddFundsViewModel</b> .....	22
<b>BankWebApp.Models.AddressModel</b> .....	23
<b>BankWebApp.Models.BankAccountModel</b> .....	24
<b>BankWebApp.Models.ContactModel</b> .....	25
<b>BankWebApp.Services.DatabaseHealthService</b> (Service for checking the health of the database. Implements the Microsoft.Extensions.Diagnostics.HealthChecks.IHealthCheck interface ) .....	26
<b>BankWebApp.Services.DatabaseService</b> (The DatabaseService class is responsible for managing the database operations. It contains methods for getting users, checking if a username exists, registering a user, getting bank accounts by user id, transferring funds, getting roles by user id, getting all bank accounts, adding funds, and getting transactions ) .....	27
<b>BankWebApp.Services.DiskHealthService</b> (Class DiskHealthService. Implements the Microsoft.Extensions.Diagnostics.HealthChecks.IHealthCheck interface. Used to check if the disk has enough free space ) .....	32
<b>BankWebApp.Models.ErrorViewModel</b> .....	33
<b>BankWebApp.Controllers.HealthCheckController</b> (Controller for handling health checks of the application ) .....	34
<b>BankWebApp.Controllers.HomeController</b> .....	36
<b>BankWebApp.Models.ListUsersViewModel</b> .....	40
<b>BankWebApp.Models.LoginModel</b> .....	41
<b>BankWebApp.Services.MemoryHealthService</b> (Class MemoryHealthService. Implements the Microsoft.Extensions.Diagnostics.HealthChecks.IHealthCheck interface. Used to check if the RAM has enough free space ) .....	42
<b>BankWebApp.Services.MySignInManager</b> (This class is responsible for managing user sign in and sign out operations ) .....	43
<b>BankWebApp.Components.NavbarViewComponent</b> .....	45
<b>BankWebApp.Program</b> (The Program class is the entry point of the application ) .....	46
<b>BankWebApp.Models.RegisterModel</b> .....	47
<b>BankWebApp.Models.RolesModel</b> .....	49
<b>BankWebApp.Models.TransactionModel</b> .....	50
<b>BankWebApp.Services.TransferService</b> (This class provides services for transferring money between bank accounts ) .....	52
<b>BankWebApp.Models.TransferViewModel</b> .....	54
<b>BankWebApp.Models.UserModel</b> (Represents a User in the system ) .....	56
<b>BankWebApp.Services.UserService</b> (Service class for managing users ) .....	58

# File Index

## File List

Here is a list of all files with brief descriptions:

C:/Users/tomas/source/repos/BankWebApp/Program.cs .....	114
C:/Users/tomas/source/repos/BankWebApp/Components/NavbarViewComponent.cs .....	62
C:/Users/tomas/source/repos/BankWebApp/Controllers/AccountController.cs .....	64
C:/Users/tomas/source/repos/BankWebApp/Controllers/HealthCheckController.cs .....	69
C:/Users/tomas/source/repos/BankWebApp/Controllers/HomeController.cs .....	72
C:/Users/tomas/source/repos/BankWebApp/env/Envs.cs .....	76
C:/Users/tomas/source/repos/BankWebApp/Models/AccountHistoryModel.cs .....	78
C:/Users/tomas/source/repos/BankWebApp/Models/AccountIndexModel.cs .....	80
C:/Users/tomas/source/repos/BankWebApp/Models/AddFundsViewModel.cs .....	82
C:/Users/tomas/source/repos/BankWebApp/Models/AddressModel.cs .....	84
C:/Users/tomas/source/repos/BankWebApp/Models/BankAccountModel.cs .....	86
C:/Users/tomas/source/repos/BankWebApp/Models/ContactModel.cs .....	88
C:/Users/tomas/source/repos/BankWebApp/Models/ErrorViewModel.cs .....	90
C:/Users/tomas/source/repos/BankWebApp/Models/ListUsersViewModel.cs .....	92
C:/Users/tomas/source/repos/BankWebApp/Models/LoginModel.cs .....	94
C:/Users/tomas/source/repos/BankWebApp/Models/RegisterModel.cs .....	96
C:/Users/tomas/source/repos/BankWebApp/Models/RolesModel.cs .....	98
C:/Users/tomas/source/repos/BankWebApp/Models/TransactionModel.cs .....	100
C:/Users/tomas/source/repos/BankWebApp/Models/TransferViewModel.cs .....	102
C:/Users/tomas/source/repos/BankWebApp/Models/UserModel.cs .....	104
C:/Users/tomas/source/repos/BankWebApp/obj/Debug/net7.0/.NETCoreApp,Version=v7.0.AssemblyAttributes.cs .....	106
C:/Users/tomas/source/repos/BankWebApp/obj/Debug/net7.0/BankWebApp.AssemblyInfo.cs .....	108
C:/Users/tomas/source/repos/BankWebApp/obj/Debug/net7.0/BankWebApp.GlobalUsings.g.cs .....	110
C:/Users/tomas/source/repos/BankWebApp/obj/Debug/net7.0/BankWebApp.RazorAssemblyInfo.cs .....	112
C:/Users/tomas/source/repos/BankWebApp/Services/DatabaseHealthService.cs .....	118
C:/Users/tomas/source/repos/BankWebApp/Services/DatabaseService.cs .....	120
C:/Users/tomas/source/repos/BankWebApp/Services/DatabaseServiceFunctions.cs .....	122
C:/Users/tomas/source/repos/BankWebApp/Services/DiskHealthService.cs .....	129
C:/Users/tomas/source/repos/BankWebApp/Services/MemoryHealthService.cs .....	131
C:/Users/tomas/source/repos/BankWebApp/Services/MySignInManager.cs .....	133
C:/Users/tomas/source/repos/BankWebApp/Services/TransferService.cs .....	135
C:/Users/tomas/source/repos/BankWebApp/Services/UserService.cs .....	137
C:/Users/tomas/source/repos/BankWebApp/Tools/ClaimTools.cs .....	140
C:/Users/tomas/source/repos/BankWebApp/Tools/PasswordHashes.cs .....	142

# Namespace Documentation

## BankWebApp Namespace Reference

### Namespaces

- namespace **Components**
- namespace **Controllers**

*HomeController class that inherits from Controller. This class is responsible for handling the requests related to the Home page of the application.*

- namespace **env**
- namespace **Models**
- namespace **Services**

*The DatabaseService class is responsible for managing the database connection. It implements the IDisposable interface to properly close the connection when it's no longer needed.*

- namespace **Tools**

### Classes

class **Program***The Program class is the entry point of the application.*

## **BankWebApp.Components Namespace Reference**

### **Classes**

class **NavbarViewComponent**

## BankWebApp.Controllers Namespace Reference

HomeController class that inherits from Controller. This class is responsible for handling the requests related to the Home page of the application.

### Classes

class **AccountController***The AccountController class is responsible for handling requests related to the user's bank account. It includes actions for displaying the account index, transferring funds, adding funds (admin only), listing users (admin only), showing user details (admin only), deleting a user (admin only), and viewing transaction history.*

class **HealthCheckController***Controller for handling health checks of the application.*

class **HomeController**

---

### Detailed Description

HomeController class that inherits from Controller. This class is responsible for handling the requests related to the Home page of the application.

## BankWebApp.env Namespace Reference

### Classes

- class **Envs**  
*The Envs static class contains environment variables for the application.*

## BankWebApp.Models Namespace Reference

### Classes

- class **AccountHistoryModel**class **AccountIndexModel**
- class **AddFundsViewModel**
- class **AddressModel**
- class **BankAccountModel**
- class **ContactModel**
- class **ErrorViewModel**
- class **ListUsersViewModel**
- class **LoginModel**
- class **RegisterModel**
- class **RolesModel**
- class **TransactionModel**
- class **TransferViewModel**
- class **UserModel**

*Represents a User in the system.*



## BankWebApp.Services Namespace Reference

The DatabaseService class is responsible for managing the database connection. It implements the IDisposable interface to properly close the connection when it's no longer needed.

### Classes

class **DatabaseHealthService***Service for checking the health of the database. Implements the Microsoft.Extensions.Diagnostics.HealthChecks.IHealthCheck interface.*

class **DatabaseService***The DatabaseService class is responsible for managing the database operations. It contains methods for getting users, checking if a username exists, registering a user, getting bank accounts by user id, transferring funds, getting roles by user id, getting all bank accounts, adding funds, and getting transactions.*

class **DiskHealthService***Class DiskHealthService. Implements the Microsoft.Extensions.Diagnostics.HealthChecks.IHealthCheck interface. Used to check if the disk has enough free space.*

class **MemoryHealthService***Class MemoryHealthService. Implements the Microsoft.Extensions.Diagnostics.HealthChecks.IHealthCheck interface. Used to check if the RAM has enough free space.*

class **MySignInManager***This class is responsible for managing user sign in and sign out operations.*

class **TransferService***This class provides services for transferring money between bank accounts.*

class **UserService***Service class for managing users.*

---

### Detailed Description

The DatabaseService class is responsible for managing the database connection. It implements the IDisposable interface to properly close the connection when it's no longer needed.

## BankWebApp.Tools Namespace Reference

### Classes

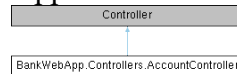
- class **ClaimTools**
- class **PasswordHashes**

# Class Documentation

## BankWebApp.Controllers.AccountController Class Reference

The AccountController class is responsible for handling requests related to the user's bank account. It includes actions for displaying the account index, transferring funds, adding funds (admin only), listing users (admin only), showing user details (admin only), deleting a user (admin only), and viewing transaction history.

Inheritance diagram for BankWebApp.Controllers.AccountController:



### Public Member Functions

- **AccountController** (ILogger< **AccountController** > logger, **UserService** userService, **TransferService** transferService)  
*Initializes a new instance of the AccountController class.*
- IActionResult **Index** ()  
*Displays the account index page.*
- IActionResult **Transfer** (bool? success=null, string? reason=null)  
*Displays the transfer page.*
- IActionResult **Transfer** (**TransferViewModel** model)  
*Handles a POST request to transfer funds.*
- IActionResult **AddFunds** (bool? success=null, string? reason=null)  
*Displays the add funds page (admin only).*
- IActionResult **AddFunds** (**AddFundsViewModel** model)  
*Handles a POST request to add funds to an account (admin only).*
- IActionResult **ListUsers** ()  
*Displays the list users page (admin only).*
- IActionResult **Show** (string id)  
*Displays the details of a user (admin only).*
- IActionResult **History** ()  
*Displays the transaction history page.*

---

### Detailed Description

The AccountController class is responsible for handling requests related to the user's bank account. It includes actions for displaying the account index, transferring funds, adding funds

(admin only), listing users (admin only), showing user details (admin only), deleting a user (admin only), and viewing transaction history.

Definition at line 14 of file **AccountController.cs**.

---

## Constructor & Destructor Documentation

**BankWebApp.Controllers.AccountController.AccountController (ILogger< AccountController > *logger*, UserService *userService*, TransferService *transferService*)**

Initializes a new instance of the AccountController class.

### Parameters

<i>logger</i>	The logger used to log information about program execution.
<i>userService</i>	The service used to interact with user data.
<i>transferService</i>	The service used to handle money transfers.

Definition at line 26 of file **AccountController.cs**.

---

## Member Function Documentation

**IActionResult BankWebApp.Controllers.AccountController.AddFunds (AddFundsViewModel *model*)**

Handles a POST request to add funds to an account (admin only).

### Parameters

<i>model</i>	The data from the add funds form.
--------------	-----------------------------------

### Returns

A redirect to the add funds page, with success and reason parameters.

Definition at line 174 of file **AccountController.cs**.

**IActionResult BankWebApp.Controllers.AccountController.AddFunds (bool? *success* = null, string? *reason* = null)**

Displays the add funds page (admin only).

### Parameters

<i>success</i>	Indicates whether the last add funds operation was successful.
<i>reason</i>	The reason for the last add funds operation's failure, if it failed.

### Returns

The add funds view.

Definition at line 156 of file **AccountController.cs**.

**IActionResult BankWebApp.Controllers.AccountController.History ()**

Displays the transaction history page.

#### Returns

The history view.

Definition at line **263** of file **AccountController.cs**.

#### **ActionResult BankWebApp.Controllers.AccountController.Index ()**

Displays the account index page.

#### Returns

The account index view.

Definition at line **38** of file **AccountController.cs**.

#### **ActionResult BankWebApp.Controllers.AccountController.ListUsers ()**

Displays the list users page (admin only).

#### Returns

The list users view.

Definition at line **209** of file **AccountController.cs**.

#### **ActionResult BankWebApp.Controllers.AccountController.Show (string *id*)**

Displays the details of a user (admin only).

#### Parameters

<i>id</i>	The ID of the user to show.
-----------	-----------------------------

#### Returns

The show view.

Definition at line **235** of file **AccountController.cs**.

#### **ActionResult BankWebApp.Controllers.AccountController.Transfer (bool? *success* = null, string? *reason* = null)**

Displays the transfer page.

#### Parameters

<i>success</i>	Indicates whether the last transfer was successful.
<i>reason</i>	The reason for the last transfer's failure, if it failed.

#### Returns

The transfer view.

Definition at line **65** of file **AccountController.cs**.

## **ActionResult BankWebApp.Controllers.AccountController.Transfer (TransferViewModel *model*)**

Handles a POST request to transfer funds.

### **Parameters**

<i>model</i>	The data from the transfer form.
--------------	----------------------------------

### **Returns**

A redirect to the transfer page, with success and reason parameters.

Definition at line **85** of file **AccountController.cs**.

---

**The documentation for this class was generated from the following file:**

- C:/Users/tomas/source/repos/BankWebApp/Controllers/AccountController.cs

## BankWebApp.Models.AccountHistoryModel Class Reference

### Properties

- `IList<TransactionModel> Transactions` [get, set]
- 

### Detailed Description

Definition at line 3 of file `AccountHistoryModel.cs`.

---

### Property Documentation

`IList<TransactionModel>`

`BankWebApp.Models.AccountHistoryModel.Transactions` [get], [set]

Definition at line 5 of file `AccountHistoryModel.cs`.

---

The documentation for this class was generated from the following file:

- `C:/Users/tomas/source/repos/BankWebApp/Models/AccountHistoryModel.cs`

## BankWebApp.Models.AccountIndexModel Class Reference

### Properties

- **UserModel SignedInUser** [get, set]
  - **ICollection< BankAccountModel > BankAccounts** [get, set]
- 

### Detailed Description

Definition at line 3 of file **AccountIndexModel.cs**.

---

### Property Documentation

**ICollection<BankAccountModel>**

**BankWebApp.Models.AccountIndexModel.BankAccounts** [get], [set]

Definition at line 6 of file **AccountIndexModel.cs**.

**UserModel BankWebApp.Models.AccountIndexModel.SignedInUser** [get], [set]

Definition at line 5 of file **AccountIndexModel.cs**.

---

The documentation for this class was generated from the following file:

- C:/Users/tomas/source/repos/BankWebApp/Models/AccountIndexModel.cs



## BankWebApp.Models.AddFundsViewModel Class Reference

### Properties

- `IList<BankAccountModel> BankAccounts` [get, set]
  - `decimal Amount` [get, set]
  - `string SelectedBankAccountNumber` [get, set]
  - `bool? Success` [get, set]
  - `string? Reason` [get, set]
- 

### Detailed Description

Definition at line 3 of file `AddFundsViewModel.cs`.

---

### Property Documentation

**decimal** `BankWebApp.Models.AddFundsViewModel.Amount` [get], [set]

Definition at line 7 of file `AddFundsViewModel.cs`.

**IList<BankAccountModel>**

**BankWebApp.Models.AddFundsViewModel.BankAccounts** [get], [set]

Definition at line 5 of file `AddFundsViewModel.cs`.

**string?** `BankWebApp.Models.AddFundsViewModel.Reason` [get], [set]

Definition at line 11 of file `AddFundsViewModel.cs`.

**string**

**BankWebApp.Models.AddFundsViewModel.SelectedBankAccountNumber** [get], [set]

Definition at line 8 of file `AddFundsViewModel.cs`.

**bool?** `BankWebApp.Models.AddFundsViewModel.Success` [get], [set]

Definition at line 10 of file `AddFundsViewModel.cs`.

---

The documentation for this class was generated from the following file:

- `C:/Users/tomas/source/repos/BankWebApp/Models/AddFundsViewModel.cs`

## BankWebApp.Models.AddressModel Class Reference

### Properties

- `int Id` [get, set]
  - `string Street` [get, set]
  - `string City` [get, set]
  - `string PostCode` [get, set]
  - `string Country` [get, set]
- 

### Detailed Description

Definition at line 3 of file `AddressModel.cs`.

---

### Property Documentation

**`string BankWebApp.Models.AddressModel.City`** [get], [set]

Definition at line 7 of file `AddressModel.cs`.

**`string BankWebApp.Models.AddressModel.Country`** [get], [set]

Definition at line 9 of file `AddressModel.cs`.

**`int BankWebApp.Models.AddressModel.Id`** [get], [set]

Definition at line 5 of file `AddressModel.cs`.

**`string BankWebApp.Models.AddressModel.PostCode`** [get], [set]

Definition at line 8 of file `AddressModel.cs`.

**`string BankWebApp.Models.AddressModel.Street`** [get], [set]

Definition at line 6 of file `AddressModel.cs`.

---

The documentation for this class was generated from the following file:

- `C:/Users/tomas/source/repos/BankWebApp/Models/AddressModel.cs`

## BankWebApp.Models.BankAccountModel Class Reference

### Properties

- `int Id` [get, set]
  - `string AccountNumber` [get, set]
  - `decimal Balance` [get, set]
  - `int UserId` [get, set]
  - `UserModel User` [get, set]
- 

### Detailed Description

Definition at line 3 of file **BankAccountModel.cs**.

---

### Property Documentation

**string BankWebApp.Models.BankAccountModel.AccountNumber** [get], [set]

Definition at line 6 of file **BankAccountModel.cs**.

**decimal BankWebApp.Models.BankAccountModel.Balance** [get], [set]

Definition at line 7 of file **BankAccountModel.cs**.

**int BankWebApp.Models.BankAccountModel.Id** [get], [set]

Definition at line 5 of file **BankAccountModel.cs**.

**UserModel BankWebApp.Models.BankAccountModel.User** [get], [set]

Definition at line 9 of file **BankAccountModel.cs**.

**int BankWebApp.Models.BankAccountModel.UserId** [get], [set]

Definition at line 8 of file **BankAccountModel.cs**.

---

The documentation for this class was generated from the following file:

- `C:/Users/tomas/source/repos/BankWebApp/Models/BankAccountModel.cs`

## BankWebApp.Models.ContactModel Class Reference

### Properties

- `int Id` [get, set]
  - `string Email` [get, set]
  - `string PhoneNumber` [get, set]
- 

### Detailed Description

Definition at line 3 of file `ContactModel.cs`.

---

### Property Documentation

**`string BankWebApp.Models.ContactModel.Email`** [get], [set]

Definition at line 6 of file `ContactModel.cs`.

**`int BankWebApp.Models.ContactModel.Id`** [get], [set]

Definition at line 5 of file `ContactModel.cs`.

**`string BankWebApp.Models.ContactModel.PhoneNumber`** [get], [set]

Definition at line 7 of file `ContactModel.cs`.

---

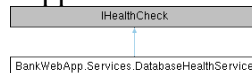
The documentation for this class was generated from the following file:

- `C:/Users/tomas/source/repos/BankWebApp/Models/ContactModel.cs`

# BankWebApp.Services.DatabaseHealthService Class Reference

Service for checking the health of the database. Implements the Microsoft.Extensions.Diagnostics.HealthChecks.IHealthCheck interface.

Inheritance diagram for BankWebApp.Services.DatabaseHealthService:



## Public Member Functions

- Task< HealthCheckResult > **CheckHealthAsync** (HealthCheckContext context, CancellationToken cancellationToken=new CancellationToken())  
*Asynchronously checks the health of the database.*

---

## Detailed Description

Service for checking the health of the database. Implements the Microsoft.Extensions.Diagnostics.HealthChecks.IHealthCheck interface.

Definition at line 9 of file **DatabaseHealthService.cs**.

---

## Member Function Documentation

**Task< HealthCheckResult >**  
**BankWebApp.Services.DatabaseHealthService.CheckHealthAsync**  
(HealthCheckContext *context*, CancellationToken *cancellationToken* =  
new CancellationToken())

Asynchronously checks the health of the database.

### Parameters

<i>context</i>	The context under which the health check is being performed.
<i>cancellationToken</i>	A System.Threading.CancellationToken that can be used to cancel the health check.

### Returns

A System.Threading.Tasks.Task that represents the asynchronous operation, containing the Microsoft.Extensions.Diagnostics.HealthChecks.HealthCheckResult of the database health check.

Definition at line 17 of file **DatabaseHealthService.cs**.

---

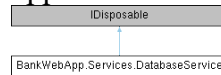
The documentation for this class was generated from the following file:

- C:/Users/tomas/source/repos/BankWebApp/Services/**DatabaseHealthService.cs**

## BankWebApp.Services.DatabaseService Class Reference

The DatabaseService class is responsible for managing the database operations. It contains methods for getting users, checking if a username exists, registering a user, getting bank accounts by user id, transferring funds, getting roles by user id, getting all bank accounts, adding funds, and getting transactions.

Inheritance diagram for BankWebApp.Services.DatabaseService:



### Public Member Functions

- **DatabaseService ()**  
*The constructor initializes a new instance of the DatabaseService class. It sets the connection string and opens the connection.*
- **void Dispose ()**  
*The Dispose method is called when the DatabaseService object is being disposed. It closes the database connection.*
- **IList< UserModel > GetUsers ()**  
*Gets all users from the database.*
- **bool UsernameExists (string \_username)**  
*Gets all users from the database.*
- **bool RegisterUser (UserModel user)**  
*Registers a new user in the database.*
- **IList< BankAccountModel >? GetBankAccountById (int UserId)**  
*Gets a list of bank accounts by user id.*
- **BankAccountModel? GetBankAccountById (string Id)**  
*Gets a bank account by account number.*
- **BankAccountModel? GetBankAccountByAccountId (int Id)**  
*Gets a bank account by account id.*
- **bool TransferFunds (Guid from, Guid To, decimal Amount)**  
*Transfers funds from one account to another.*
- **IList< RolesModel > GetRolesById (int uid)**  
*Gets a list of roles by user id.*
- **IList< BankAccountModel > GetAllBankAccounts ()**  
*Gets all bank accounts from the database.*
- **void AddFunds (Guid guid, decimal amount)**  
*Adds funds to a bank account.*

- **ICollection< TransactionModel > GetTransactions ()**  
*Gets all transactions from the database.*
- **ICollection< TransactionModel > GetTransactions (int uid)**  
*Gets a list of transactions by user id.*
- **bool Ping ()**  
*A helper method for checking if the database is alive. Mainly used for health checks.*

---

## Detailed Description

The DatabaseService class is responsible for managing the database operations. It contains methods for getting users, checking if a username exists, registering a user, getting bank accounts by user id, transferring funds, getting roles by user id, getting all bank accounts, adding funds, and getting transactions.

Definition at line **10** of file **DatabaseService.cs**.

---

## Constructor & Destructor Documentation

### BankWebApp.Services.DatabaseService.DatabaseService ()

The constructor initializes a new instance of the DatabaseService class. It sets the connection string and opens the connection.

Definition at line **26** of file **DatabaseService.cs**.

---

## Member Function Documentation

### void BankWebApp.Services.DatabaseService.AddFunds (Guid *guid*, decimal *amount*)

Adds funds to a bank account.

#### Parameters

<i>guid</i>	The account number to add funds to.
<i>amount</i>	The amount to add.

Definition at line **360** of file **DatabaseServiceFunctions.cs**.

### void BankWebApp.Services.DatabaseService.Dispose ()

The Dispose method is called when the DatabaseService object is being disposed. It closes the database connection.

Definition at line **54** of file **DatabaseService.cs**.

**IList< BankAccountModel >**  
**BankWebApp.Services.DatabaseService.GetAllBankAccounts ()**

Gets all bank accounts from the database.

**Returns**

A list of BankAccountModel objects.

Definition at line 327 of file **DatabaseServiceFunctions.cs**.

**BankAccountModel?**  
**BankWebApp.Services.DatabaseService.GetBankAccountById (int *Id*)**

Gets a bank account by account id.

**Parameters**

<i>Id</i>	The account id to get the bank account for.
-----------	---

**Returns**

A BankAccountModel object.

Definition at line 213 of file **DatabaseServiceFunctions.cs**.

**IList< BankAccountModel >?**  
**BankWebApp.Services.DatabaseService.GetBankAccountByld (int *UserId*)**

Gets a list of bank accounts by user id.

**Parameters**

<i>UserId</i>	The user id to get the bank accounts for.
---------------	---

**Returns**

A list of BankAccountModel objects.

Definition at line 142 of file **DatabaseServiceFunctions.cs**.

**BankAccountModel? BankWebApp.Services.DatabaseService.GetBankAccountByld (string *Id*)**

Gets a bank account by account number.

**Parameters**

<i>Id</i>	The account number to get the bank account for.
-----------	---

**Returns**

A BankAccountModel object.

Definition at line 179 of file **DatabaseServiceFunctions.cs**.

**IList< RolesModel > BankWebApp.Services.DatabaseService.GetRolesByld (int *uid*)**

Gets a list of roles by user id.



### Parameters

<i>uid</i>	The user id to get the roles for.
------------	-----------------------------------

### Returns

A list of RolesModel objects.

Definition at line **302** of file **DatabaseServiceFunctions.cs**.

## **ICollection< TransactionModel > BankWebApp.Services.DatabaseService.GetTransactions ()**

Gets all transactions from the database.

### Returns

A list of TransactionModel objects.

Definition at line **374** of file **DatabaseServiceFunctions.cs**.

## **ICollection< TransactionModel > BankWebApp.Services.DatabaseService.GetTransactions (int uid)**

Gets a list of transactions by user id.

### Parameters

<i>uid</i>	The user id to get the transactions for.
------------	--

### Returns

A list of TransactionModel objects.

Definition at line **411** of file **DatabaseServiceFunctions.cs**.

## **ICollection< UserModel > BankWebApp.Services.DatabaseService.GetUsers ()**

Gets all users from the database.

### Returns

A list of UserModel objects.

Definition at line **19** of file **DatabaseServiceFunctions.cs**.

## **bool BankWebApp.Services.DatabaseService.Ping ()**

A helper method for checking if the database is alive. Mainly used for health checks.

### Returns

True if the database is alive, false otherwise.

Definition at line **450** of file **DatabaseServiceFunctions.cs**.

## **bool BankWebApp.Services.DatabaseService.RegisterUser (UserModel user)**

Registers a new user in the database.

### Parameters

<i>user</i>	The user to register.
-------------	-----------------------

### Returns

True if the registration was successful, false otherwise.

Definition at line **76** of file **DatabaseServiceFunctions.cs**.

**bool BankWebApp.Services.DatabaseService.TransferFunds (Guid *from*, Guid *To*, decimal *Amount*)**

Transfers funds from one account to another.

### Parameters

<i>from</i>	The account number to transfer funds from.
<i>To</i>	The account number to transfer funds to.
<i>Amount</i>	The amount to transfer.

### Returns

True if the transfer was successful, false otherwise.

Definition at line **251** of file **DatabaseServiceFunctions.cs**.

**bool BankWebApp.Services.DatabaseService.UsernameExists (string *\_username*)**

Gets all users from the database.

### Returns

A list of UserModel objects.

Definition at line **61** of file **DatabaseServiceFunctions.cs**.

---

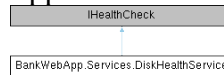
**The documentation for this class was generated from the following files:**

- C:/Users/tomas/source/repos/BankWebApp/Services/**DatabaseService.cs**
- C:/Users/tomas/source/repos/BankWebApp/Services/**DatabaseServiceFunctions.cs**

## BankWebApp.Services.DiskHealthService Class Reference

Class DiskHealthService. Implements the Microsoft.Extensions.Diagnostics.HealthChecks.IHealthCheck interface. Used to check if the disk has enough free space.

Inheritance diagram for BankWebApp.Services.DiskHealthService:



### Public Member Functions

- Task< HealthCheckResult > **CheckHealthAsync** (HealthCheckContext context, CancellationToken cancellationToken)  
*Checks the remaining space on the disk asynchronously.*

---

### Detailed Description

Class DiskHealthService. Implements the Microsoft.Extensions.Diagnostics.HealthChecks.IHealthCheck interface. Used to check if the disk has enough free space.

Definition at line 10 of file **DiskHealthService.cs**.

---

### Member Function Documentation

**Task< HealthCheckResult >**

**BankWebApp.Services.DiskHealthService.CheckHealthAsync** (HealthCheckContext context, CancellationToken cancellationToken)

Checks the remaining space on the disk asynchronously.

#### Parameters

<i>context</i>	The context.
<i>cancellationToken</i>	The cancellation token.

#### Returns

A Task representing the asynchronous operation. The Task result contains the HealthCheckResult.

#### Exceptions

<i>Exception</i>	Thrown when failed to check disk health.
------------------	--

Definition at line 34 of file **DiskHealthService.cs**.

---

The documentation for this class was generated from the following file:

- C:/Users/tomas/source/repos/BankWebApp/Services/**DiskHealthService.cs**

## BankWebApp.Models.ErrorViewModel Class Reference

### Properties

- `string? RequestId` [get, set]
  - `bool ShowRequestId` [get]
- 

### Detailed Description

Definition at line 3 of file **ErrorViewModel.cs**.

---

### Property Documentation

**string? BankWebApp.Models.ErrorViewModel.RequestId** [get], [set]

Definition at line 5 of file **ErrorViewModel.cs**.

**bool BankWebApp.Models.ErrorViewModel.ShowRequestId** [get]

Definition at line 7 of file **ErrorViewModel.cs**.

---

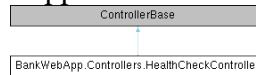
The documentation for this class was generated from the following file:

- `C:/Users/tomas/source/repos/BankWebApp/Models/ErrorViewModel.cs`

# BankWebApp.Controllers.HealthCheckController Class Reference

Controller for handling health checks of the application.

Inheritance diagram for BankWebApp.Controllers.HealthCheckController:



## Public Member Functions

- **HealthCheckController** (ILogger< **HealthCheckController** > logger, **DatabaseHealthService** dbService, **DiskHealthService** diskService, **MemoryHealthService** memoryService)  
*Initializes a new instance of the HealthCheckController class.*
- async Task< IActionResult > **All** ()  
*Gets the health status of all components. If any component is unhealthy, the overall status will be unhealthy. If any component is degraded, the overall status will be degraded.*
- async Task< IActionResult > **Database** ()  
*Gets the health status of the database.*
- async Task< IActionResult > **Disk** ()  
*Gets the health status of the disk.*
- async Task< IActionResult > **RAM** ()  
*Gets the health status of the RAM.*

---

## Detailed Description

Controller for handling health checks of the application.

Definition at line **12** of file **HealthCheckController.cs**.

---

## Constructor & Destructor Documentation

**BankWebApp.Controllers.HealthCheckController.HealthCheckController** (ILogger< **HealthCheckController** > *logger*, **DatabaseHealthService** *dbService*, **DiskHealthService** *diskService*, **MemoryHealthService** *memoryService*)

Initializes a new instance of the HealthCheckController class.

### Parameters

<i>logger</i>	The logger used to log information about program execution.
<i>dbService</i>	The service used to check the health of the database.
<i>diskService</i>	The service used to check the health of the disk.
<i>memoryService</i>	The service used to check the health of the memory.

Definition at line **29** of file **HealthCheckController.cs**.

---

## Member Function Documentation

### **async Task< IActionResult > BankWebApp.Controllers.HealthCheckController.All ()**

Gets the health status of all components. If any component is unhealthy, the overall status will be unhealthy. If any component is degraded, the overall status will be degraded.

#### **Returns**

The health status of all components as a JSON object.

Definition at line **48** of file **HealthCheckController.cs**.

### **async Task< IActionResult > BankWebApp.Controllers.HealthCheckController.Database ()**

Gets the health status of the database.

#### **Returns**

The health status of the database as a JSON object.

Definition at line **109** of file **HealthCheckController.cs**.

### **async Task< IActionResult > BankWebApp.Controllers.HealthCheckController.Disk ()**

Gets the health status of the disk.

#### **Returns**

The health status of the disk as a JSON object.

Definition at line **119** of file **HealthCheckController.cs**.

### **async Task< IActionResult > BankWebApp.Controllers.HealthCheckController.RAM ()**

Gets the health status of the RAM.

#### **Returns**

The health status of the RAM as a JSON object.

Definition at line **129** of file **HealthCheckController.cs**.

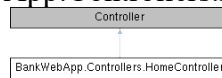
---

**The documentation for this class was generated from the following file:**

- C:/Users/tomas/source/repos/BankWebApp/Controllers/HealthCheckController.cs

# BankWebApp.Controllers.HomeController Class Reference

Inheritance diagram for BankWebApp.Controllers.HomeController:



## Public Member Functions

- **HomeController** (ILogger< **HomeController** > logger, **UserService** userService, **MySignInManager** signInManager)  
*HomeController constructor. Initializes a new instance of the HomeController class.*
- IActionResult **Index** ()  
*Handles the GET request for the Index view.*
- IActionResult **Privacy** ()  
*Handles the GET request for the Privacy view.*
- IActionResult **Login** ()  
*Handles the GET request for the Login view.*
- IActionResult **Login** (**LoginModel** loginModel)  
*Handles the POST request for the Login view.*
- IActionResult **Logout** ()  
*Handles the request to log out the user.*
- IActionResult **Error** ()  
*Handles the GET request for the Error view.*
- IActionResult **AccessDenied** ()  
*Handles the request when the user is denied access.*
- IActionResult **Register** ()  
*Handles the GET request for the Register view.*
- IActionResult **Register** (**RegisterModel** registerModel)  
*Handles the POST request for the Register view.*

---

## Detailed Description

Definition at line 13 of file **HomeController.cs**.

---

## Constructor & Destructor Documentation

**BankWebApp.Controllers.HomeController.HomeController (ILogger< HomeController > *logger*, UserService *userService*, MySignInManager *signInManager*)**

HomeController constructor. Initializes a new instance of the HomeController class.

### Parameters

<i>logger</i>	An instance of ILogger interface to handle logging.
<i>userService</i>	An instance of UserService to handle user related operations.
<i>signInManager</i>	An instance of MySignInManager to handle user sign in operations.

Definition at line 26 of file **HomeController.cs**.

---

## Member Function Documentation

**ActionResult BankWebApp.Controllers.HomeController.AccessDenied ()**

Handles the request when the user is denied access.

### Returns

The Error view along with the request id.

Definition at line 132 of file **HomeController.cs**.

**ActionResult BankWebApp.Controllers.HomeController.Error ()**

Handles the GET request for the Error view.

### Returns

The Error view along with the request id.

Definition at line 123 of file **HomeController.cs**.

**ActionResult BankWebApp.Controllers.HomeController.Index ()**

Handles the GET request for the Index view.

### Returns

The Index view.

Definition at line 37 of file **HomeController.cs**.

**ActionResult BankWebApp.Controllers.HomeController.Login ()**

Handles the GET request for the Login view.

### Returns

The Login view.



Definition at line **55** of file **HomeController.cs**.

### **ActionResult BankWebApp.Controllers.HomeController.Login (LoginModel loginModel)**

Handles the POST request for the Login view.

#### **Parameters**

<i>loginModel</i>	The login details provided by the user.
-------------------	---

#### **Returns**

The Login view if the model state is invalid, otherwise redirects to the appropriate view based on the login details.

Definition at line **66** of file **HomeController.cs**.

### **ActionResult BankWebApp.Controllers.HomeController.Logout ()**

Handles the request to log out the user.

#### **Returns**

Redirects to the Login view after successfully logging out the user.

Definition at line **106** of file **HomeController.cs**.

### **ActionResult BankWebApp.Controllers.HomeController.Privacy ()**

Handles the GET request for the Privacy view.

#### **Returns**

The Privacy view.

Definition at line **46** of file **HomeController.cs**.

### **ActionResult BankWebApp.Controllers.HomeController.Register ()**

Handles the GET request for the Register view.

#### **Returns**

The Register view.

Definition at line **141** of file **HomeController.cs**.

### **ActionResult BankWebApp.Controllers.HomeController.Register (RegisterModel registerModel)**

Handles the POST request for the Register view.

#### **Parameters**

<i>registerModel</i>	The registration details provided by the user.
----------------------	--

### Returns

The Register view if the model state is invalid, otherwise redirects to the Login view after successful registration.

Definition at line **152** of file **HomeController.cs**.

---

**The documentation for this class was generated from the following file:**

- C:/Users/tomas/source/repos/BankWebApp/Controllers/**HomeController.cs**

## BankWebApp.Models.ListUsersViewModel Class Reference

### Properties

- **UserModel** **UserModel** [get, set]
  - **IList<BankAccountModel>** **BankAccounts** [get, set]
  - **IList<TransactionModel>** **Transactions** [get, set]
- 

### Detailed Description

Definition at line 3 of file **ListUsersViewModel.cs**.

---

### Property Documentation

**IList<BankAccountModel>**

**BankWebApp.Models.ListUsersViewModel.BankAccounts** [get], [set]

Definition at line 6 of file **ListUsersViewModel.cs**.

**IList<TransactionModel>**

**BankWebApp.Models.ListUsersViewModel.Transactions** [get], [set]

Definition at line 8 of file **ListUsersViewModel.cs**.

**UserModel** **BankWebApp.Models.ListUsersViewModel.UserModel** [get], [set]

Definition at line 5 of file **ListUsersViewModel.cs**.

---

The documentation for this class was generated from the following file:

- C:/Users/tomas/source/repos/BankWebApp/Models/**ListUsersViewModel.cs**

## BankWebApp.Models.LoginModel Class Reference

### Properties

- string **Username** [get, set]
  - string **Password** [get, set]
  - bool **RememberMe** [get, set]
- 

### Detailed Description

Definition at line 3 of file **LoginModel.cs**.

---

### Property Documentation

**string BankWebApp.Models.LoginModel.Password** [get], [set]

Definition at line 6 of file **LoginModel.cs**.

**bool BankWebApp.Models.LoginModel.RememberMe** [get], [set]

Definition at line 7 of file **LoginModel.cs**.

**string BankWebApp.Models.LoginModel.Username** [get], [set]

Definition at line 5 of file **LoginModel.cs**.

---

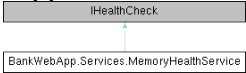
The documentation for this class was generated from the following file:

- C:/Users/tomas/source/repos/BankWebApp/Models/**LoginModel.cs**

# BankWebApp.Services.MemoryHealthService Class Reference

Class MemoryHealthService. Implements the Microsoft.Extensions.Diagnostics.HealthChecks.IHealthCheck interface. Used to check if the RAM has enough free space.

Inheritance diagram for BankWebApp.Services.MemoryHealthService:



## Public Member Functions

- Task< HealthCheckResult > **CheckHealthAsync** (HealthCheckContext context, CancellationToken cancellationToken)  
*Checks the remaining space in the memory asynchronously.*

---

## Detailed Description

Class MemoryHealthService. Implements the Microsoft.Extensions.Diagnostics.HealthChecks.IHealthCheck interface. Used to check if the RAM has enough free space.

Definition at line 11 of file MemoryHealthService.cs.

---

## Member Function Documentation

**Task< HealthCheckResult > BankWebApp.Services.MemoryHealthService.CheckHealthAsync (HealthCheckContext context, CancellationToken cancellationToken)**

Checks the remaining space in the memory asynchronously.

### Parameters

<i>context</i>	The context.
<i>cancellationToken</i>	The cancellation token.

### Returns

A Task representing the asynchronous operation. The Task result contains the HealthCheckResult.

### Exceptions

<i>Exception</i>	Thrown when failed to check memory health.
------------------	--

Definition at line 32 of file MemoryHealthService.cs.

---

The documentation for this class was generated from the following file:

- C:/Users/tomas/source/repos/BankWebApp/Services/MemoryHealthService.cs

## BankWebApp.Services.MySignInManager Class Reference

This class is responsible for managing user sign in and sign out operations.

### Public Member Functions

- **MySignInManager** (IHttpContextAccessor httpContextAccessor, **UserService** userService)  
*Initializes a new instance of the MySignInManager class.*
- async Task **SignInAsync** (**UserModel** user, bool isPersistent=false)  
*Signs in the specified user.*
- async Task **SignOutAsync** ()  
*Signs out the current user.*

---

### Detailed Description

This class is responsible for managing user sign in and sign out operations.

Definition at line **10** of file **MySignInManager.cs**.

---

### Constructor & Destructor Documentation

**BankWebApp.Services.MySignInManager.MySignInManager** (IHttpContextAccessor *httpContextAccessor*, **UserService** *userService*)

Initializes a new instance of the MySignInManager class.

#### Parameters

<i>httpContextAccess</i> or <i>userService</i>	The HTTP context accessor.
<i>userService</i>	The user service.

Definition at line **20** of file **MySignInManager.cs**.

---

### Member Function Documentation

async Task **BankWebApp.Services.MySignInManager.SignInAsync** (**UserModel** *user*, bool *isPersistent* = false)

Signs in the specified user.

#### Parameters

<i>user</i>	The user to sign in.
<i>isPersistent</i>	if set to true the sign in is persistent.

**Returns**

A Task representing the asynchronous operation.

The user model should already be validated before calling this method.

Definition at line **35** of file **MySignInManager.cs**.

**async Task BankWebApp.Services.MySignInManager.SignOutAsync ()**

Signs out the current user.

**Returns**

A Task representing the asynchronous operation.

Definition at line **68** of file **MySignInManager.cs**.

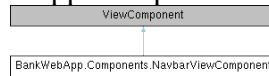
---

**The documentation for this class was generated from the following file:**

- C:/Users/tomas/source/repos/BankWebApp/Services/MySignInManager.cs

## BankWebApp.Components.NavbarViewComponent Class Reference

Inheritance diagram for BankWebApp.Components.NavbarViewComponent:



### Public Member Functions

- `IViewComponentResult Invoke ()`

---

### Detailed Description

Definition at line 5 of file `NavbarViewComponent.cs`.

---

### Member Function Documentation

**`IViewComponentResult BankWebApp.Components.NavbarViewComponent.Invoke ()`**

Definition at line 7 of file `NavbarViewComponent.cs`.

---

The documentation for this class was generated from the following file:

- `C:/Users/tomas/source/repos/BankWebApp/Components/NavbarViewComponent.cs`



## BankWebApp.Program Class Reference

The Program class is the entry point of the application.

### Static Public Member Functions

- static void **Main** (string[] args)  
*The Main method is responsible for setting up and running the web application.*

---

### Detailed Description

The Program class is the entry point of the application.

Definition at line 9 of file **Program.cs**.

---

### Member Function Documentation

**static void BankWebApp.Program.Main** (string[] *args*) [*static*]

The Main method is responsible for setting up and running the web application.

#### Parameters

<i>args</i>	Command-line arguments passed to the application.
-------------	---

Definition at line 15 of file **Program.cs**.

---

The documentation for this class was generated from the following file:

- C:/Users/tomas/source/repos/BankWebApp/**Program.cs**

## BankWebApp.Models.RegisterModel Class Reference

### Properties

- string **Username** [get, set]
  - string **Password** [get, set]
  - string **ConfirmPassword** [get, set]
  - string **Email** [get, set]
  - string **PhoneNumber** [get, set]
  - string **Street** [get, set]
  - string **City** [get, set]
  - string **PostCode** [get, set]
  - string **Country** [get, set]
  - bool? **Success** [get, set]
  - string? **Reason** [get, set]
- 

### Detailed Description

Definition at line 5 of file **RegisterModel.cs**.

---

### Property Documentation

**string BankWebApp.Models.RegisterModel.City** [get], [set]

Definition at line 21 of file **RegisterModel.cs**.

**string BankWebApp.Models.RegisterModel.ConfirmPassword** [get], [set]

Definition at line 11 of file **RegisterModel.cs**.

**string BankWebApp.Models.RegisterModel.Country** [get], [set]

Definition at line 25 of file **RegisterModel.cs**.

**string BankWebApp.Models.RegisterModel.Email** [get], [set]

Definition at line 14 of file **RegisterModel.cs**.

**string BankWebApp.Models.RegisterModel.Password** [get], [set]

Definition at line 10 of file **RegisterModel.cs**.

**string BankWebApp.Models.RegisterModel.PhoneNumber** [get], [set]

Definition at line 16 of file **RegisterModel.cs**.

**string BankWebApp.Models.RegisterModel.PostCode** [get], [set]

Definition at line 23 of file **RegisterModel.cs**.

**string? BankWebApp.Models.RegisterModel.Reason** [get], [set]

Definition at line 29 of file **RegisterModel.cs**.

**string BankWebApp.Models.RegisterModel.Street** [get], [set]

Definition at line 19 of file **RegisterModel.cs**.

**bool? BankWebApp.Models.RegisterModel.Success** [get], [set]

Definition at line 28 of file **RegisterModel.cs**.

**string BankWebApp.Models.RegisterModel.Username** [get], [set]

Definition at line 8 of file **RegisterModel.cs**.

---

**The documentation for this class was generated from the following file:**

- C:/Users/tomas/source/repos/BankWebApp/Models/**RegisterModel.cs**

## BankWebApp.Models.RolesModel Class Reference

### Properties

- `int Id` [get, set]
  - `string RoleName` [get, set]
  - `int UserId` [get, set]
- 

### Detailed Description

Definition at line 3 of file **RolesModel.cs**.

---

### Property Documentation

**int BankWebApp.Models.RolesModel.Id** [get], [set]

Definition at line 5 of file **RolesModel.cs**.

**string BankWebApp.Models.RolesModel.RoleName** [get], [set]

Definition at line 6 of file **RolesModel.cs**.

**int BankWebApp.Models.RolesModel.UserId** [get], [set]

Definition at line 7 of file **RolesModel.cs**.

---

The documentation for this class was generated from the following file:

- `C:/Users/tomas/source/repos/BankWebApp/Models/RolesModel.cs`

## BankWebApp.Models.TransactionModel Class Reference

### Properties

- `int Id` [get, set]
- `int SenderId` [get, set]
- `BankAccountModel Sender` [get, set]
- `int ReceiverId` [get, set]
- `BankAccountModel Receiver` [get, set]
- `decimal Amount` [get, set]
- `DateTime SentAt` [get, set]

---

### Detailed Description

Definition at line 3 of file `TransactionModel.cs`.

---

### Property Documentation

**decimal BankWebApp.Models.TransactionModel.Amount** [get], [set]

Definition at line 10 of file `TransactionModel.cs`.

**int BankWebApp.Models.TransactionModel.Id** [get], [set]

Definition at line 5 of file `TransactionModel.cs`.

**BankAccountModel BankWebApp.Models.TransactionModel.Receiver** [get], [set]

Definition at line 9 of file `TransactionModel.cs`.

**int BankWebApp.Models.TransactionModel.ReceiverId** [get], [set]

Definition at line 8 of file `TransactionModel.cs`.

**BankAccountModel BankWebApp.Models.TransactionModel.Sender** [get], [set]

Definition at line 7 of file `TransactionModel.cs`.

**int BankWebApp.Models.TransactionModel.SenderId** [get], [set]

Definition at line 6 of file `TransactionModel.cs`.

**DateTime BankWebApp.Models.TransactionModel.SentAt** [get], [set]

Definition at line 11 of file `TransactionModel.cs`.

---

**The documentation for this class was generated from the following file:**

- `C:/Users/tomas/source/repos/BankWebApp/Models/TransactionModel.cs`

## BankWebApp.Services.TransferService Class Reference

This class provides services for transferring money between bank accounts.

### Public Member Functions

- **TransferService ()**  
*Initializes a new instance of the TransferService class.*
- **bool string Reason TransferMoney** (string FromAcc, string ToAcc, decimal Amount)
- **void PrintMoney** (string AccountNumber, decimal Amount)  
*Adds funds to a specified account.*

### Public Attributes

- **bool Success**  
*Transfers money from one account to another.*

---

### Detailed Description

This class provides services for transferring money between bank accounts.

Definition at line 5 of file **TransferService.cs**.

---

### Constructor & Destructor Documentation

#### BankWebApp.Services.TransferService.TransferService ()

Initializes a new instance of the TransferService class.

Definition at line 12 of file **TransferService.cs**.

---

### Member Function Documentation

**void BankWebApp.Services.TransferService.PrintMoney** (string *AccountNumber*, decimal *Amount*)

Adds funds to a specified account.

#### Parameters

<i>AccountNumber</i>	The account number to add funds to.
<i>Amount</i>	The amount of money to add.

Definition at line 63 of file **TransferService.cs**.

**bool string Reason BankWebApp.Services.TransferService.TransferMoney** (string *FromAcc*, string *ToAcc*, decimal *Amount*)

Definition at line **24** of file **TransferService.cs**.

---

## Member Data Documentation

### **bool BankWebApp.Services.TransferService.Success**

Transfers money from one account to another.

#### **Parameters**

<i>FromAcc</i>	The account number to transfer money from.
<i>ToAcc</i>	The account number to transfer money to.
<i>Amount</i>	The amount of money to transfer.

#### **Returns**

A tuple containing a boolean indicating success or failure, and a string containing the reason for failure.

Definition at line **24** of file **TransferService.cs**.

---

**The documentation for this class was generated from the following file:**

- C:/Users/tomas/source/repos/BankWebApp/Services/**TransferService.cs**



# BankWebApp.Models.TransferViewModel Class Reference

## Properties

- `ICollection<BankAccountModel> BankAccounts` [get, set]
- `string FromAccountId` [get, set]
- `string ToAccountId` [get, set]
- `decimal Amount` [get, set]
- `bool? Success` [get, set]
- `string? Reason` [get, set]
- `static TransferViewModel Empty` [get]

---

## Detailed Description

Definition at line 3 of file `TransferViewModel.cs`.

---

## Property Documentation

**decimal BankWebApp.Models.TransferViewModel.Amount** [get], [set]

Definition at line 10 of file `TransferViewModel.cs`.

**ICollection<BankAccountModel>  
BankWebApp.Models.TransferViewModel.BankAccounts** [get], [set]

Definition at line 5 of file `TransferViewModel.cs`.

**TransferViewModel BankWebApp.Models.TransferViewModel.Empty** [static], [get]

Definition at line 15 of file `TransferViewModel.cs`.

**string BankWebApp.Models.TransferViewModel.FromAccountId** [get], [set]

Definition at line 7 of file `TransferViewModel.cs`.

**string? BankWebApp.Models.TransferViewModel.Reason** [get], [set]

Definition at line 13 of file `TransferViewModel.cs`.

**bool? BankWebApp.Models.TransferViewModel.Success** [get], [set]

Definition at line 12 of file `TransferViewModel.cs`.

**string BankWebApp.Models.TransferViewModel.ToAccountId** [get], [set]

Definition at line 8 of file `TransferViewModel.cs`.

---

**The documentation for this class was generated from the following file:**

- `C:/Users/tomas/source/repos/BankWebApp/Models/TransferViewModel.cs`

## BankWebApp.Models.UserModel Class Reference

Represents a User in the system.

### Properties

- **int Id** [get, set]  
*Unique identifier for the user.*
- **string Username** [get, set]  
*Username of the user.*
- **string PasswordHash** [get, set]  
*Hashed password of the user. with bcrypt.*
- **DateTime CreatedAt** [get, set]  
*A date when the user was created. (in database)*
- **ContactModel Contact** [get, set]  
*Contact model associated with the user.*
- **AddressModel Address** [get, set]  
*Address model associated with the user.*

---

### Detailed Description

Represents a User in the system.

Definition at line 6 of file **UserModel.cs**.

---

### Property Documentation

**AddressModel BankWebApp.Models.UserModel.Address** [get], [set]

Address model associated with the user.

Definition at line 31 of file **UserModel.cs**.

**ContactModel BankWebApp.Models.UserModel.Contact** [get], [set]

Contact model associated with the user.

Definition at line 27 of file **UserModel.cs**.

**DateTime BankWebApp.Models.UserModel.CreatedAt** [get], [set]

A date when the user was created. (in database)

Definition at line **23** of file **UserModel.cs**.

**int BankWebApp.Models.UserModel.Id** [get], [set]

Unique identifier for the user.

Definition at line **11** of file **UserModel.cs**.

**string BankWebApp.Models.UserModel.PasswordHash** [get], [set]

Hashed password of the user. with bcrypt.

Definition at line **19** of file **UserModel.cs**.

**string BankWebApp.Models.UserModel.Username** [get], [set]

Username of the user.

Definition at line **15** of file **UserModel.cs**.

---

**The documentation for this class was generated from the following file:**

- C:/Users/tomas/source/repos/BankWebApp/Models/**UserModel.cs**

## BankWebApp.Services.UserService Class Reference

Service class for managing users.

### Public Member Functions

- **UserService ()**  
*Constructor for UserService. Initializes a new instance of the DatabaseService and refreshes the cache.*
- **ICollection< UserModel > GetUsers ()**  
*Retrieves the list of users. If the cache is null or expired, it refreshes the cache before returning the users.*
- **UserModel? GetUserById (int id)**  
*Retrieves a user by their ID.*
- **UserModel? GetUserByUsername (string username)**  
*Retrieves a user by their username.*
- **bool string reason RegisterUser (RegisterModel newUser)**
- **ICollection< BankAccountModel > GetBankAccountsById (int uid)**  
*Retrieves the bank accounts of a user by their ID.*
- **BankAccountModel? GetBankAccountsById (string id)**  
*Retrieves a bank account by its account number.*
- **ICollection< BankAccountModel > GetAllBankAccounts ()**  
*Retrieves all bank accounts.*
- **ICollection< RolesModel > GetRolesById (int uid)**  
*Retrieves the roles of a user by their ID.*
- **ICollection< TransactionModel > GetAllTransactions ()**  
*Retrieves all transactions.*
- **ICollection< TransactionModel > GetTransactionsByAccountId (int accountId)**  
*Retrieves the transactions of a bank account by its ID.*

### Public Attributes

- **bool success**  
*Registers a new user.*

---

### Detailed Description

Service class for managing users.

Definition at line 9 of file **UserService.cs**.

---

## Constructor & Destructor Documentation

### **BankWebApp.Services.UserService.UserService ()**

Constructor for UserService. Initializes a new instance of the DatabaseService and refreshes the cache.

Definition at line 26 of file **UserService.cs**.

---

## Member Function Documentation

### **ICollection< BankAccountModel > BankWebApp.Services.UserService.GetAllBankAccounts ()**

Retrieves all bank accounts.

#### **Returns**

A list of all bank accounts.

Definition at line 153 of file **UserService.cs**.

### **ICollection< TransactionModel > BankWebApp.Services.UserService.GetAllTransactions ()**

Retrieves all transactions.

#### **Returns**

A list of all transactions.

Definition at line 172 of file **UserService.cs**.

### **ICollection< BankAccountModel > BankWebApp.Services.UserService.GetBankAccountsById (int *uid*)**

Retrieves the bank accounts of a user by their ID.

#### **Parameters**

<i>uid</i>	The ID of the user.
------------	---------------------

#### **Returns**

A list of bank accounts owned by the user.

Definition at line 134 of file **UserService.cs**.

### **BankAccountModel? BankWebApp.Services.UserService.GetBankAccountsById (string *id*)**

Retrieves a bank account by its account number.

#### Parameters

<i>id</i>	The account number of the bank account.
-----------	---

#### Returns

The bank account with the given account number, or null if no such account exists.

Definition at line **144** of file **UserService.cs**.

#### **ICollection< RolesModel > BankWebApp.Services.UserService.GetRolesById (int *uid*)**

Retrieves the roles of a user by their ID.

#### Parameters

<i>uid</i>	The ID of the user.
------------	---------------------

#### Returns

A list of roles assigned to the user.

Definition at line **163** of file **UserService.cs**.

#### **ICollection< TransactionModel > BankWebApp.Services.UserService.GetTransactionsByAccountId (int *accountId*)**

Retrieves the transactions of a bank account by its ID.

#### Parameters

<i>accountId</i>	The ID of the bank account.
------------------	-----------------------------

#### Returns

A list of transactions associated with the bank account.

Definition at line **182** of file **UserService.cs**.

#### **UserModel? BankWebApp.Services.UserService.GetUserById (int *id*)**

Retrieves a user by their ID.

#### Parameters

<i>id</i>	The ID of the user.
-----------	---------------------

#### Returns

The user with the given ID, or null if no such user exists.

Definition at line **60** of file **UserService.cs**.

#### **UserModel? BankWebApp.Services.UserService.GetUserByUsername (string *username*)**

Retrieves a user by their username.

#### Parameters

<i>username</i>	The username of the user.
-----------------	---------------------------

### Returns

The user with the given username, or null if no such user exists.

Definition at line **70** of file **UserService.cs**.

### **IList< UserModel > BankWebApp.Services.UserService.GetUsers ()**

Retrieves the list of users. If the cache is null or expired, it refreshes the cache before returning the users.

### Returns

A list of UserModel instances.

Definition at line **36** of file **UserService.cs**.

### **bool string reason BankWebApp.Services.UserService.RegisterUser (RegisterModel newUser)**

Definition at line **80** of file **UserService.cs**.

---

## Member Data Documentation

### **bool BankWebApp.Services.UserService.success**

Registers a new user.

### Parameters

<i>newUser</i>	The details of the new user.
----------------	------------------------------

### Returns

A tuple indicating whether the registration was successful and a reason for failure, if applicable.

Definition at line **80** of file **UserService.cs**.

---

**The documentation for this class was generated from the following file:**

- C:/Users/tomas/source/repos/BankWebApp/Services/**UserService.cs**



# File Documentation

## C:/Users/tomas/source/repos/BankWebApp/Components/NavbarViewComponent.cs File Reference

### Classes

**class** BankWebApp.Components.NavbarViewComponent**Namespaces**

- namespace **BankWebApp**
- namespace **BankWebApp.Components**

## NavbarViewComponent.cs

Go to the documentation of this file.00001 using  
00002 Microsoft.AspNetCore.Mvc;  
00003 namespace BankWebApp.Components;  
00004  
00005 public class NavbarViewComponent : ViewComponent  
00006 {  
00007 public IViewComponentResult Invoke()  
00008 {  
00009 return View();  
00010 }  
00011 }

## C:/Users/tomas/source/repos/BankWebApp/Controllers/AccountController.cs File Reference

### Classes

class **BankWebApp.Controllers.AccountController***The AccountController class is responsible for handling requests related to the user's bank account. It includes actions for displaying the account index, transferring funds, adding funds (admin only), listing users (admin only), showing user details (admin only), deleting a user (admin only), and viewing transaction history.*

### Namespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Controllers**  
*HomeController class that inherits from Controller. This class is responsible for handling the requests related to the Home page of the application.*

## AccountController.cs

```
Go to the documentation of this file.00001 using BankWebApp.Models;
00002 using BankWebApp.Services;
00003 using BankWebApp.Tools;
00004 using Microsoft.AspNetCore.Authorization;
00005 using Microsoft.AspNetCore.Mvc;
00006
00007 namespace BankWebApp.Controllers;
00008
00013 [Authorize]
00014 public class AccountController : Controller
00015 {
00016     private readonly ILogger<AccountController> _logger;
00017     private readonly UserService _userService;
00018     private readonly TransferService _transferService;
00019
00026     public AccountController(ILogger<AccountController> logger, UserService
userService,
                                TransferService transferService)
00027     {
00028         {
00029             _logger = logger;
00030             _userService = userService;
00031             _transferService = transferService;
00032         }
00033
00038     public IActionResult Index()
00039     {
00040         // users claims
00041         var claims = User.Claims.ToArray();
00042
00043         // extract information from claims (username not needed)
00044         //string username = claims[0].Value;
00045         int id = claims[1].Value.ToInt32();
00046
00047         var user = _userService.GetUserById(id)!;
00048         var bankAccounts = _userService.GetBankAccountsById(user.Id);
00049
00050         var model = new AccountIndexModel()
00051         {
00052             SignedInUser = user,
00053             BankAccounts = bankAccounts
00054         };
00055
00056         return View(model);
00057     }
00058
00065     public IActionResult Transfer(bool? success = null, string? reason = null)
00066     {
00067         var model = TransferViewModel.Empty;
00068         var userId = User.Claims.ToArray()[1].Value.ToInt32();
00069
00070         model.Success = success;
00071         model.Reason = reason;
00072
00073         var bankAccounts = _userService.GetBankAccountsById(userId);
00074         model.BankAccounts = bankAccounts;
00075
00076         return View(model);
00077     }
00078
00084     [HttpPost]
00085     public IActionResult Transfer(TransferViewModel model)
00086     {
00087         var success = true;
00088         var reason = "";
00089
00090         // check if from account is valid and the user owns it
00091         var fromAccount = _userService.GetBankAccountsById(model.FromAccountId);
00092
00093         if (fromAccount == null)
00094         {
00095             success = false;
00096             reason = "Invalid account to send from.";
```

```

00097         return RedirectToAction("Transfer", new { Success = success, Reason =
reason });
00098     }
00099
00100     var userId = User.Claims.ToArray()[1].Value.ToInt32();
00101     var user = _userService.GetUserById(userId!);
00102     var userOwnsAccount = fromAccount.UserId == user.Id;
00103
00104     if (!userOwnsAccount)
00105     {
00106         success = false;
00107         reason = "You do not own this account.";
00108
00109         return RedirectToAction("Transfer", new { Success = success, Reason =
reason });
00110     }
00111
00112     // now check if to account is valid
00113
00114     var toAccount = _userService.GetBankAccountsById(model.ToAccountId);
00115
00116     if (toAccount == null)
00117     {
00118         success = false;
00119         reason = "Invalid account to send to.";
00120         return RedirectToAction("Transfer", new { Success = success, Reason =
reason });
00121     }
00122
00123     // check if amount is valid
00124
00125     if (model.Amount <= 0)
00126     {
00127         success = false;
00128         reason = "Invalid amount (cannot be less than 0).";
00129         return RedirectToAction("Transfer", new { Success = success, Reason =
reason });
00130     }
00131
00132     // check if from account has enough money
00133
00134     if (fromAccount.Balance <= model.Amount)
00135     {
00136         success = false;
00137         reason = "Not enough money in account.";
00138         return RedirectToAction("Transfer", new { Success = success, Reason =
reason });
00139     }
00140
00141     // transfer money
00142
00143     var transferCheck =
_transferService.TransferMoney(fromAccount.AccountNumber, toAccount.AccountNumber,
model.Amount);
00144
00145
00146     return RedirectToAction("Transfer", new { Success = success, Reason = reason
});
00147 }
00148
00155 [Authorize(Roles = "Admin")]
00156 public IActionResult AddFunds(bool? success = null, string? reason = null)
00157 {
00158     var model = new AddFundsViewModel()
00159     {
00160         BankAccounts = _userService.GetAllBankAccounts(),
00161         Success = success,
00162         Reason = reason
00163     };
00164     return View(model);
00165 }
00166
00172 [HttpPost]
00173 [Authorize(Roles = "Admin")]
00174 public IActionResult AddFunds(AddFundsViewModel model)
00175 {
00176     bool success = true;

```

```

00177         string reason = "";
00178
00179         // check if such account exists
00180         var bankAccount =
00181             _userService.GetBankAccountsById(model.SelectedBankAccountNumber);
00182         if (bankAccount == null)
00183         {
00184             success = false;
00185             reason = "Invalid account.";
00186             return RedirectToAction("AddFunds", new { Success = success, Reason =
reason });
00187         }
00188
00189         // check if amount is valid
00190
00191         if (model.Amount <= 0)
00192         {
00193             success = false;
00194             reason = "Invalid amount (cannot be less than 0).";
00195             return RedirectToAction("AddFunds", new { Success = success, Reason =
reason });
00196         }
00197
00198         // add funds
00199         _transferService.PrintMoney(model.SelectedBankAccountNumber,
model.Amount);
00200
00201         return RedirectToAction("AddFunds", new { Success = success, Reason = reason
});
00202     }
00203
00208     [Authorize(Roles = "Admin")]
00209     public IActionResult ListUsers()
00210     {
00211         var model = new List<ListUsersViewModel>();
00212
00213         var users = _userService.GetUsers();
00214
00215         foreach (var user in users)
00216         {
00217             ListUsersViewModel userViewModel = new()
00218             {
00219                 UserModel = user,
00220                 BankAccounts = _userService.GetBankAccountsById(user.Id)
00221             };
00222
00223             model.Add(userViewModel);
00224         }
00225
00226         return View(model);
00227     }
00228
00234     [Authorize(Roles = "Admin")]
00235     public IActionResult Show(string id)
00236     {
00237         if (!int.TryParse(id, out ))
00238         {
00239             return RedirectToAction("ListUsers");
00240         }
00241
00242         var user = _userService.GetUserById(int.Parse(id));
00243         var bankAccounts = userService.GetBankAccountsById(user!.Id);
00244
00245         var transactions = bankAccounts.ToList().SelectMany(bankAccount =>
00246             _userService.GetTransactionsByAccountId(bankAccount.Id)).ToList();
00247
00248         var model = new ListUsersViewModel()
00249         {
00250             UserModel = user,
00251             BankAccounts = bankAccounts,
00252             Transactions = transactions,
00253         };
00254
00255         return View(model);
00256     }
00257 }

```

```

00258
00263     public IActionResult History()
00264     {
00265         AccountHistoryModel model = new();
00266
00267         var myId = (User.Claims.ToArray()[1].Value).ToInt32();
00268
00269         var bankAccounts = _userService.GetBankAccountsById(myId);
00270
00271         // get all transactions from all bank accounts
00272         var transactions = bankAccounts.ToList().SelectMany(bankAccount =>
00273             _userService.GetTransactionsByAccountId(bankAccount.Id)).ToList();
00274
00275         model.Transactions = transactions;
00276
00277         return View(model);
00278     }
00279 }

```

## C:/Users/tomas/source/repos/BankWebApp/Controllers/HealthCheckController.cs File Reference

### Classes

class **BankWebApp.Controllers.HealthCheckController***Controller for handling health checks of the application.*

### Namespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Controllers**  
*HomeController class that inherits from Controller. This class is responsible for handling the requests related to the Home page of the application.*



## HealthCheckController.cs

```
Go to the documentation of this file.00001 using System.Text.Json;
00002 using BankWebApp.Services;
00003 using Microsoft.AspNetCore.Mvc;
00004 using Microsoft.Extensions.Diagnostics.HealthChecks;
00005
00006 namespace BankWebApp.Controllers;
00007
00011 [Route("api/[controller]/[action]")]
00012 public class HealthCheckController : ControllerBase
00013 {
00014     private readonly ILogger<HealthCheckController> _logger;
00015
00016     // Services used to check the health of the database, disk, and memory.
00017     private readonly DatabaseHealthService _dbService;
00018     private readonly DiskHealthService _diskService;
00019     private readonly MemoryHealthService _memoryService;
00020
00021
00029     public HealthCheckController(ILogger<HealthCheckController> logger,
00030         DatabaseHealthService dbService,
00031         DiskHealthService diskService,
00032         MemoryHealthService memoryService)
00033     {
00034         _logger = logger;
00035         _dbService = dbService;
00036         _diskService = diskService;
00037         _memoryService = memoryService;
00038     }
00039
00040
00047     [HttpGet]
00048     public async Task<IActionResult> All()
00049     {
00050         var DatabaseHealth = await CheckDatabaseHealth();
00051         var DiskHealth = await CheckDiskHealth();
00052         var RAMHealth = await CheckRAMHealth();
00053
00054         var healthArray = new HealthCheckResult[]
00055         {
00056             DatabaseHealth.Item2,
00057             DiskHealth.Item2,
00058             RAMHealth.Item2
00059         };
00060
00061         #region OverallHealthCheck
00062         HealthStatus overallHealth = HealthStatus.Healthy;
00063
00064         foreach (var healthValue in healthArray)
00065         {
00066             if (healthValue.Status == HealthStatus.Unhealthy)
00067             {
00068                 overallHealth = HealthStatus.Unhealthy;
00069                 break;
00070             }
00071             else if (healthValue.Status == HealthStatus.Degraded)
00072             {
00073                 overallHealth = HealthStatus.Degraded;
00074             }
00075         }
00076         #endregion
00077
00078         var obj = new
00079         {
00080             OverallStatusInt = overallHealth,
00081             OverallStatus = overallHealth.ToString(),
00082
00083             StatusEnum = new {
00084                 HealthStatus.Healthy,
00085                 HealthStatus.Degraded,
00086                 HealthStatus.Unhealthy
00087             },
00088
00089             DatabaseHealth = DatabaseHealth.Item2,
```

```

00090         DiskHealth = DiskHealth.Item2,
00091         RAMHealth = RAMHealth.Item2
00092     };
00093
00094     var json = JsonSerializer.Serialize(obj);
00095
00096     return overallHealth switch
00097     {
00098         HealthStatus.Healthy => Ok(json),
00099         HealthStatus.Degraded => BadRequest(json),
00100         HealthStatus.Unhealthy => BadRequest(json)
00101     };
00102 }
00103
00104 [HttpGet]
00105 public async Task<IActionResult> Database()
00106 {
00107     return (await CheckDatabaseHealth()).Item1;
00108 }
00109
00110 [HttpGet]
00111 public async Task<IActionResult> Disk()
00112 {
00113     return (await CheckDiskHealth()).Item1;
00114 }
00115
00116 [HttpGet]
00117 public async Task<IActionResult> RAM()
00118 {
00119     return (await CheckRAMHealth()).Item1;
00120 }
00121
00122 private async Task<(IActionResult, HealthCheckResult)> CheckDatabaseHealth()
00123 {
00124     var report = await _dbService.CheckHealthAsync(new (), new ());
00125     string json = JsonSerializer.Serialize(report);
00126
00127     return report.Status switch
00128     {
00129         HealthStatus.Healthy => (Ok(json), report),
00130         HealthStatus.Degraded => (BadRequest(json), report),
00131         HealthStatus.Unhealthy => (BadRequest(json), report)
00132     };
00133 }
00134
00135 private async Task<(IActionResult, HealthCheckResult)> CheckDiskHealth()
00136 {
00137     var report = await _diskService.CheckHealthAsync(new (), new ());
00138     string json = JsonSerializer.Serialize(report);
00139
00140     return report.Status switch
00141     {
00142         HealthStatus.Healthy => (Ok(json), report),
00143         HealthStatus.Degraded => (BadRequest(json), report),
00144         HealthStatus.Unhealthy => (BadRequest(json), report)
00145     };
00146 }
00147
00148 private async Task<(IActionResult, HealthCheckResult)> CheckRAMHealth()
00149 {
00150     var report = await _memoryService.CheckHealthAsync(new (), new ());
00151     string json = JsonSerializer.Serialize(report);
00152
00153     return report.Status switch
00154     {
00155         HealthStatus.Healthy => (Ok(json), report),
00156         HealthStatus.Degraded => (BadRequest(json), report),
00157         HealthStatus.Unhealthy => (BadRequest(json), report)
00158     };
00159 }
00160 }
00161
00162 }
00163
00164 }
00165
00166 }
00167
00168 }
00169
00170 }
00171
00172 }
00173
00174 }
00175
00176 }
00177
00178 }
00179
00180 }
00181
00182 }
00183
00184 }
00185 }

```

## C:/Users/tomas/source/repos/BankWebApp/Controllers/HomeController.cs File Reference

### Classes

#### **class** BankWebApp.Controllers.HomeControllerNamespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Controllers**  
*HomeController class that inherits from Controller. This class is responsible for handling the requests related to the Home page of the application.*

### Variables

- **\$ Loginfailedforuser**
  - Passworddoesntmatch **\$ Loginsuccessfulforuser**
- 

### Variable Documentation

#### **accountnotfound \$ Loginfailedforuser**

Definition at line **106** of file **HomeController.cs**.

#### **Passworddoesntmatch \$ Loginsuccessfulforuser**

Definition at line **106** of file **HomeController.cs**.

## HomeController.cs

```
Go to the documentation of this file.00001 using BankWebApp.Models;
00002 using Microsoft.AspNetCore.Mvc;
00003 using System.Diagnostics;
00004 using BankWebApp.Services;
00005 using Microsoft.AspNetCore.Identity;
00006
00011 namespace BankWebApp.Controllers
00012 {
00013     public class HomeController : Controller
00014     {
00015         private readonly ILogger<HomeController> _logger;
00016         private readonly UserService _userService;
00017         private readonly MySignInManager _signInManager;
00018
00026         public HomeController(ILogger<HomeController> logger, UserService
userService, MySignInManager signInManager)
00027         {
00028             _logger = logger;
00029             _userService = userService;
00030             _signInManager = signInManager;
00031         }
00032
00037         public IActionResult Index()
00038         {
00039             return View();
00040         }
00041
00046         public IActionResult Privacy()
00047         {
00048             return View();
00049         }
00050
00055         public IActionResult Login()
00056         {
00057             return View();
00058         }
00059
00065         [HttpPost]
00066         public IActionResult Login(LoginModel loginModel)
00067         {
00068             if (ModelState.IsValid)
00069             {
00070                 var Username = loginModel.Username;
00071                 var Password = loginModel.Password;
00072
00073                 if (string.IsNullOrEmpty(Username) &&
!string.IsNullOrEmpty(Password))
00074                 {
00075                     _logger.LogDebug($"Login failed for user {Username} (empty
credentials)");
00076                     return RedirectToAction();
00077                 }
00078
00079                 var user = _userService.GetUserByUsername(Username);
00080                 if (user == null)
00081                 {
00082                     _logger.LogDebug($"Login failed for user {Username} (account
not found)");
00083                     return RedirectToAction();
00084                 }
00085
00086                 if (!Tools.PasswordHashes.VerifyPassword(Password,
user.PasswordHash))
00087                 {
00088                     _logger.LogDebug($"Login failed for user {Username} (Password
doesn't match)");
00089                     return RedirectToAction();
00090                 }
00091
00092                 _signInManager.SignInAsync(user,
loginModel.RememberMe).GetAwaiter().GetResult();
00093
00094                 _logger.LogDebug($"Login successful for user {Username}");
```

```

00095
00096         return RedirectToAction("Index", "Account");
00097     }
00098 }
00099     return View();
00100 }
00101
00106     public IActionResult Logout()
00107     {
00108         var signedIn = User.Identity!.IsAuthenticated;
00109         if (!signedIn)
00110         {
00111             return RedirectToAction("Login");
00112         }
00113
00114         _signInManager.SignOutAsync().GetAwaiter().GetResult(); //wait for
sign out before redirecting
00115         return RedirectToAction("Login");
00116     }
00117
00122     [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore
= true)]
00123     public IActionResult Error()
00124     {
00125         return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??
HttpContext.TraceIdentifier });
00126     }
00127
00132     public IActionResult AccessDenied()
00133     {
00134         return View("Error", new ErrorViewModel { RequestId =
Activity.Current?.Id ?? HttpContext.TraceIdentifier });
00135     }
00136
00141     public IActionResult Register()
00142     {
00143         return View();
00144     }
00145
00151     [HttpPost]
00152     public IActionResult Register(RegisterModel registerModel)
00153     {
00154         if (ModelState.IsValid)
00155         {
00156             var resp = _userService.RegisterUser(registerModel);
00157
00158             if (resp.success)
00159             {
00160                 _logger.LogDebug($"Registered user
{registerModel.Username}");
00161
00162                 registerModel.Success = true;
00163
00164                 return RedirectToAction("Login");
00165             }
00166             else
00167             {
00168                 _logger.LogDebug($"Failed to register user
{registerModel.Username}: {resp.reason}");
00169                 ModelState.AddModelError("Username", resp.reason);
00170
00171                 registerModel.Success = false;
00172                 registerModel.Reason = resp.reason;
00173
00174                 return View(registerModel);
00175             }
00176         }
00177
00178         registerModel.Success = false;
00179         registerModel.Reason = string.Join("; ",
ModelState.Values.SelectMany(v => v.Errors).Select(e => e.ErrorMessage));
00180
00181         return View(registerModel);
00182     }
00183 }
00184 }

```



## C:/Users/tomas/source/repos/BankWebApp/env/Envs.cs File Reference

### Classes

- class **BankWebApp.env.Envs**  
*The Envs static class contains environment variables for the application.*

### Namespaces

- namespace **BankWebApp**
- namespace **BankWebApp.env**

## Envs.cs

```
Go to the documentation of this file.00001 namespace BankWebApp.env;
00002
00006 public static class Envs
00007 {
00014     public static string ConnectionString;
00015 }
```



## **C:/Users/tomas/source/repos/BankWebApp/Models/AccountHistoryModel.cs File Reference**

### **Classes**

**class** BankWebApp.Models.AccountHistoryModel**Namespaces**

- namespace **BankWebApp**
- namespace **BankWebApp.Models**

## AccountHistoryModel.cs

Go to the documentation of this file.00001 namespace BankWebApp.Models;  
00002  
00003 public class AccountHistoryModel  
00004 {  
00005 public IList<TransactionModel> Transactions { get; set; }  
00006 }

## **C:/Users/tomas/source/repos/BankWebApp/Models/AccountIndexModel.cs File Reference**

### **Classes**

**class** BankWebApp.Models.AccountIndexModelNamespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Models**

## AccountIndexModel.cs

Go to the documentation of this file.

```
00001 namespace BankWebApp.Models;
00002
00003 public class AccountIndexModel
00004 {
00005     public UserModel SignedInUser { get; set; }
00006     public IList<BankAccountModel> BankAccounts { get; set; }
00007 }
```

## **C:/Users/tomas/source/repos/BankWebApp/Models/AddFunds ViewModel.cs File Reference**

### **Classes**

**class** BankWebApp.Models.AddFundsViewModel**Namespaces**

- namespace **BankWebApp**
- namespace **BankWebApp.Models**

## AddFundsViewModel.cs

Go to the documentation of this file.

```
00001 namespace BankWebApp.Models;
00002
00003 public class AddFundsViewModel
00004 {
00005     public IList<BankAccountModel> BankAccounts { get; set; }
00006
00007     public decimal Amount { get; set; }
00008     public string SelectedBankAccountNumber { get; set; }
00009
00010     public bool? Success { get; set; }
00011     public string? Reason { get; set; }
00012 }
```

## **C:/Users/tomas/source/repos/BankWebApp/Models/AddressModel.cs File Reference**

### **Classes**

**class** BankWebApp.Models.AddressModelNamespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Models**

## AddressModel.cs

```
Go to the documentation of this file.00001 namespace BankWebApp.Models;
00002
00003 public class AddressModel
00004 {
00005     public int Id { get; set; }
00006     public string Street { get; set; }
00007     public string City { get; set; }
00008     public string PostCode { get; set; }
00009     public string Country { get; set; }
00010 }
```



## **C:/Users/tomas/source/repos/BankWebApp/Models/BankAccountModel.cs File Reference**

### **Classes**

**class** BankWebApp.Models.BankAccountModelNamespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Models**

## BankAccountModel.cs

Go to the documentation of this file.

```
00001 namespace BankWebApp.Models;
00002
00003 public class BankAccountModel
00004 {
00005     public int Id { get; set; }
00006     public string AccountNumber { get; set; }
00007     public decimal Balance { get; set; }
00008     public int UserId { get; set; }
00009     public UserModel User { get; set; }
00010 }
```

## **C:/Users/tomas/source/repos/BankWebApp/Models/ContactModel.cs File Reference**

### **Classes**

**class** BankWebApp.Models.ContactModelNamespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Models**

## ContactModel.cs

```
Go to the documentation of this file.00001 namespace BankWebApp.Models;
00002
00003 public class ContactModel
00004 {
00005     public int Id { get; set; }
00006     public string Email { get; set; }
00007     public string PhoneNumber { get; set; }
00008 }
```

## **C:/Users/tomas/source/repos/BankWebApp/Models/ErrorView Model.cs File Reference**

### **Classes**

**class** BankWebApp.Models.ErrorViewModelNamespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Models**

## ErrorViewModel.cs

Go to the documentation of this file.00001 namespace BankWebApp.Models

```
00002 {
00003     public class ErrorViewModel
00004     {
00005         public string? RequestId { get; set; }
00006
00007         public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
00008     }
00009 }
```

## **C:/Users/tomas/source/repos/BankWebApp/Models/ListUsers ViewModel.cs File Reference**

### **Classes**

**class** BankWebApp.Models.ListUsersViewModel**Namespaces**

- namespace **BankWebApp**
- namespace **BankWebApp.Models**

## ListUsersViewModel.cs

Go to the documentation of this file.

```
00001 namespace BankWebApp.Models;
00002
00003 public class ListUsersViewModel
00004 {
00005     public UserModel UserModel { get; set; }
00006     public IList<BankAccountModel> BankAccounts { get; set; }
00007
00008     public IList<TransactionModel> Transactions { get; set; }
00009 }
```



## **C:/Users/tomas/source/repos/BankWebApp/Models/LoginModel.cs File Reference**

### **Classes**

**class** BankWebApp.Models.LoginModelNamespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Models**

## LoginModel.cs

Go to the documentation of this file.00001 namespace BankWebApp.Models

```
00002 {  
00003     public class LoginModel  
00004     {  
00005         public string Username { get; set; }  
00006         public string Password { get; set; }  
00007         public bool RememberMe { get; set; }  
00008     }  
00009 }
```

## **C:/Users/tomas/source/repos/BankWebApp/Models/RegisterModel.cs File Reference**

### **Classes**

**class** BankWebApp.Models.RegisterModelNamespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Models**

## RegisterModel.cs

Go to the documentation of this file.00001 using  
System.ComponentModel.DataAnnotations;  
00002  
00003 namespace BankWebApp.Models;  
00004  
00005 public class RegisterModel  
00006 {  
00007 [MaxLength(50)]  
00008 public string Username { get; set; }  
00009  
00010 public string Password { get; set; }  
00011 public string ConfirmPassword { get; set; }  
00012  
00013 [EmailAddress]  
00014 public string Email { get; set; }  
00015 [Phone]  
00016 public string PhoneNumber { get; set; }  
00017  
00018 [MaxLength(50)]  
00019 public string Street { get; set; }  
00020 [MaxLength(45)]  
00021 public string City { get; set; }  
00022 [MaxLength(45)]  
00023 public string PostCode { get; set; }  
00024 [MaxLength(45)]  
00025 public string Country { get; set; }  
00026  
00027  
00028 public bool? Success { get; set; }  
00029 public string? Reason { get; set; }  
00030 }

## **C:/Users/tomas/source/repos/BankWebApp/Models/RolesModel.cs File Reference**

### **Classes**

**class** BankWebApp.Models.RolesModelNamespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Models**

## RolesModel.cs

Go to the documentation of this file.

```
00001 namespace BankWebApp.Models;
00002
00003 public class RolesModel
00004 {
00005     public int Id { get; set; }
00006     public string RoleName { get; set; }
00007     public int UserId { get; set; }
00008 }
```

## **C:/Users/tomas/source/repos/BankWebApp/Models/TransactionModel.cs File Reference**

### **Classes**

**class** BankWebApp.Models.TransactionModelNamespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Models**

## TransactionModel.cs

```
Go to the documentation of this file.00001 namespace BankWebApp.Models;
00002
00003 public class TransactionModel
00004 {
00005     public int Id { get; set; }
00006     public int SenderId { get; set; }
00007     public BankAccountModel Sender { get; set; }
00008     public int ReceiverId { get; set; }
00009     public BankAccountModel Receiver { get; set; }
00010     public decimal Amount { get; set; }
00011     public DateTime SentAt { get; set; }
00012 }
```



## **C:/Users/tomas/source/repos/BankWebApp/Models/TransferViewModel.cs File Reference**

### **Classes**

**class** BankWebApp.Models.TransferViewModelNamespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Models**

## TransferViewModel.cs

Go to the documentation of this file.

```
00001 namespace BankWebApp.Models;
00002
00003 public class TransferViewModel
00004 {
00005     public IList<BankAccountModel> BankAccounts { get; set; }
00006
00007     public string FromAccountId { get; set; }
00008     public string ToAccountId { get; set; }
00009
00010     public decimal Amount { get; set; }
00011
00012     public bool? Success { get; set; }
00013     public string? Reason { get; set; }
00014
00015     public static TransferViewModel Empty => new TransferViewModel();
00016 }
```

## C:/Users/tomas/source/repos/BankWebApp/Models/UserModel.cs File Reference

### Classes

class **BankWebApp.Models.UserModel***Represents a User in the system.*

### Namespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Models**

## UserModel.cs

Go to the documentation of this file.

```
00001 namespace BankWebApp.Models;
00002
00006 public class UserModel
00007 {
00011     public int Id { get; set; }
00015     public string Username { get; set; }
00019     public string PasswordHash { get; set; }
00023     public DateTime CreatedAt { get; set; }
00027     public ContactModel Contact { get; set; }
00031     public AddressModel Address { get; set; }
00032 }
```

**C:/Users/tomas/source/repos/BankWebApp/obj/Debug/net7.0/.NETCoreApp,Version=v7.0.AssemblyAttributes.cs File Reference**

## **.NETCoreApp,Version=v7.0.AssemblyAttributes.cs**

```
Go to the documentation of this file.00001 // <autogenerated />
00002 using System;
00003 using System.Reflection;
00004 [assembly:
global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v7.0",
FrameworkDisplayName = ".NET 7.0")]
```

**C:/Users/tomas/source/repos/BankWebApp/obj/Debug/net7.0/  
BankWebApp.AssemblyInfo.cs File Reference**

## BankWebApp.AssemblyInfo.cs

```
Go to the documentation of this file.00001
//-----
00002 // <auto-generated>
00003 //     This code was generated by a tool.
00004 //
00005 //     Changes to this file may cause incorrect behavior and will be lost if
00006 //     the code is regenerated.
00007 // </auto-generated>
00008 //-----
00009
00010 using System;
00011 using System.Reflection;
00012
00013 [assembly:
Microsoft.Extensions.Configuration.UserSecrets.UserSecretsIdAttribute("a8aecb95-9885-4
362-be94-922d42874f8a")]
00014 [assembly: System.Reflection.AssemblyCompanyAttribute("BankWebApp")]
00015 [assembly: System.Reflection.AssemblyConfigurationAttribute("Debug")]
00016 [assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
00017 [assembly: System.Reflection.AssemblyInformationalVersionAttribute("1.0.0")]
00018 [assembly: System.Reflection.AssemblyProductAttribute("BankWebApp")]
00019 [assembly: System.Reflection.AssemblyTitleAttribute("BankWebApp")]
00020 [assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]
00021
00022 // Generated by the MSBuild WriteCodeFragment class.
00023
```



**C:/Users/tomas/source/repos/BankWebApp/obj/Debug/net7.0/  
BankWebApp.GlobalUsings.g.cs File Reference**

## BankWebApp.GlobalUsings.g.cs

```
Go to the documentation of this file.00001 // <auto-generated/>
00002 global using global::Microsoft.AspNetCore.Builder;
00003 global using global::Microsoft.AspNetCore.Hosting;
00004 global using global::Microsoft.AspNetCore.Http;
00005 global using global::Microsoft.AspNetCore.Routing;
00006 global using global::Microsoft.Extensions.Configuration;
00007 global using global::Microsoft.Extensions.DependencyInjection;
00008 global using global::Microsoft.Extensions.Hosting;
00009 global using global::Microsoft.Extensions.Logging;
00010 global using global::System;
00011 global using global::System.Collections.Generic;
00012 global using global::System.IO;
00013 global using global::System.Linq;
00014 global using global::System.Net.Http;
00015 global using global::System.Net.Http.Json;
00016 global using global::System.Threading;
00017 global using global::System.Threading.Tasks;
```

**C:/Users/tomas/source/repos/BankWebApp/obj/Debug/net7.0/  
BankWebApp.RazorAssemblyInfo.cs File Reference**

## BankWebApp.RazorAssemblyInfo.cs

```
Go to the documentation of this file.00001
//-----
00002 // <auto-generated>
00003 //     This code was generated by a tool.
00004 //     Runtime Version:4.0.30319.42000
00005 //
00006 //     Changes to this file may cause incorrect behavior and will be lost if
00007 //     the code is regenerated.
00008 // </auto-generated>
00009 //-----
00010
00011 using System;
00012 using System.Reflection;
00013
00014 [assembly:
Microsoft.AspNetCore.Mvc.ApplicationParts.ProvideApplicationPartFactoryAttribute("Micr
osoft.AspNetCore.Mvc.ApplicationParts.ConsolidatedAssemblyApplicationPartFact" +
00015     "ory, Microsoft.AspNetCore.Mvc.Razor")]
00016
00017 // Generated by the MSBuild WriteCodeFragment class.
00018
```

## C:/Users/tomas/source/repos/BankWebApp/Program.cs File Reference

### Classes

class **BankWebApp.Program***The Program class is the entry point of the application.*

### Namespaces

- namespace **BankWebApp**

## Program.cs

```
Go to the documentation of this file.00001 using BankWebApp.env;
00002 using BankWebApp.Services;
00003
00004 namespace BankWebApp
00005 {
00006     public class Program
00007     {
00008         public static void Main(string[] args)
00009         {
00010             // Initialize a new instance of the WebApplication builder with the
provided command-line arguments.
00011             var builder = WebApplication.CreateBuilder(args);
00012
00013             // Set the connection string from appsettings.json
00014             var connectionString =
builder.Configuration.GetConnectionString("DefaultConnectionString");
00015             Envs.ConnectionString = connectionString!;
00016
00017             // Register MVC controllers and views to the services.
00018             builder.Services.AddControllersWithViews();
00019
00020             // Register application-specific services to the services container.
00021             builder.Services.AddSingleton<UserService>();
00022             builder.Services.AddSingleton<IHttpContextAccessor,
HttpContextAccessor>();
00023             builder.Services.AddSingleton<MySignInManager>();
00024             builder.Services.AddSingleton<TransferService>();
00025             builder.Services.AddSingleton<DatabaseHealthService>();
00026             builder.Services.AddSingleton<DiskHealthService>();
00027             builder.Services.AddSingleton<MemoryHealthService>();
00028
00029             // Register health checks to the services container.
00030             builder.Services.AddHealthChecks()
                .AddCheck<DatabaseHealthService>(nameof(DatabaseHealthService))
                .AddCheck<DiskHealthService>(nameof(DiskHealthService))
                .AddCheck<MemoryHealthService>(nameof(MemoryHealthService));
00031
00032             // Configure authentication services with a custom scheme and cookie
settings.
00033             builder.Services.AddAuthentication(options =>
            {
00034                 options.DefaultScheme = "custom";
00035                 options.DefaultSignInScheme = "custom";
00036                 options.DefaultSignOutScheme = "custom";
00037                 options.DefaultChallengeScheme = "custom";
00038             })
                .AddCookie("custom", options =>
            {
00039                 options.LoginPath = "/Home/Login";
00040                 options.LogoutPath = "/Home/Logout";
00041                 options.AccessDeniedPath = "/Home/AccessDenied";
00042             });
00043
00044             // Build the web application instance.
00045             var app = builder.Build();
00046
00047             // Configure the HTTP request pipeline.
00048             if (!app.Environment.IsDevelopment())
00049             {
00050                 // In non-development environments, use the exception handler
middleware to handle exceptions globally.
00051                 app.UseExceptionHandler("/Home/Error");
00052                 // Use HSTS middleware to add the Strict-Transport-Security header
to HTTP responses.
00053                 app.UseHsts();
00054             }
00055
00056             // Use HTTPS redirection middleware to redirect HTTP requests to HTTPS.
00057             app.UseHttpsRedirection();
00058             // Use static files middleware to serve static files.
00059             app.UseStaticFiles();
00060
00061             // Use routing middleware to route requests to the correct endpoint.
```

```
00075         app.UseRouting();
00076
00077         // Use authorization middleware to authorize users based on their roles
00078         and claims.
00079         app.UseAuthorization();
00080
00081         // maps health checks to /health
00082         app.MapHealthChecks("/health");
00083
00084         // Map the default controller route.
00085         app.MapControllerRoute(
00086             name: "default",
00087             pattern: "{controller=Home}/{action=Index}/{id?}");
00088
00089         // Run the application.
00090         app.Run();
00091     }
00092 }
```

## **C:/Users/tomas/source/repos/BankWebApp/README.md File Reference**



## C:/Users/tomas/source/repos/BankWebApp/Services/DatabaseHealthService.cs File Reference

### Classes

class **BankWebApp.Services.DatabaseHealthServiceService** *for checking the health of the database. Implements the Microsoft.Extensions.Diagnostics.HealthChecks.IHealthCheck interface.*

### Namespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Services**

*The DatabaseService class is responsible for managing the database connection. It implements the IDisposable interface to properly close the connection when it's no longer needed.*

## DatabaseHealthService.cs

```
Go to the documentation of this file.00001 using
Microsoft.Extensions.Diagnostics.HealthChecks;
00002
00003 namespace BankWebApp.Services;
00004
00009 public class DatabaseHealthService: IHealthCheck
00010 {
00017     public Task<HealthCheckResult> CheckHealthAsync(HealthCheckContext context,
CancellationToken cancellationToken = new CancellationToken())
00018     {
00019
00020         try
00021         {
00022             bool alive;
00023             using (var db = new DatabaseService())
00024             {
00025                 alive = db.Ping();
00026             }
00027
00028             if (!alive)
00029             {
00030                 return Task.FromResult(new
HealthCheckResult(HealthStatus.Unhealthy, "Failed to connect to database.));
00031             }
00032
00033             return Task.FromResult(new HealthCheckResult(HealthStatus.Healthy,
"Database connection is healthy.));
00034         }
00035         catch (Exception e)
00036         {
00037             return Task.FromResult(new HealthCheckResult(HealthStatus.Unhealthy,
"Failed to connect to database. (Exception)", e));
00038         }
00039     }
00040 }
```

## C:/Users/tomas/source/repos/BankWebApp/Services/DatabaseService.cs File Reference

### Classes

class **BankWebApp.Services.DatabaseService***The DatabaseService class is responsible for managing the database operations. It contains methods for getting users, checking if a username exists, registering a user, getting bank accounts by user id, transferring funds, getting roles by user id, getting all bank accounts, adding funds, and getting transactions.*

### Namespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Services**

*The DatabaseService class is responsible for managing the database connection. It implements the IDisposable interface to properly close the connection when it's no longer needed.*

## DatabaseService.cs

Go to the documentation of this file.

```
00001 using BankWebApp.env;
00002 using Microsoft.Data.SqlClient;
00003
00008 namespace BankWebApp.Services
00009 {
00010     public partial class DatabaseService : IDisposable
00011     {
00015         private readonly string _connectionString;
00016
00020         private SqlConnection _connection;
00021
00026         public DatabaseService()
00027         {
00028             _connectionString = Envs.ConnectionString;
00029             _connection = new SqlConnection(_connectionString);
00030
00031             Open();
00032         }
00033
00037         private void Open()
00038         {
00039             _connection.Open();
00040         }
00041
00045         private void Close()
00046         {
00047             _connection.Close();
00048         }
00049
00054         public void Dispose()
00055         {
00056             Close();
00057         }
00058     }
00059 }
```

## C:/Users/tomas/source/repos/BankWebApp/Services/DatabaseServiceFunctions.cs File Reference

### Classes

class **BankWebApp.Services.DatabaseService***The DatabaseService class is responsible for managing the database operations. It contains methods for getting users, checking if a username exists, registering a user, getting bank accounts by user id, transferring funds, getting roles by user id, getting all bank accounts, adding funds, and getting transactions.*

### Namespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Services**

*The DatabaseService class is responsible for managing the database connection. It implements the IDisposable interface to properly close the connection when it's no longer needed.*

## DatabaseServiceFunctions.cs

```
Go to the documentation of this file.00001 using System.Data;
00002 using BankWebApp.env;
00003 using BankWebApp.Models;
00004 using Microsoft.Data.SqlClient;
00005
00006 namespace BankWebApp.Services;
00007
00012 public partial class DatabaseService
00013 {
00014
00019     public IList<UserModel> GetUsers()
00020     {
00021         var users = new List<UserModel>();
00022
00023         var sql = "SELECT [Users].Id, [Users].Username, [Users].PasswordHash,
00024 [Users].CreatedAt, [Users].ContactId, [Users].AddressId, " +
00025             "[Contacts].Email, [Contacts].PhoneNumber, " +
00026             "[Addresses].Street, [Addresses].City, [Addresses].PostCode,
00027 [Addresses].Country " +
00028             "FROM [Users] " +
00029             "INNER JOIN [Contacts] ON [Users].[ContactId] = [Contacts].[Id]
00030 " +
00031             "INNER JOIN [Addresses] ON [Users].[AddressId] =
00032 [Addresses].[Id]";
00033         var cmd = new SqlCommand(sql, _connection);
00034         var reader = cmd.ExecuteReader();
00035         while (reader.Read())
00036         {
00037             users.Add(new UserModel
00038             {
00039                 Id = reader.GetInt32(0),
00040                 Username = reader.GetString(1),
00041                 PasswordHash = reader.GetString(2),
00042                 CreatedAt = reader.GetDateTime(3),
00043                 Contact = new ContactModel
00044                 {
00045                     Email = reader.GetString(6),
00046                     PhoneNumber = reader.GetString(7)
00047                 },
00048                 Address = new AddressModel
00049                 {
00050                     Street = reader.GetString(8),
00051                     City = reader.GetString(9),
00052                     PostCode = reader.GetString(10),
00053                     Country = reader.GetString(11)
00054                 }
00055             });
00056         }
00057         return users;
00058     }
00059
00061     public bool UsernameExists(string _username)
00062     {
00063         var sql = "SELECT Username FROM Users WHERE Username = @username";
00064         var cmd = new SqlCommand(sql, _connection);
00065         cmd.Parameters.AddWithValue("@username", _username);
00066         var result = cmd.ExecuteScalar();
00067
00068         return result != null;
00069     }
00070
00076     public bool RegisterUser(UserModel user)
00077     {
00078         using (SqlConnection con = new SqlConnection(Envs.ConnectionString))
00079         {
00080             con.Open();
00081             var transaction = con.BeginTransaction();
00082
00083             try
00084             {
00085
00086                 // Insert contact
```

```

00087         var sql = "INSERT INTO Contacts (Email, PhoneNumber) OUTPUT
INSERTED.Id VALUES (@Email, @PhoneNumber)";
00088         var cmd = new SqlCommand(sql, con);
00089         cmd.Parameters.AddWithValue("@Email", user.Contact.Email);
00090         cmd.Parameters.AddWithValue("@PhoneNumber",
user.Contact.PhoneNumber);
00091         cmd.Transaction = transaction;
00092         var contactId = (int)cmd.ExecuteScalar();
00093
00094         // Insert address
00095         sql =
00096             "INSERT INTO Addresses (Street, City, PostCode, Country) OUTPUT
INSERTED.Id VALUES (@Street, @City, @PostCode, @Country)";
00097         cmd = new SqlCommand(sql, con);
00098         cmd.Parameters.AddWithValue("@Street", user.Address.Street);
00099         cmd.Parameters.AddWithValue("@City", user.Address.City);
00100         cmd.Parameters.AddWithValue("@PostCode", user.Address.PostCode);
00101         cmd.Parameters.AddWithValue("@Country", user.Address.Country);
00102         cmd.Transaction = transaction;
00103         var addressId = (int)cmd.ExecuteScalar();
00104
00105         // Insert user
00106         sql =
00107             "INSERT INTO Users (Username, PasswordHash, CreatedAt,
ContactId, AddressId) OUTPUT INSERTED.Id VALUES (@Username, @PasswordHash, @CreatedAt,
@ContactId, @AddressId)";
00108         cmd = new SqlCommand(sql, con);
00109         cmd.Parameters.AddWithValue("@Username", user.Username);
00110         cmd.Parameters.AddWithValue("@PasswordHash", user.PasswordHash);
00111         cmd.Parameters.AddWithValue("@CreatedAt", DateTime.Now);
00112         cmd.Parameters.AddWithValue("@ContactId", contactId);
00113         cmd.Parameters.AddWithValue("@AddressId", addressId);
00114         cmd.Transaction = transaction;
00115         var userId = cmd.ExecuteScalar();
00116
00117
00118         sql = "INSERT INTO BankAccount (UserId, AccountNumber, Balance)
VALUES (@UserId, @AccountNumber, 0)";
00119         cmd = new SqlCommand(sql, con);
00120         cmd.Parameters.AddWithValue("@UserId", userId);
00121         cmd.Parameters.AddWithValue("@AccountNumber", Guid.NewGuid());
00122         cmd.Transaction = transaction;
00123         cmd.ExecuteNonQuery();
00124
00125         transaction.Commit();
00126
00127         return true;
00128     }
00129     catch (Exception e)
00130     {
00131         transaction.Rollback();
00132         return false;
00133     }
00134 }
00135 }
00136
00142     public IList<BankAccountModel>? GetBankAccountById(int UserId)
00143     {
00144         List<BankAccountModel> accounts = new List<BankAccountModel>();
00145
00146         var sql = "SELECT Id, AccountNumber, Balance, UserId FROM BankAccount WHERE
UserId = @Id";
00147         var cmd = new SqlCommand(sql, connection);
00148         cmd.Parameters.AddWithValue("@Id", UserId);
00149         var reader = cmd.ExecuteReader();
00150
00151         if (reader.Read())
00152         {
00153             accounts.Add(new BankAccountModel
00154             {
00155                 Id = reader.GetInt32(0),
00156                 AccountNumber = reader.GetString(1),
00157                 Balance = reader.GetDecimal(2),
00158                 UserId = reader.GetInt32(3)
00159             });
00160         }
00161     }

```

```

00162         var users = GetUsers();
00163
00164         foreach (var account in accounts)
00165         {
00166             account.User = users.First(u => u.Id == account.UserId);
00167         }
00168
00169         reader.Close();
00170
00171         return accounts.Count > 0 ? accounts : null;
00172     }
00173
00174     public BankAccountModel? GetBankAccountById(string Id)
00175     {
00176         var sql = "SELECT Id, AccountNumber, Balance, UserId FROM BankAccount WHERE
AccountNumber = @Id";
00177         var cmd = new SqlCommand(sql, _connection);
00178         cmd.Parameters.AddWithValue("@Id", Id);
00179         var reader = cmd.ExecuteReader();
00180
00181         if (reader.Read())
00182         {
00183             var result = new BankAccountModel
00184             {
00185                 Id = reader.GetInt32(0),
00186                 AccountNumber = reader.GetString(1),
00187                 Balance = reader.GetDecimal(2),
00188                 UserId = reader.GetInt32(3)
00189             };
00190
00191             var users = GetUsers();
00192
00193             result.User = users.First(u => u.Id == result.UserId);
00194
00195             reader.Close();
00196
00197             return result;
00198         }
00199
00200         return null;
00201     }
00202
00203     public BankAccountModel? GetBankAccountByAccountId(int Id)
00204     {
00205         var sql = "SELECT Id, AccountNumber, Balance, UserId FROM BankAccount WHERE
Id = @Id";
00206         var cmd = new SqlCommand(sql, _connection);
00207         cmd.Parameters.AddWithValue("@Id", Id);
00208         var reader = cmd.ExecuteReader();
00209
00210         if (reader.Read())
00211         {
00212             var result = new BankAccountModel
00213             {
00214                 Id = reader.GetInt32(0),
00215                 AccountNumber = reader.GetString(1),
00216                 Balance = reader.GetDecimal(2),
00217                 UserId = reader.GetInt32(3)
00218             };
00219
00220             var users = GetUsers();
00221
00222             result.User = users.First(u => u.Id == result.UserId);
00223
00224             reader.Close();
00225
00226             return result;
00227         }
00228
00229         return null;
00230     }
00231
00232     public bool TransferFunds(Guid from, Guid To, decimal Amount)
00233     {
00234         var fromAccount = GetBankAccountById(from.ToString());

```



```

00254         var toAccount = GetBankAccountById(To.ToString());
00255
00256         using (SqlConnection con = new SqlConnection(Envs.ConnectionString))
00257         {
00258             con.Open();
00259             SqlTransaction transaction = con.BeginTransaction();
00260
00261             try
00262             {
00263                 var sql = "UPDATE BankAccount SET Balance = Balance - @Amount WHERE
AccountNumber = @From";
00264                 var cmd = new SqlCommand(sql, con);
00265                 cmd.Parameters.AddWithValue("@Amount", Amount);
00266                 cmd.Parameters.AddWithValue("@From", from);
00267                 cmd.Transaction = transaction;
00268                 cmd.ExecuteNonQuery();
00269
00270                 sql = "UPDATE BankAccount SET Balance = Balance + @Amount WHERE
AccountNumber = @To";
00271                 cmd = new SqlCommand(sql, con);
00272                 cmd.Parameters.AddWithValue("@Amount", Amount);
00273                 cmd.Parameters.AddWithValue("@To", To);
00274                 cmd.Transaction = transaction;
00275                 cmd.ExecuteNonQuery();
00276
00277                 sql = "INSERT INTO Transactions (SenderId, ReceiverId, Amount,
SentAt) VALUES (@From, @To, @Amount, @SentAt)";
00278                 cmd = new SqlCommand(sql, con);
00279                 cmd.Parameters.AddWithValue("@From", fromAccount!.Id);
00280                 cmd.Parameters.AddWithValue("@To", toAccount!.Id);
00281                 cmd.Parameters.AddWithValue("@Amount", Amount);
00282                 cmd.Parameters.AddWithValue("@SentAt", DateTime.Now);
00283                 cmd.Transaction = transaction;
00284                 cmd.ExecuteNonQuery();
00285
00286                 transaction.Commit();
00287                 return true;
00288             }
00289             catch (Exception e)
00290             {
00291                 transaction.Rollback();
00292                 return false;
00293             }
00294         }
00295     }
00296
00302     public IList<RolesModel> GetRolesById(int uid)
00303     {
00304         var roles = new List<RolesModel>();
00305
00306         var sql = "SELECT [UserRoles].Id, [UserRoles].RoleName, [UserRoles].UserId
FROM [UserRoles] WHERE [UserRoles].UserId = @Id";
00307         var cmd = new SqlCommand(sql, _connection);
00308         cmd.Parameters.AddWithValue("@Id", uid);
00309         var reader = cmd.ExecuteReader();
00310         while (reader.Read())
00311         {
00312             roles.Add(new RolesModel
00313             {
00314                 Id = reader.GetInt32(0),
00315                 RoleName = reader.GetString(1),
00316                 UserId = reader.GetInt32(2)
00317             });
00318         }
00319
00320         return roles;
00321     }
00322
00327     public IList<BankAccountModel> GetAllBankAccounts()
00328     {
00329         var accounts = new List<BankAccountModel>();
00330
00331         var sql = "SELECT Id, AccountNumber, Balance, UserId FROM BankAccount";
00332         var cmd = new SqlCommand(sql, _connection);
00333         var reader = cmd.ExecuteReader();
00334         while (reader.Read())
00335         {

```

```

00336         accounts.Add(new BankAccountModel
00337         {
00338             Id = reader.GetInt32(0),
00339             AccountNumber = reader.GetString(1),
00340             Balance = reader.GetDecimal(2),
00341             UserId = reader.GetInt32(3)
00342         });
00343     }
00344
00345     var users = GetUsers();
00346
00347     foreach (var account in accounts)
00348     {
00349         account.User = users.First(u => u.Id == account.UserId);
00350     }
00351
00352     return accounts;
00353 }
00354
00360 public void AddFunds(Guid guid, decimal amount)
00361 {
00362     var sql = "UPDATE BankAccount SET Balance = Balance + @Amount WHERE
AccountNumber = @AccountNumber";
00363     var cmd = new SqlCommand(sql, _connection);
00364     cmd.Parameters.AddWithValue("@Amount", amount);
00365     cmd.Parameters.AddWithValue("@AccountNumber", guid);
00366     cmd.ExecuteNonQuery();
00367 }
00368
00374 public IList<TransactionModel> GetTransactions()
00375 {
00376     var sql = "SELECT Id, SenderId, ReceiverId, Amount, SentAt FROM
Transactions";
00377     var cmd = new SqlCommand(sql, _connection);
00378
00379     var transactions = new List<TransactionModel>();
00380
00381     var reader = cmd.ExecuteReader();
00382
00383     while (reader.Read())
00384     {
00385         transactions.Add(new TransactionModel
00386         {
00387             Id = reader.GetInt32(reader.GetOrdinal("Id")),
00388             SenderId = reader.GetInt32(reader.GetOrdinal("SenderId")),
00389             ReceiverId = reader.GetInt32(reader.GetOrdinal("ReceiverId")),
00390             Amount = reader.GetDecimal(reader.GetOrdinal("Amount")),
00391             SentAt = reader.GetDateTime(reader.GetOrdinal("SentAt"))
00392         });
00393     }
00394
00395     reader.Close();
00396
00397     foreach (var transaction in transactions)
00398     {
00399         transaction.Sender = GetBankAccountByAccountId(transaction.SenderId);
00400         transaction.Receiver =
GetBankAccountByAccountId(transaction.ReceiverId);
00401     }
00402
00403     return transactions;
00404 }
00405
00411 public IList<TransactionModel> GetTransactions(int uid)
00412 {
00413     var sql = "SELECT Id, SenderId, ReceiverId, Amount, SentAt FROM Transactions
WHERE SenderId = @Id OR ReceiverId = @Id";
00414     var cmd = new SqlCommand(sql, _connection);
00415
00416     cmd.Parameters.AddWithValue("@Id", uid);
00417
00418     var transactions = new List<TransactionModel>();
00419
00420     var reader = cmd.ExecuteReader();
00421
00422     while (reader.Read())

```

```

00423     {
00424         transactions.Add(new TransactionModel
00425         {
00426             Id = reader.GetInt32(reader.GetOrdinal("Id")),
00427             SenderId = reader.GetInt32(reader.GetOrdinal("SenderId")),
00428             ReceiverId = reader.GetInt32(reader.GetOrdinal("ReceiverId")),
00429             Amount = reader.GetDecimal(reader.GetOrdinal("Amount")),
00430             SentAt = reader.GetDateTime(reader.GetOrdinal("SentAt"))
00431         });
00432     }
00433
00434     reader.Close();
00435
00436     foreach (var transaction in transactions)
00437     {
00438         transaction.Sender = GetBankAccountByAccountId(transaction.SenderId);
00439         transaction.Receiver =
00440             GetBankAccountByAccountId(transaction.ReceiverId);
00441     }
00442     return transactions;
00443 }
00444
00450 public bool Ping()
00451 {
00452     try
00453     {
00454         var sql = "SELECT 1";
00455         using (var cmd = new SqlCommand(sql, _connection))
00456         {
00457             var response = cmd.ExecuteScalar();
00458
00459             // check if the response is 1
00460             if (response is int i && i == 1)
00461             {
00462                 return true;
00463             }
00464             else
00465             {
00466                 return false;
00467             }
00468         }
00469     }
00470
00471     catch (Exception e)
00472     {
00473         return false;
00474     }
00475 }
00476 }

```

## C:/Users/tomas/source/repos/BankWebApp/Services/DiskHealthService.cs File Reference

### Classes

class **BankWebApp.Services.DiskHealthServiceClass** *DiskHealthService. Implements the Microsoft.Extensions.Diagnostics.HealthChecks.IHealthCheck interface. Used to check if the disk has enough free space.*

### Namespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Services**

*The DatabaseService class is responsible for managing the database connection. It implements the IDisposable interface to properly close the connection when it's no longer needed.*

## DiskHealthService.cs

```
Go to the documentation of this file.00001 using
Microsoft.Extensions.Diagnostics.HealthChecks;
00002
00003 namespace BankWebApp.Services;
00004
00010 public class DiskHealthService : IHealthCheck
00011 {
00016     private const long MinimumFreeSpace = 1000000000; // 1 GB
00017
00023     private const long CriticalFreeSpace = 500000000; // 500 MB
00024
00025
00033
00034     public Task<HealthCheckResult> CheckHealthAsync(HealthCheckContext context,
CancellationToken cancellationToken)
00035     {
00036         try
00037         {
00038             var driveInfo = new
DriveInfo(Path.GetPathRoot(Directory.GetCurrentDirectory()));
00039             var free = driveInfo.AvailableFreeSpace;
00040
00041             if (free < MinimumFreeSpace)
00042             {
00043                 return Task.FromResult(HealthCheckResult.Unhealthy($"Not enough
free disk space. (Minimum: {MinimumFreeSpace} bytes, Actual: {free} bytes)"));
00044             }
00045             else if (free < CriticalFreeSpace)
00046             {
00047                 return Task.FromResult(HealthCheckResult.Degraded($"Low disk
space. (Minimum: {MinimumFreeSpace} bytes, Actual: {free} bytes)"));
00048             }
00049             else
00050             {
00051                 return Task.FromResult(HealthCheckResult.Healthy($"Disk has
enough free space. (Minimum: {MinimumFreeSpace} bytes, Actual: {free} bytes)"));
00052             }
00053
00054         }
00055         catch (Exception e)
00056         {
00057             return Task.FromResult(HealthCheckResult.Unhealthy("Failed to check
disk health. (Exception)", e));
00058         }
00059     }
00060 }
```

## C:/Users/tomas/source/repos/BankWebApp/Services/MemoryHealthService.cs File Reference

### Classes

class **BankWebApp.Services.MemoryHealthServiceClass** *MemoryHealthService. Implements the Microsoft.Extensions.Diagnostics.HealthChecks.IHealthCheck interface. Used to check if the RAM has enough free space.*

### Namespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Services**

*The DatabaseService class is responsible for managing the database connection. It implements the IDisposable interface to properly close the connection when it's no longer needed.*

## MemoryHealthService.cs

```
Go to the documentation of this file.00001 using
Microsoft.Extensions.Diagnostics.HealthChecks;
00002
00003 namespace BankWebApp.Services;
00004
00010
00011 public class MemoryHealthService : IHealthCheck
00012 {
00016     private const long MinimumFreeMemory = 5_000_000; // 5 MB
00017
00023     private const long CriticalFreeMemory = 1_000_000; // 1 MB
00024
00032     public Task<HealthCheckResult> CheckHealthAsync(HealthCheckContext context,
CancellationToken cancellationToken)
00033     {
00034         try
00035         {
00036             var free = GC.GetTotalMemory(false);
00037
00038             if (free < MinimumFreeMemory)
00039             {
00040                 return Task.FromResult(HealthCheckResult.Unhealthy($"Not enough
free memory. (Minimum: {MinimumFreeMemory} bytes, Actual: {free} bytes)"));
00041             }
00042             else if (free < CriticalFreeMemory)
00043             {
00044                 return Task.FromResult(HealthCheckResult.Degraded($"Low memory.
(Minimum: {MinimumFreeMemory} bytes, Actual: {free} bytes)"));
00045             }
00046             else
00047             {
00048                 return Task.FromResult(HealthCheckResult.Healthy($"Memory has
enough free space. (Minimum: {MinimumFreeMemory} bytes, Actual: {free} bytes)"));
00049             }
00050
00051         }
00052         catch (Exception e)
00053         {
00054             return Task.FromResult(HealthCheckResult.Unhealthy("Failed to check
memory health. (Exception)", e));
00055         }
00056     }
00057 }
```

## C:/Users/tomas/source/repos/BankWebApp/Services/MySignInManager.cs File Reference

### Classes

class **BankWebApp.Services.MySignInManager***This class is responsible for managing user sign in and sign out operations.*

### Namespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Services**

*The DatabaseService class is responsible for managing the database connection. It implements the IDisposable interface to properly close the connection when it's no longer needed.*



## MySignInManager.cs

```
Go to the documentation of this file.00001 using System.Security.Claims;
00002 using BankWebApp.Models;
00003 using Microsoft.AspNetCore.Authentication;
00004
00005 namespace BankWebApp.Services
00006 {
00010     public class MySignInManager
00011     {
00012         private readonly IHttpContextAccessor _httpContextAccessor;
00013         private readonly UserService _userService;
00014
00020         public MySignInManager(IHttpContextAccessor httpContextAccessor,
UserService userService)
00021         {
00022             _httpContextAccessor = httpContextAccessor;
00023             _userService = userService;
00024         }
00025
00035         public async Task SignInAsync(UserModel user, bool isPersistent = false)
00036         {
00037             var roles = _userService.GetRolesById(user.Id);
00038
00039             // Create a claims identity
00040             var claims = new List<Claim>
00041             {
00042                 new Claim(ClaimTypes.Name, user.Username),
00043                 new Claim(ClaimTypes.PrimarySid, user.Id.ToString()),
00044             };
00045
00046             claims.AddRange(roles.Select(
00047                 role => new Claim(ClaimTypes.Role, role.RoleName)
00048             ));
00049
00050             var claimsIdentity = new ClaimsIdentity(claims, "custom");
00051
00052             // Create a claims principal
00053             var claimsPrincipal = new ClaimsPrincipal(claimsIdentity);
00054
00055             // Sign in the user
00056             await _httpContextAccessor.HttpContext.SignInAsync("custom",
claimsPrincipal, new AuthenticationProperties
00057             {
00058                 IsPersistent = isPersistent,
00059                 ExpiresUtc = DateTime.UtcNow.AddMinutes(30) // Set expiration as
needed
00060             });
00061
00062         }
00063
00068         public async Task SignOutAsync()
00069         {
00070             // Sign out the user
00071             await _httpContextAccessor.HttpContext.SignOutAsync("custom");
00072         }
00073     }
00074 }
```

## C:/Users/tomas/source/repos/BankWebApp/Services/TransferService.cs File Reference

### Classes

class **BankWebApp.Services.TransferService***This class provides services for transferring money between bank accounts.*

### Namespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Services**  
*The DatabaseService class is responsible for managing the database connection. It implements the IDisposable interface to properly close the connection when it's no longer needed.*

## TransferService.cs

Go to the documentation of this file.

```
00001 namespace BankWebApp.Services;
00005 public class TransferService
00006 {
00007     private readonly DatabaseService _databaseService;
00008
00012     public TransferService()
00013     {
00014         _databaseService = new DatabaseService();
00015     }
00016
00024     public (bool Success, string Reason) TransferMoney(string FromAcc, string ToAcc,
decimal Amount)
00025     {
00026         var fromAcc = _databaseService.GetBankAccountById(FromAcc);
00027         var toAcc = _databaseService.GetBankAccountById(ToAcc);
00028
00029         if (fromAcc == null)
00030         {
00031             return (false, "Invalid account to send from.");
00032         }
00033         else if (toAcc == null)
00034         {
00035             return (false, "Invalid account to send to.");
00036         }
00037
00038         if (fromAcc.Balance <= Amount)
00039         {
00040             return (false, "Insufficient funds.");
00041         }
00042
00043         Guid fromGuid = Guid.Parse(fromAcc.AccountNumber);
00044         Guid toGuid = Guid.Parse(toAcc.AccountNumber);
00045
00046         var success = _databaseService.TransferFunds(fromGuid, toGuid, Amount);
00047
00048         if (!success)
00049         {
00050             return (false, "An error occurred while transferring funds.");
00051         }
00052         else
00053         {
00054             return (true, "");
00055         }
00056     }
00057
00063     public void PrintMoney(string AccountNumber, decimal Amount)
00064     {
00065         var account = _databaseService.GetBankAccountById(AccountNumber)!;
00066
00067         Guid guid = Guid.Parse(account.AccountNumber);
00068
00069         _databaseService.AddFunds(guid, Amount);
00070     }
00071 }
```

## C:/Users/tomas/source/repos/BankWebApp/Services/UserService.cs File Reference

### Classes

class **BankWebApp.Services.UserServiceService** *class for managing users.*

### Namespaces

- namespace **BankWebApp**
- namespace **BankWebApp.Services**

*The DatabaseService class is responsible for managing the database connection. It implements the IDisposable interface to properly close the connection when it's no longer needed.*

## UserService.cs

```
Go to the documentation of this file.00001 using BankWebApp.Models;
00002 using BankWebApp.Tools;
00003
00004 namespace BankWebApp.Services
00005 {
00006     public class UserService
00007     {
00008         // Database service instance for database operations
00009         private readonly DatabaseService _databaseService;
00010
00011         // The time when the cache was last updated
00012         private DateTime _lastUpdatedAt;
00013
00014         // Cache for storing user data
00015         private IList<UserModel>? _users = null;
00016
00017         // Duration for which the cache is valid
00018         private readonly TimeSpan _cacheDuration = TimeSpan.FromMinutes(1);
00019
00020         public UserService()
00021         {
00022             _databaseService = new DatabaseService();
00023             RefreshCache();
00024         }
00025
00026         public IList<UserModel> GetUsers()
00027         {
00028             if (_users == null || (_lastUpdatedAt + _cacheDuration) < DateTime.Now)
00029             {
00030                 RefreshCache();
00031             }
00032
00033             return _users!;
00034         }
00035
00036         private void RefreshCache()
00037         {
00038             _users = _databaseService.GetUsers();
00039             _lastUpdatedAt = DateTime.Now;
00040         }
00041
00042         public UserModel? GetUserById(int id)
00043         {
00044             return _users?.FirstOrDefault(user => user.Id == id);
00045         }
00046
00047         public UserModel? GetUserByUsername(string username)
00048         {
00049             return _users?.FirstOrDefault(user => user.Username == username);
00050         }
00051
00052         public (bool success, string reason) RegisterUser(RegisterModel newUser)
00053         {
00054             bool existCheck = _databaseService.UsernameExists(newUser.Username);
00055             if (existCheck)
00056             {
00057                 return (false, "Username already exists");
00058             }
00059
00060             bool passwordMatches = (newUser.Password == newUser.ConfirmPassword);
00061             if (!passwordMatches)
00062             {
00063                 return (false, "Passwords do not match");
00064             }
00065
00066             var NewUser = new UserModel()
00067             {
00068                 // Id auto generated
00069                 Username = newUser.Username,
00070                 PasswordHash = newUser.Password.HashPassword(),
00071                 Contact = new ContactModel()
00072                 {
00073                     // Id auto generated
00074                 }
00075             }
00076         }
00077     }
00078 }
```

```

00102         Email = newUser.Email,
00103         PhoneNumber = newUser.PhoneNumber
00104     },
00105     Address = new AddressModel()
00106     {
00107         // Id auto generated
00108         Street = newUser.Street,
00109         City = newUser.City,
00110         PostCode = newUser.PostCode,
00111         Country = newUser.Country
00112     }
00113     // CreatedAt auto generated
00114 };
00115
00116 var registerCheck = _databaseService.RegisterUser(NewUser);
00117
00118 if (registerCheck)
00119 {
00120     RefreshCache();
00121     return (true, "");
00122 }
00123 else
00124 {
00125     return (false, "Something went wrong during the registering
process");
00126 }
00127 }
00128
00134 public IList<BankAccountModel> GetBankAccountsById(int uid)
00135 {
00136     return _databaseService.GetBankAccountById(uid);
00137 }
00138
00144 public BankAccountModel? GetBankAccountsById(string id)
00145 {
00146     return _databaseService.GetBankAccountById(id);
00147 }
00148
00153 public IList<BankAccountModel> GetAllBankAccounts()
00154 {
00155     return _databaseService.GetAllBankAccounts();
00156 }
00157
00163 public IList<RolesModel> GetRolesById(int uid)
00164 {
00165     return _databaseService.GetRolesById(uid);
00166 }
00167
00172 public IList<TransactionModel> GetAllTransactions()
00173 {
00174     return databaseService.GetTransactions();
00175 }
00176
00182 public IList<TransactionModel> GetTransactionsByAccountId(int accountId)
00183 {
00184     return _databaseService.GetTransactions(accountId);
00185 }
00186
00187 }
00188 }

```

## **C:/Users/tomas/source/repos/BankWebApp/Tools/ClaimTools.cs File Reference**

### **Classes**

- class **BankWebApp.Tools.ClaimTools**

### **Namespaces**

- namespace **BankWebApp**
- namespace **BankWebApp.Tools**

## ClaimTools.cs

Go to the documentation of this file.

```
00001 namespace BankWebApp.Tools;
00002
00003 public static class ClaimTools
00004 {
00006     public static int ToInt32(this string str) => int.Parse(str);
00007 }
```



## **C:/Users/tomas/source/repos/BankWebApp/Tools/PasswordHashes.cs File Reference**

### **Classes**

- class **BankWebApp.Tools.PasswordHashes**

### **Namespaces**

- namespace **BankWebApp**
- namespace **BankWebApp.Tools**

## PasswordHashes.cs

Go to the documentation of this file.00001 using  
System.Runtime.Intrinsics.Arm;  
00002 using System.Security.Cryptography;  
00003 using System.Text;  
00004  
00005 namespace BankWebApp.Tools;  
00006  
00007 public static class PasswordHashes  
00008 {  
00009 public static string HashPassword(this string text)  
00010 {  
00011 return BCrypt.Net.BCrypt.HashPassword(text);  
00012 }  
00013  
00014 public static bool VerifyPassword(string unhashedp, string p2)  
00015 {  
00016 return BCrypt.Net.BCrypt.Verify(unhashedp, p2);  
00017 }  
00018 }

**C:/Users/tomas/source/repos/BankWebApp/wwwroot/lib/jquery-validation/LICENSE.md File Reference**

# **Index**

INDEX