Fake bank

AUTHOR
Version
Wed Jan 10 2024

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):

Hierarchical Index

Class Hierarchy

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

Class Index

Class List

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

File Index

File List

Here is a list of all files with brief descriptions:

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

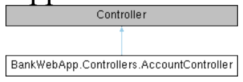
- class **ClaimTools**
- class **PasswordHashes**

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

logger	The logger used to log information about program execution.
userService	The service used to interact with user data.
transferService	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

model	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

success	Indicates whether the last add funds operation was successful.
reason	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**.

IActionResult BankWebApp.Controllers.AccountController.Index ()

Displays the account index page.

Returns

The account index view.

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

IActionResult BankWebApp.Controllers.AccountController.ListUsers ()

Displays the list users page (admin only).

Returns

The list users view.

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

IActionResult BankWebApp.Controllers.AccountController.Show (string *id*)

Displays the details of a user (admin only).

Parameters

id	The ID of the user to show.

Returns

The show view.

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

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

Displays the transfer page.

Parameters

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

Returns

The transfer view.

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

IActionResult BankWebApp.Controllers.AccountController.Transfer (TransferViewModel *model*)

Handles a POST request to transfer funds.

Parameters

model	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]`
- IList< **BankAccountModel** > **BankAccounts** `[get, set]`

Detailed Description

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

Property Documentation

IList<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**.

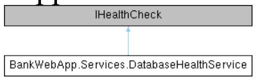
The documentation for this class was generated from the following file:

- 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

context	The context under which the health check is being performed.
cancellationToken	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**.

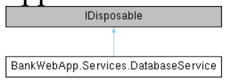
The documentation for this class was generated from the following file:

- 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.

- IList< **TransactionModel** > **GetTransactions** ()
  Gets all transactions from the database.

- IList< **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

guid	The account number to add funds to.
amount	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.GetBankAccountByAccountId (int *Id*)

Gets a bank account by account id.

Parameters

Id	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.GetBankAccountById (int *UserId*)

Gets a list of bank accounts by user id.

Parameters

UserId	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.GetBankAccountById (string *Id*)

Gets a bank account by account number.

Parameters

Id	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.GetRolesById (int *uid*)

Gets a list of roles by user id.

Parameters

uid	The user id to get the roles for.

Returns

A list of RolesModel objects.

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

IList< 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**.

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

Gets a list of transactions by user id.

Parameters

uid	The user id to get the transactions for.

Returns

A list of TransactionModel objects.

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

IList< 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

user	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

from	The account number to transfer funds from.
To	The account number to transfer funds to.
Amount	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**.

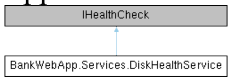
The documentation for this class was generated from the following files:

- 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

context	The context.
cancellationToken	The cancellation token.

Returns

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

Exceptions

Exception	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**.

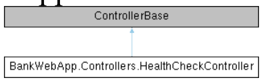
The documentation for this class was generated from the following file:

- 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

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

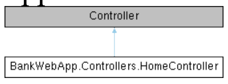
Definition at line **129** of file **HealthCheckController.cs**.

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

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

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

Member Function Documentation

IActionResult 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**.

IActionResult 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**.

IActionResult BankWebApp.Controllers.HomeController.Index ()

Handles the GET request for the Index view.

Returns

The Index view.

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

IActionResult BankWebApp.Controllers.HomeController.Login ()

Handles the GET request for the Login view.

Returns

The Login view.

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

IActionResult BankWebApp.Controllers.HomeController.Login (LoginModel *loginModel*)

Handles the POST request for the Login view.

Parameters

loginModel	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**.

IActionResult 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**.

IActionResult BankWebApp.Controllers.HomeController.Privacy ()

Handles the GET request for the Privacy view.

Returns

The Privacy view.

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

IActionResult BankWebApp.Controllers.HomeController.Register ()

Handles the GET request for the Register view.

Returns

The Register view.

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

IActionResult BankWebApp.Controllers.HomeController.Register (RegisterModel *registerModel*)

Handles the POST request for the Register view.

Parameters

registerModel	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**.

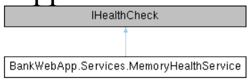
The documentation for this class was generated from the following file:

- 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

context	The context.
cancellationToken	The cancellation token.

Returns

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

Exceptions

Exception	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

httpContextAccessor	The HTTP context accessor.
userService	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

user	The user to sign in.
isPersistent	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.

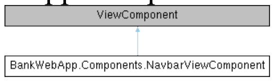
Definition at line **68** of file **MySignInManager.cs**.

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

args	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

AccountNumber	The account number to add funds to.
Amount	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

FromAcc	The account number to transfer money from.
ToAcc	The account number to transfer money to.
Amount	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

- IList< **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**.

IList<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.

- IList< **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)
- IList< **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.

- IList< **BankAccountModel** > **GetAllBankAccounts** ()
  Retrieves all bank accounts.

- IList< **RolesModel** > **GetRolesById** (int uid)
  Retrieves the roles of a user by their ID.

- IList< **TransactionModel** > **GetAllTransactions** ()
  Retrieves all transactions.

- IList< **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

IList< BankAccountModel > BankWebApp.Services.UserService.GetAllBankAccounts ()

Retrieves all bank accounts.

Returns

A list of all bank accounts.

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

IList< TransactionModel > BankWebApp.Services.UserService.GetAllTransactions ()

Retrieves all transactions.

Returns

A list of all transactions.

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

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

Retrieves the bank accounts of a user by their ID.

Parameters

uid	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

id	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**.

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

Retrieves the roles of a user by their ID.

Parameters

uid	The ID of the user.

Returns

A list of roles assigned to the user.

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

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

Retrieves the transactions of a bank account by its ID.

Parameters

accountId	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

id	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

username	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

newUser	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.

```
using Microsoft.AspNetCore.Mvc;

namespace BankWebApp.Components;

public class NavbarViewComponent : ViewComponent
{
    public IViewComponentResult Invoke()
    {
        return View();
    }
}
```

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.

```
using BankWebApp.Models;
using BankWebApp.Services;
using BankWebApp.Tools;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace BankWebApp.Controllers;

[Authorize]
public class AccountController : Controller
{
    private readonly ILogger<AccountController> _logger;
    private readonly UserService _userService;
    private readonly TransferService _transferService;

    public AccountController(ILogger<AccountController> logger, UserService userService,
                            TransferService transferService)
    {
        _logger = logger;
        _userService = userService;
        _transferService = transferService;
    }

    public IActionResult Index()
    {
        // users claims
        var claims = User.Claims.ToArray();

        // extract information from claims (username not needed)
        //string username = claims[0].Value;
        int id = claims[1].Value.ToInt32();

        var user = _userService.GetUserById(id)!;
        var bankAccounts = _userService.GetBankAccountsById(user.Id);

        var model = new AccountIndexModel()
        {
            SignedInUser = user,
            BankAccounts = bankAccounts
        };

        return View(model);
    }

    public IActionResult Transfer(bool? success = null, string? reason = null)
    {
        var model = TransferViewModel.Empty;
        var userId = User.Claims.ToArray()[1].Value.ToInt32();

        model.Success = success;
        model.Reason = reason;

        var bankAccounts = _userService.GetBankAccountsById(userId);
        model.BankAccounts = bankAccounts;

        return View(model);
    }

    [HttpPost]
    public IActionResult Transfer(TransferViewModel model)
    {
        var success = true;
        var reason = "";

        // check if from account is valid and the user owns it
        var fromAccount = _userService.GetBankAccountsById(model.FromAccountId);

        if (fromAccount == null)
        {
            success = false;
            reason = "Invalid account to send from.";
            return RedirectToAction("Transfer", new { Success = success, Reason = reason });
        }

        var userId = User.Claims.ToArray()[1].Value.ToInt32();
        var user = _userService.GetUserById(userId)!;
        var userOwnsAccount = fromAccount.UserId == user.Id;

        if (!userOwnsAccount)
        {
            success = false;
            reason = "You do not own this account.";

            return RedirectToAction("Transfer", new { Success = success, Reason = reason });
        }

        // now check if to account is valid

        var toAccount = _userService.GetBankAccountsById(model.ToAccountId);

        if (toAccount == null)
        {
            success = false;
            reason = "Invalid account to send to.";
            return RedirectToAction("Transfer", new { Success = success, Reason = reason });
        }

        // check if amount is valid

        if (model.Amount <= 0)
        {
            success = false;
            reason = "Invalid amount (cannot be less than 0).";
            return RedirectToAction("Transfer", new { Success = success, Reason = reason });
        }

        // check if from account has enough money

        if (fromAccount.Balance <= model.Amount)
        {
            success = false;
            reason = "Not enough money in account.";
            return RedirectToAction("Transfer", new { Success = success, Reason = reason });
        }

        // transfer money

        var transferCheck = _transferService.TransferMoney(fromAccount.AccountNumber, toAccount.AccountNumber, model.Amount);


        return RedirectToAction("Transfer", new { Success = success, Reason = reason });
    }

    [Authorize(Roles = "Admin")]
    public IActionResult AddFunds(bool? success = null, string? reason = null)
    {
        var model = new AddFundsViewModel()
        {
            BankAccounts = _userService.GetAllBankAccounts(),
            Success = success,
            Reason = reason
        };
        return View(model);
    }

    [HttpPost]
    [Authorize(Roles = "Admin")]
    public IActionResult AddFunds(AddFundsViewModel model)
    {
        bool success = true;
        string reason = "";

        // check if such account exists
        var bankAccount = _userService.GetBankAccountsById(model.SelectedBankAccountNumber);

        if (bankAccount == null)
        {
            success = false;
            reason = "Invalid account.";
            return RedirectToAction("AddFunds", new { Success = success, Reason = reason });
        }

        // check if amount is valid

        if (model.Amount <= 0)
        {
            success = false;
            reason = "Invalid amount (cannot be less than 0).";
            return RedirectToAction("AddFunds", new { Success = success, Reason = reason });
        }

        // add funds
        _transferService.PrintMoney(model.SelectedBankAccountNumber, model.Amount);

        return RedirectToAction("AddFunds", new { Success = success, Reason = reason });
    }

    [Authorize(Roles = "Admin")]
    public IActionResult ListUsers()
    {
        var model = new List<ListUsersViewModel>();

        var users = _userService.GetUsers();

        foreach (var user in users)
        {
            ListUsersViewModel userViewModel = new()
            {
                UserModel = user,
                BankAccounts = _userService.GetBankAccountsById(user.Id)
            };

            model.Add(userViewModel);
        }

        return View(model);
    }

    [Authorize(Roles = "Admin")]
    public IActionResult Show(string id)
    {
        if (!int.TryParse(id, out _))
        {
            return RedirectToAction("ListUsers");
        }

        var user = _userService.GetUserById(int.Parse(id));
        var bankAccounts = _userService.GetBankAccountsById(user!.Id);

        var transactions = bankAccounts.ToList().SelectMany(bankAccount =>
            _userService.GetTransactionsByAccountId(bankAccount.Id)).ToList();

        var model = new ListUsersViewModel()
        {
            UserModel = user,
            BankAccounts = bankAccounts,
            Transactions = transactions,
        };


        return View(model);
    }

    public IActionResult History()
    {
        AccountHistoryModel model = new();

        var myId = (User.Claims.ToArray()[1].Value).ToInt32();

        var bankAccounts = _userService.GetBankAccountsById(myId);

        // get all transactions from all bank accounts
        var transactions = bankAccounts.ToList().SelectMany(bankAccount =>
            _userService.GetTransactionsByAccountId(bankAccount.Id)).ToList();

        model.Transactions = transactions;

        return View(model);
    }
}
```

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.

```
using System.Text.Json;
using BankWebApp.Services;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Diagnostics.HealthChecks;

namespace BankWebApp.Controllers;

[Route("api/[controller]/[action]")]
public class HealthCheckController : ControllerBase
{
    private readonly ILogger<HealthCheckController> _logger;

    // Services used to check the health of the database, disk, and memory.
    private readonly DatabaseHealthService _dbService;
    private readonly DiskHealthService _diskService;
    private readonly MemoryHealthService _memoryService;


    public HealthCheckController(ILogger<HealthCheckController> logger,
        DatabaseHealthService dbService,
        DiskHealthService diskService,
        MemoryHealthService memoryService)
    {
        _logger = logger;
        _dbService = dbService;
        _diskService = diskService;
        _memoryService = memoryService;
    }


    [HttpGet]
    public async Task<IActionResult> All()
    {
        var DatabaseHealth = await CheckDatabaseHealth();
        var DiskHealth = await CheckDiskHealth();
        var RAMHealth = await CheckRAMHealth();

        var healthArray = new HealthCheckResult[]
        {
            DatabaseHealth.Item2,
            DiskHealth.Item2,
            RAMHealth.Item2
        };

        #region OverallHealthCheck
        HealthStatus overallHealth = HealthStatus.Healthy;

        foreach (var healthValue in healthArray)
        {
            if (healthValue.Status == HealthStatus.Unhealthy)
            {
                overallHealth = HealthStatus.Unhealthy;
                break;
            }
            else if (healthValue.Status == HealthStatus.Degraded)
            {
                overallHealth = HealthStatus.Degraded;
            }
        }
        #endregion

        var obj = new
        {
            OverallStatusInt = overallHealth,
            OverallStatus = overallHealth.ToString(),

            StatusEnum = new {
                HealthStatus.Healthy,
                HealthStatus.Degraded,
                HealthStatus.Unhealthy
            },

            DatabaseHealth = DatabaseHealth.Item2,
            DiskHealth = DiskHealth.Item2,
            RAMHealth = RAMHealth.Item2
        };

        var json = JsonSerializer.Serialize(obj);

        return overallHealth switch
        {
            HealthStatus.Healthy => Ok(json),
            HealthStatus.Degraded => BadRequest(json),
            HealthStatus.Unhealthy => BadRequest(json)
        };
    }

    [HttpGet]
    public async Task<IActionResult> Database()
    {
        return (await CheckDatabaseHealth()).Item1;
    }

    [HttpGet]
    public async Task<IActionResult> Disk()
    {
        return (await CheckDiskHealth()).Item1;
    }

    [HttpGet]
    public async Task<IActionResult> RAM()
    {
        return (await CheckRAMHealth()).Item1;
    }

    private async Task<(IActionResult, HealthCheckResult)> CheckDatabaseHealth()
    {
        var report = await _dbService.CheckHealthAsync(new (), new ());
        string json = JsonSerializer.Serialize(report);

        return report.Status switch
        {
            HealthStatus.Healthy => (Ok(json), report),
            HealthStatus.Degraded => (BadRequest(json), report),
            HealthStatus.Unhealthy => (BadRequest(json), report)
        };
    }

    private async Task<(IActionResult, HealthCheckResult)> CheckDiskHealth()
    {
        var report = await _diskService.CheckHealthAsync(new (), new ());
        string json = JsonSerializer.Serialize(report);

        return report.Status switch
        {
            HealthStatus.Healthy => (Ok(json), report),
            HealthStatus.Degraded => (BadRequest(json), report),
            HealthStatus.Unhealthy => (BadRequest(json), report)

        };
    }

    private async Task<(IActionResult, HealthCheckResult)> CheckRAMHealth()
    {
        var report = await _memoryService.CheckHealthAsync(new (), new ());
        string json = JsonSerializer.Serialize(report);

        return report.Status switch
        {
            HealthStatus.Healthy => (Ok(json), report),
            HealthStatus.Degraded => (BadRequest(json), report),
            HealthStatus.Unhealthy => (BadRequest(json), report)
        };
    }
}
```

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

Classes

class BankWebApp.Controllers.HomeController**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.

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.

```
using BankWebApp.Models;
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;
using BankWebApp.Services;
using Microsoft.AspNetCore.Identity;

namespace BankWebApp.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        private readonly UserService _userService;
        private readonly MySignInManager _signInManager;

        public HomeController(ILogger<HomeController> logger, UserService userService, MySignInManager signInManager)
        {
            _logger = logger;
            _userService = userService;
            _signInManager = signInManager;
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Privacy()
        {
            return View();
        }

        public IActionResult Login()
        {
            return View();
        }

        [HttpPost]
        public IActionResult Login(LoginModel loginModel)
        {
            if (ModelState.IsValid)
            {
                var Username = loginModel.Username;
                var Password = loginModel.Password;

                if (string.IsNullOrEmpty(Username) && !string.IsNullOrEmpty(Password))
                {
                    _logger.LogDebug($"Login failed for user {Username} (empty credentials)");
                    return RedirectToAction();
                }

                var user = _userService.GetUserByUsername(Username);
                if (user == null)
                {
                    _logger.LogDebug($"Login failed for user {Username} (account not found)");
                    return RedirectToAction();
                }

                if (!Tools.PasswordHashes.VerifyPassword(Password, user.PasswordHash))
                {
                    _logger.LogDebug($"Login failed for user {Username} (Password doesnt match)");
                    return RedirectToAction();
                }

                _signInManager.SignInAsync(user, loginModel.RememberMe).GetAwaiter().GetResult();

                _logger.LogDebug($"Login successful for user {Username}");

                return RedirectToAction("Index", "Account");

            }
            return View();
        }

        public IActionResult Logout()
        {
            var signedIn = User.Identity!.IsAuthenticated;
            if (!signedIn)
            {
                return RedirectToAction("Login");
            }

            _signInManager.SignOutAsync().GetAwaiter().GetResult(); //wait for sign out before redirecting
            return RedirectToAction("Login");
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }

        public IActionResult AccessDenied()
        {
            return View("Error", new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }

        public IActionResult Register()
        {
            return View();
        }

        [HttpPost]
        public IActionResult Register(RegisterModel registerModel)
        {
            if (ModelState.IsValid)
            {
                var resp = _userService.RegisterUser(registerModel);

                if (resp.success)
                {
                    _logger.LogDebug($"Registered user {registerModel.Username}");

                    registerModel.Success = true;

                    return RedirectToAction("Login");
                }
                else
                {
                    _logger.LogDebug($"Failed to register user {registerModel.Username}: {resp.reason}");
                    ModelState.AddModelError("Username", resp.reason);

                    registerModel.Success = false;
                    registerModel.Reason = resp.reason;

                    return View(registerModel);
                }
            }

            registerModel.Success = false;
            registerModel.Reason = string.Join("; ", ModelState.Values.SelectMany(v => v.Errors).Select(e => e.ErrorMessage));;

            return View(registerModel);
        }
    }
}
```

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.namespace BankWebApp.env;

public static class Envs
{
    public static string ConnectionString;
}
```

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.

```
namespace BankWebApp.Models;

public class AccountHistoryModel
{
    public IList<TransactionModel> Transactions { get; set; }
}
```

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

Classes

class BankWebApp.Models.AccountIndexModel**Namespaces**

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

AccountIndexModel.cs

Go to the documentation of this file.

```
namespace BankWebApp.Models;

public class AccountIndexModel
{
    public UserModel SignedInUser { get; set; }
    public IList<BankAccountModel> BankAccounts { get; set; }
}
```

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

Classes

class BankWebApp.Models.AddFundsViewModel**Namespaces**

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

AddFundsViewModel.cs

Go to the documentation of this file.

```
namespace BankWebApp.Models;

public class AddFundsViewModel
{
    public IList<BankAccountModel> BankAccounts { get; set; }

    public decimal Amount { get; set; }
    public string SelectedBankAccountNumber { get; set; }

    public bool? Success { get; set; }
    public string? Reason { get; set; }
}
```

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

Classes

class BankWebApp.Models.AddressModel**Namespaces**

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

AddressModel.cs

Go to the documentation of this file.

```
namespace BankWebApp.Models;

public class AddressModel
{
    public int Id { get; set; }
    public string Street { get; set; }
    public string City { get; set; }
    public string PostCode { get; set; }
    public string Country { get; set; }
}
```

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

Classes

class BankWebApp.Models.BankAccountModel**Namespaces**

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

BankAccountModel.cs

Go to the documentation of this file.
```
namespace BankWebApp.Models;

public class BankAccountModel
{
    public int Id { get; set; }
    public string AccountNumber { get; set; }
    public decimal Balance { get; set; }
    public int UserId { get; set; }
    public UserModel User { get; set; }
}
```

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

Classes

class BankWebApp.Models.ContactModel**Namespaces**

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

ContactModel.cs

Go to the documentation of this file.
```
namespace BankWebApp.Models;

public class ContactModel
{
    public int Id { get; set; }
    public string Email { get; set; }
    public string PhoneNumber { get; set; }
}
```

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

Classes

class BankWebApp.Models.ErrorViewModel**Namespaces**

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

ErrorViewModel.cs

Go to the documentation of this file.
```
namespace BankWebApp.Models
{
    public class ErrorViewModel
    {
        public string? RequestId { get; set; }

        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}
```

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

Classes

class BankWebApp.Models.ListUsersViewModel**Namespaces**

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

ListUsersViewModel.cs

Go to the documentation of this file.
```
namespace BankWebApp.Models;

public class ListUsersViewModel
{
    public UserModel UserModel { get; set; }
    public IList<BankAccountModel> BankAccounts { get; set; }

    public IList<TransactionModel> Transactions { get; set; }
}
```

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

Classes

class BankWebApp.Models.LoginModel**Namespaces**

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

LoginModel.cs

Go to the documentation of this file.
```
namespace BankWebApp.Models
{
    public class LoginModel
    {
        public string Username { get; set; }
        public string Password { get; set; }
        public bool RememberMe { get; set; }
    }
}
```

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

Classes

class BankWebApp.Models.RegisterModel**Namespaces**

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

RegisterModel.cs

Go to the documentation of this file.
```
using System.ComponentModel.DataAnnotations;

namespace BankWebApp.Models;

public class RegisterModel
{
    [MaxLength(50)]
    public string Username { get; set; }

    public string Password { get; set; }
    public string ConfirmPassword { get; set; }

    [EmailAddress]
    public string Email { get; set; }
    [Phone]
    public string PhoneNumber { get; set; }

    [MaxLength(50)]
    public string Street { get; set; }
    [MaxLength(45)]
    public string City { get; set; }
    [MaxLength(45)]
    public string PostCode { get; set; }
    [MaxLength(45)]
    public string Country { get; set; }


    public bool? Success { get; set; }
    public string? Reason { get; set; }
}
```

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

Classes

class BankWebApp.Models.RolesModel**Namespaces**

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

RolesModel.cs

Go to the documentation of this file.
```
namespace BankWebApp.Models;

public class RolesModel
{
    public int Id { get; set; }
    public string RoleName { get; set; }
    public int UserId { get; set; }
}
```

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

Classes

class BankWebApp.Models.TransactionModel**Namespaces**

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

TransactionModel.cs

Go to the documentation of this file.
```
namespace BankWebApp.Models;

public class TransactionModel
{
    public int Id { get; set; }
    public int SenderId { get; set; }
    public BankAccountModel Sender { get; set; }
    public int ReceiverId { get; set; }
    public BankAccountModel Receiver { get; set; }
    public decimal Amount { get; set; }
    public DateTime SentAt { get; set; }
}
```

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

Classes

class BankWebApp.Models.TransferViewModel**Namespaces**

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

TransferViewModel.cs

Go to the documentation of this file.

```
namespace BankWebApp.Models;

public class TransferViewModel
{
    public IList<BankAccountModel> BankAccounts { get; set; }

    public string FromAccountId { get; set; }
    public string ToAccountId { get; set; }

    public decimal Amount { get; set; }

    public bool? Success { get; set; }
    public string? Reason { get; set; }

    public static TransferViewModel Empty => new TransferViewModel();
}
```

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.
```
namespace BankWebApp.Models;

public class UserModel
{
    public int Id { get; set; }
    public string Username { get; set; }
    public string PasswordHash { get; set; }
    public DateTime CreatedAt { get; set; }
    public ContactModel Contact { get; set; }
    public AddressModel Address { get; set; }
}
```

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.

```
// <autogenerated />
using System;
using System.Reflection;
[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.

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

using System;
using System.Reflection;

[assembly: Microsoft.Extensions.Configuration.UserSecrets.UserSecretsIdAttribute("a8aecb95-9885-4362-be94-922d42874f8a")]
[assembly: System.Reflection.AssemblyCompanyAttribute("BankWebApp")]
[assembly: System.Reflection.AssemblyConfigurationAttribute("Debug")]
[assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
[assembly: System.Reflection.AssemblyInformationalVersionAttribute("1.0.0")]
[assembly: System.Reflection.AssemblyProductAttribute("BankWebApp")]
[assembly: System.Reflection.AssemblyTitleAttribute("BankWebApp")]
[assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]

// Generated by the MSBuild WriteCodeFragment class.

```

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.

```
// <auto-generated/>
global using global::Microsoft.AspNetCore.Builder;
global using global::Microsoft.AspNetCore.Hosting;
global using global::Microsoft.AspNetCore.Http;
global using global::Microsoft.AspNetCore.Routing;
global using global::Microsoft.Extensions.Configuration;
global using global::Microsoft.Extensions.DependencyInjection;
global using global::Microsoft.Extensions.Hosting;
global using global::Microsoft.Extensions.Logging;
global using global::System;
global using global::System.Collections.Generic;
global using global::System.IO;
global using global::System.Linq;
global using global::System.Net.Http;
global using global::System.Net.Http.Json;
global using global::System.Threading;
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.

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

using System;
using System.Reflection;

[assembly: 
Microsoft.AspNetCore.Mvc.ApplicationParts.ProvideApplicationPartFactoryAttribute("Microsoft.AspNetCore.Mvc.ApplicationParts.ConsolidatedAssemblyApplicationPartFact" +
    "ory, Microsoft.AspNetCore.Mvc.Razor")]

// Generated by the MSBuild WriteCodeFragment class.

```

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.

```
using BankWebApp.env;
using BankWebApp.Services;

namespace BankWebApp
{
    public class Program
    {
        public static void Main(string[] args)
        {
            // Initialize a new instance of the WebApplication builder with the provided command-line arguments.
            var builder = WebApplication.CreateBuilder(args);

            // Set the connection string from appsettings.json
            var connectionString = builder.Configuration.GetConnectionString("DefaultConnectionString");
            Envs.ConnectionString = connectionString!;

            // Register MVC controllers and views to the services.
            builder.Services.AddControllersWithViews();

            // Register application-specific services to the services container.
            builder.Services.AddSingleton<UserService>();
            builder.Services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();
            builder.Services.AddSingleton<MySignInManager>();
            builder.Services.AddSingleton<TransferService>();
            builder.Services.AddSingleton<DatabaseHealthService>();
            builder.Services.AddSingleton<DiskHealthService>();
            builder.Services.AddSingleton<MemoryHealthService>();

            // Register health checks to the services container.
            builder.Services.AddHealthChecks()
                .AddCheck<DatabaseHealthService>(nameof(DatabaseHealthService))
                .AddCheck<DiskHealthService>(nameof(DiskHealthService))
                .AddCheck<MemoryHealthService>(nameof(MemoryHealthService));

            // Configure authentication services with a custom scheme and cookie settings.
            builder.Services.AddAuthentication(options =>
            {
                options.DefaultScheme = "custom";
                options.DefaultSignInScheme = "custom";
                options.DefaultSignOutScheme = "custom";
                options.DefaultChallengeScheme = "custom";
            })
                .AddCookie("custom", options =>
            {
                options.LoginPath = "/Home/Login";
                options.LogoutPath = "/Home/Logout";
                options.AccessDeniedPath = "/Home/AccessDenied";
            });

            // Build the web application instance.
            var app = builder.Build();

            // Configure the HTTP request pipeline.
            if (!app.Environment.IsDevelopment())
            {
                // In non-development environments, use the exception handler middleware to handle exceptions globally.
                app.UseExceptionHandler("/Home/Error");
                // Use HSTS middleware to add the Strict-Transport-Security header to HTTP responses.
                app.UseHsts();
            }

            // Use HTTPS redirection middleware to redirect HTTP requests to HTTPS.
            app.UseHttpsRedirection();
            // Use static files middleware to serve static files.
            app.UseStaticFiles();

            // Use routing middleware to route requests to the correct endpoint.
            app.UseRouting();

            // Use authorization middleware to authorize users based on their roles and claims.
            app.UseAuthorization();

            // maps health checks to /health
            app.MapHealthChecks("/health");

            // Map the default controller route.
            app.MapControllerRoute(
                name: "default",
                pattern: "{controller=Home}/{action=Index}/{id?}");

            // Run the application.
            app.Run();
        }
    }
}
```

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.DatabaseHealthService***Service 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.

```
using Microsoft.Extensions.Diagnostics.HealthChecks;

namespace BankWebApp.Services;

public class DatabaseHealthService: IHealthCheck
{
    public Task<HealthCheckResult> CheckHealthAsync(HealthCheckContext context, CancellationToken cancellationToken = new CancellationToken())
    {

        try
        {
            bool alive;
            using (var db = new DatabaseService())
            {
                alive = db.Ping();
            }

            if (!alive)
            {
                return Task.FromResult(new HealthCheckResult(HealthStatus.Unhealthy, "Failed to connect to database."));
            }

            return Task.FromResult(new HealthCheckResult(HealthStatus.Healthy, "Database connection is healthy."));
        }
        catch (Exception e)
        {
            return Task.FromResult(new HealthCheckResult(HealthStatus.Unhealthy, "Failed to connect to database. (Exception)", e));
        }
    }
}
```

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.

```
using BankWebApp.env;
using Microsoft.Data.SqlClient;

namespace BankWebApp.Services
{
    public partial class DatabaseService : IDisposable
    {
        private readonly string _connectionString;

        private SqlConnection _connection;

        public DatabaseService()
        {
            _connectionString = Envs.ConnectionString;
            _connection = new SqlConnection(_connectionString);

            Open();
        }

        private void Open()
        {
            _connection.Open();
        }

        private void Close()
        {
            _connection.Close();
        }

        public void Dispose()
        {
            Close();
        }
    }
}
```

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.

```
using System.Data;
using BankWebApp.env;
using BankWebApp.Models;
using Microsoft.Data.SqlClient;

namespace BankWebApp.Services;

public partial class DatabaseService
{

    public IList<UserModel> GetUsers()
    {
        var users = new List<UserModel>();

        var sql = "SELECT [Users].Id, [Users].Username, [Users].PasswordHash, [Users].CreatedAt, [Users].ContactId, [Users].AddressId, " +
                        "[Contacts].Email, [Contacts].PhoneNumber, " +
                        "[Addresses].Street, [Addresses].City, [Addresses].PostCode, [Addresses].Country " +
                        "FROM [Users] " +
                        "INNER JOIN [Contacts] ON [Users].[ContactId] = [Contacts].[Id] " +
                        "INNER JOIN [Addresses] ON [Users].[AddressId] = [Addresses].[Id]";
        var cmd = new SqlCommand(sql, _connection);
        var reader = cmd.ExecuteReader();
        while (reader.Read())
        {
            users.Add(new UserModel
            {
                Id = reader.GetInt32(0),
                Username = reader.GetString(1),
                PasswordHash = reader.GetString(2),
                CreatedAt = reader.GetDateTime(3),
                Contact = new ContactModel
                {
                    Email = reader.GetString(6),
                    PhoneNumber = reader.GetString(7)
                },
                Address = new AddressModel
                {
                    Street = reader.GetString(8),
                    City = reader.GetString(9),
                    PostCode = reader.GetString(10),
                    Country = reader.GetString(11)
                }
            });
        }

        return users;
    }

    public bool UsernameExists(string _username)
    {
        var sql = "SELECT Username FROM Users WHERE Username = @username";
        var cmd = new SqlCommand(sql, _connection);
        cmd.Parameters.AddWithValue("@username", _username);
        var result = cmd.ExecuteScalar();

        return result != null;
    }

    public bool RegisterUser(UserModel user)
    {
        using (SqlConnection con = new SqlConnection(Envs.ConnectionString))
        {
            con.Open();
            var transaction = con.BeginTransaction();

            try
            {

                // Insert contact
                var sql = "INSERT INTO Contacts (Email, PhoneNumber) OUTPUT INSERTED.Id VALUES (@Email, @PhoneNumber)";
                var cmd = new SqlCommand(sql, con);
                cmd.Parameters.AddWithValue("@Email", user.Contact.Email);
                cmd.Parameters.AddWithValue("@PhoneNumber", user.Contact.PhoneNumber);
                cmd.Transaction = transaction;
                var contactId = (int)cmd.ExecuteScalar();

                // Insert address
                sql =
                    "INSERT INTO Addresses (Street, City, PostCode, Country) OUTPUT INSERTED.Id VALUES (@Street, @City, @PostCode, @Country)";
                cmd = new SqlCommand(sql, con);
                cmd.Parameters.AddWithValue("@Street", user.Address.Street);
                cmd.Parameters.AddWithValue("@City", user.Address.City);
                cmd.Parameters.AddWithValue("@PostCode", user.Address.PostCode);
                cmd.Parameters.AddWithValue("@Country", user.Address.Country);
                cmd.Transaction = transaction;
                var addressId = (int)cmd.ExecuteScalar();

                // Insert user
                sql =
                    "INSERT INTO Users (Username, PasswordHash, CreatedAt, ContactId, AddressId) OUTPUT INSERTED.Id VALUES (@Username, @PasswordHash, @CreatedAt, @ContactId, @AddressId)";
                cmd = new SqlCommand(sql, con);
                cmd.Parameters.AddWithValue("@Username", user.Username);
                cmd.Parameters.AddWithValue("@PasswordHash", user.PasswordHash);
                cmd.Parameters.AddWithValue("@CreatedAt", DateTime.Now);
                cmd.Parameters.AddWithValue("@ContactId", contactId);
                cmd.Parameters.AddWithValue("@AddressId", addressId);
                cmd.Transaction = transaction;
                var userId = cmd.ExecuteScalar();


                sql = "INSERT INTO BankAccount (UserId, AccountNumber, Balance) VALUES (@UserId, @AccountNumber, 0)";
                cmd = new SqlCommand(sql, con);
                cmd.Parameters.AddWithValue("@UserId", userId);
                cmd.Parameters.AddWithValue("@AccountNumber", Guid.NewGuid());
                cmd.Transaction = transaction;
                cmd.ExecuteNonQuery();

                transaction.Commit();

                return true;
            }
            catch (Exception e)
            {
                transaction.Rollback();
                return false;
            }
        }
    }

    public IList<BankAccountModel>? GetBankAccountById(int UserId)
    {
        List<BankAccountModel> accounts = new List<BankAccountModel>();

        var sql = "SELECT Id, AccountNumber, Balance, UserId FROM BankAccount WHERE UserId = @Id";
        var cmd = new SqlCommand(sql, _connection);
        cmd.Parameters.AddWithValue("@Id", UserId);
        var reader = cmd.ExecuteReader();

        if (reader.Read())
        {
            accounts.Add(new BankAccountModel
            {
                Id = reader.GetInt32(0),
                AccountNumber = reader.GetString(1),
                Balance = reader.GetDecimal(2),
                UserId = reader.GetInt32(3)
            });
        }

        var users = GetUsers();

        foreach (var account in accounts)
        {
            account.User = users.First(u => u.Id == account.UserId);
        }

        reader.Close();

        return accounts.Count > 0 ? accounts : null;
    }

    public BankAccountModel? GetBankAccountById(string Id)
    {
        var sql = "SELECT Id, AccountNumber, Balance, UserId FROM BankAccount WHERE AccountNumber = @Id";
        var cmd = new SqlCommand(sql, _connection);
        cmd.Parameters.AddWithValue("@Id", Id);
        var reader = cmd.ExecuteReader();

        if (reader.Read())
        {
            var result = new BankAccountModel
            {
                Id = reader.GetInt32(0),
                AccountNumber = reader.GetString(1),
                Balance = reader.GetDecimal(2),
                UserId = reader.GetInt32(3)
            };

            var users = GetUsers();

            result.User = users.First(u => u.Id == result.UserId);

            reader.Close();

            return result;
        }

        return null;
    }

    public BankAccountModel? GetBankAccountByAccountId(int Id)
    {
        var sql = "SELECT Id, AccountNumber, Balance, UserId FROM BankAccount WHERE Id = @Id";
        var cmd = new SqlCommand(sql, _connection);

        cmd.Parameters.AddWithValue("@Id", Id);

        var reader = cmd.ExecuteReader();

        if (reader.Read())
        {
            var result = new BankAccountModel
            {
                Id = reader.GetInt32(0),
                AccountNumber = reader.GetString(1),
                Balance = reader.GetDecimal(2),
                UserId = reader.GetInt32(3)
            };

            var users = GetUsers();

            result.User = users.First(u => u.Id == result.UserId);

            reader.Close();

            return result;
        }

        return null;
    }

    public bool TransferFunds(Guid from, Guid To, decimal Amount)
    {
        var fromAccount = GetBankAccountById(from.ToString());
        var toAccount = GetBankAccountById(To.ToString());

        using (SqlConnection con = new SqlConnection(Envs.ConnectionString))
        {
            con.Open();
            SqlTransaction transaction = con.BeginTransaction();

            try
            {
                var sql = "UPDATE BankAccount SET Balance = Balance - @Amount WHERE AccountNumber = @From";
                var cmd = new SqlCommand(sql, con);
                cmd.Parameters.AddWithValue("@Amount", Amount);
                cmd.Parameters.AddWithValue("@From", from);
                cmd.Transaction = transaction;
                cmd.ExecuteNonQuery();

                sql = "UPDATE BankAccount SET Balance = Balance + @Amount WHERE AccountNumber = @To";
                cmd = new SqlCommand(sql, con);
                cmd.Parameters.AddWithValue("@Amount", Amount);
                cmd.Parameters.AddWithValue("@To", To);
                cmd.Transaction = transaction;
                cmd.ExecuteNonQuery();

                sql = "INSERT INTO Transactions (SenderId, ReceiverId, Amount, SentAt) VALUES (@From, @To, @Amount, @SentAt)";
                cmd = new SqlCommand(sql, con);
                cmd.Parameters.AddWithValue("@From", fromAccount!.Id);
                cmd.Parameters.AddWithValue("@To", toAccount!.Id);
                cmd.Parameters.AddWithValue("@Amount", Amount);
                cmd.Parameters.AddWithValue("@SentAt", DateTime.Now);
                cmd.Transaction = transaction;
                cmd.ExecuteNonQuery();

                transaction.Commit();
                return true;
            }
            catch (Exception e)
            {
                transaction.Rollback();
                return false;
            }
        }
    }

    public IList<RolesModel> GetRolesById(int uid)
    {
        var roles = new List<RolesModel>();

        var sql = "SELECT [UserRoles].Id, [UserRoles].RoleName, [UserRoles].UserId FROM [UserRoles] WHERE [UserRoles].UserId = @Id";
        var cmd = new SqlCommand(sql, _connection);
        cmd.Parameters.AddWithValue("@Id", uid);
        var reader = cmd.ExecuteReader();
        while (reader.Read())
        {
            roles.Add(new RolesModel
            {
                Id = reader.GetInt32(0),
                RoleName = reader.GetString(1),
                UserId = reader.GetInt32(2)
            });
        }

        return roles;
    }

    public IList<BankAccountModel> GetAllBankAccounts()
    {
        var accounts = new List<BankAccountModel>();

        var sql = "SELECT Id, AccountNumber, Balance, UserId FROM BankAccount";
        var cmd = new SqlCommand(sql, _connection);
        var reader = cmd.ExecuteReader();
        while (reader.Read())
        {
            accounts.Add(new BankAccountModel
            {
                Id = reader.GetInt32(0),
                AccountNumber = reader.GetString(1),
                Balance = reader.GetDecimal(2),
                UserId = reader.GetInt32(3)
            });
        }

        var users = GetUsers();

        foreach (var account in accounts)
        {
            account.User = users.First(u => u.Id == account.UserId);
        }

        return accounts;
    }

    public void AddFunds(Guid guid, decimal amount)
    {
        var sql = "UPDATE BankAccount SET Balance = Balance + @Amount WHERE
AccountNumber = @AccountNumber";
        var cmd = new SqlCommand(sql, _connection);
        cmd.Parameters.AddWithValue("@Amount", amount);
        cmd.Parameters.AddWithValue("@AccountNumber", guid);
        cmd.ExecuteNonQuery();

    }

    public IList<TransactionModel> GetTransactions()
    {
        var sql = "SELECT Id, SenderId, ReceiverId, Amount, SentAt FROM
Transactions";
        var cmd = new SqlCommand(sql, _connection);

        var transactions = new List<TransactionModel>();

        var reader = cmd.ExecuteReader();

        while (reader.Read())
        {
            transactions.Add(new TransactionModel
            {
                Id = reader.GetInt32(reader.GetOrdinal("Id")),
                SenderId = reader.GetInt32(reader.GetOrdinal("SenderId")),
                ReceiverId = reader.GetInt32(reader.GetOrdinal("ReceiverId")),
                Amount = reader.GetDecimal(reader.GetOrdinal("Amount")),
                SentAt = reader.GetDateTime(reader.GetOrdinal("SentAt"))
            });
        }

        reader.Close();

        foreach (var transaction in transactions)
        {
            transaction.Sender = GetBankAccountByAccountId(transaction.SenderId);
            transaction.Receiver =
GetBankAccountByAccountId(transaction.ReceiverId);
        }

        return transactions;
    }

    public IList<TransactionModel> GetTransactions(int uid)
    {
        var sql = "SELECT Id, SenderId, ReceiverId, Amount, SentAt FROM Transactions
WHERE SenderId = @Id OR ReceiverId = @Id";
        var cmd = new SqlCommand(sql, _connection);

        cmd.Parameters.AddWithValue("@Id", uid);

        var transactions = new List<TransactionModel>();

        var reader = cmd.ExecuteReader();

        while (reader.Read())
        {
            transactions.Add(new TransactionModel
            {
                Id = reader.GetInt32(reader.GetOrdinal("Id")),
                SenderId = reader.GetInt32(reader.GetOrdinal("SenderId")),
                ReceiverId = reader.GetInt32(reader.GetOrdinal("ReceiverId")),
                Amount = reader.GetDecimal(reader.GetOrdinal("Amount")),
                SentAt = reader.GetDateTime(reader.GetOrdinal("SentAt"))
            });
        }

        reader.Close();

        foreach (var transaction in transactions)
        {
            transaction.Sender = GetBankAccountByAccountId(transaction.SenderId);
            transaction.Receiver =
GetBankAccountByAccountId(transaction.ReceiverId);
        }

        return transactions;
    }

    public bool Ping()
    {
        try
        {
            var sql = "SELECT 1";
            using (var cmd = new SqlCommand(sql, _connection))
            {
                var response = cmd.ExecuteScalar();

                // check if the response is 1
                if (response is int i && i == 1)
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }

        }
        catch (Exception e)
        {
            return false;
        }
    }
}
```

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

Classes

class **BankWebApp.Services.DiskHealthService***Class 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.

```
using Microsoft.Extensions.Diagnostics.HealthChecks;

namespace BankWebApp.Services;

public class DiskHealthService : IHealthCheck
{
    private const long MinimumFreeSpace = 1000000000; // 1 GB

    private const long CriticalFreeSpace = 500000000; // 500 MB



    public Task<HealthCheckResult> CheckHealthAsync(HealthCheckContext context, CancellationToken cancellationToken)
    {
        try
        {
            var driveInfo = new DriveInfo(Path.GetPathRoot(Directory.GetCurrentDirectory()));
            var free = driveInfo.AvailableFreeSpace;

            if (free < MinimumFreeSpace)
            {
                return Task.FromResult(HealthCheckResult.Unhealthy($"Not enough free disk space. (Minimum: {MinimumFreeSpace} bytes, Actual: {free} bytes)"));
            }
            else if (free < CriticalFreeSpace)
            {
                return Task.FromResult(HealthCheckResult.Degraded($"Low disk space. (Minimum: {MinimumFreeSpace} bytes, Actual: {free} bytes)"));
            }
            else
            {
                return Task.FromResult(HealthCheckResult.Healthy($"Disk has enough free space. (Minimum: {MinimumFreeSpace} bytes, Actual: {free} bytes)"));
            }

        }
        catch (Exception e)
        {
            return Task.FromResult(HealthCheckResult.Unhealthy("Failed to check disk health. (Exception)", e));
        }
    }
}
```

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

Classes

class **BankWebApp.Services.MemoryHealthService***Class 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.

```
using Microsoft.Extensions.Diagnostics.HealthChecks;

namespace BankWebApp.Services;


public class MemoryHealthService : IHealthCheck
{
    private const long MinimumFreeMemory = 5_000_000; // 5 MB

    private const long CriticalFreeMemory = 1_000_000; // 1 MB

    public Task<HealthCheckResult> CheckHealthAsync(HealthCheckContext context, CancellationToken cancellationToken)
    {
        try
        {
            var free = GC.GetTotalMemory(false);

            if (free < MinimumFreeMemory)
            {
                return Task.FromResult(HealthCheckResult.Unhealthy($"Not enough free memory. (Minimum: {MinimumFreeMemory} bytes, Actual: {free} bytes)"));
            }
            else if (free < CriticalFreeMemory)
            {
                return Task.FromResult(HealthCheckResult.Degraded($"Low memory. (Minimum: {MinimumFreeMemory} bytes, Actual: {free} bytes)"));
            }
            else
            {
                return Task.FromResult(HealthCheckResult.Healthy($"Memory has enough free space. (Minimum: {MinimumFreeMemory} bytes, Actual: {free} bytes)"));
            }

        }
        catch (Exception e)
        {
            return Task.FromResult(HealthCheckResult.Unhealthy("Failed to check memory health. (Exception)", e));
        }
    }
}
```

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.

```
using System.Security.Claims;
using BankWebApp.Models;
using Microsoft.AspNetCore.Authentication;

namespace BankWebApp.Services
{
    public class MySignInManager
    {
        private readonly IHttpContextAccessor _httpContextAccessor;
        private readonly UserService _userService;

        public MySignInManager(IHttpContextAccessor httpContextAccessor, UserService userService)
        {
            _httpContextAccessor = httpContextAccessor;
            _userService = userService;
        }

        public async Task SignInAsync(UserModel user, bool isPersistent = false)
        {
            var roles = _userService.GetRolesById(user.Id);

            // Create a claims identity
            var claims = new List<Claim>
            {
                new Claim(ClaimTypes.Name, user.Username),
                new Claim(ClaimTypes.PrimarySid, user.Id.ToString()),
            };

            claims.AddRange(roles.Select(
                role => new Claim(ClaimTypes.Role, role.RoleName)
                ));

            var claimsIdentity = new ClaimsIdentity(claims, "custom");

            // Create a claims principal
            var claimsPrincipal = new ClaimsPrincipal(claimsIdentity);

            // Sign in the user
            await _httpContextAccessor.HttpContext.SignInAsync("custom", claimsPrincipal, new AuthenticationProperties
            {
                IsPersistent = isPersistent,
                ExpiresUtc = DateTime.UtcNow.AddMinutes(30) // Set expiration as needed
            });

        }

        public async Task SignOutAsync()
        {
            // Sign out the user
            await _httpContextAccessor.HttpContext.SignOutAsync("custom");
        }
    }
}
```

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.
```
namespace BankWebApp.Services;
public class TransferService
{
    private readonly DatabaseService _databaseService;

    public TransferService()
    {
        _databaseService = new DatabaseService();
    }

    public (bool Success, string Reason) TransferMoney(string FromAcc, string ToAcc,
decimal Amount)
    {
        var fromAcc = _databaseService.GetBankAccountById(FromAcc);
        var toAcc = _databaseService.GetBankAccountById(ToAcc);

        if (fromAcc == null)
        {
            return (false, "Invalid account to send from.");
        }
        else if (toAcc == null)
        {
            return (false, "Invalid account to send to.");
        }

        if (fromAcc.Balance <= Amount)
        {
            return (false, "Insufficient funds.");
        }

        Guid fromGuid = Guid.Parse(fromAcc.AccountNumber);
        Guid toGuid = Guid.Parse(toAcc.AccountNumber);

        var success = _databaseService.TransferFunds(fromGuid, toGuid, Amount);

        if (!success)
        {
            return (false, "An error occurred while transferring funds.");
        }
        else
        {
            return (true, "");
        }
    }

    public void PrintMoney(string AccountNumber, decimal Amount)
    {
        var account = _databaseService.GetBankAccountById(AccountNumber)!;

        Guid guid = Guid.Parse(account.AccountNumber);

        _databaseService.AddFunds(guid, Amount);
    }
}
```

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

Classes

class **BankWebApp.Services.UserService***Service 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.
```
using BankWebApp.Models;
using BankWebApp.Tools;

namespace BankWebApp.Services
{
    public class UserService
    {
        // Database service instance for database operations
        private readonly DatabaseService _databaseService;

        // The time when the cache was last updated
        private DateTime _lastUpdateAt;

        // Cache for storing user data
        private IList<UserModel>? _users = null;

        // Duration for which the cache is valid
        private readonly TimeSpan _cacheDuration = TimeSpan.FromMinutes(1);

        public UserService()
        {
            _databaseService = new DatabaseService();
            RefreshCache();
        }

        public IList<UserModel> GetUsers()
        {
            if (_users == null || (_lastUpdateAt + _cacheDuration) < DateTime.Now)
            {
                RefreshCache();
            }

            return _users!;
        }

        private void RefreshCache()
        {
            _users = _databaseService.GetUsers();
            _lastUpdateAt = DateTime.Now;
        }

        public UserModel? GetUserById(int id)
        {
            return _users?.FirstOrDefault(user => user.Id == id);
        }

        public UserModel? GetUserByUsername(string username)
        {
            return _users?.FirstOrDefault(user => user.Username == username);
        }

        public (bool success, string reason) RegisterUser(RegisterModel newUser)
        {
            bool existCheck = _databaseService.UsernameExists(newUser.Username);
            if (existCheck)
            {
                return (false, "Username already exists");
            }

            bool passwordMatches = (newUser.Password == newUser.ConfirmPassword);
            if (!passwordMatches)
            {
                return (false, "Passwords do not match");
            }

            var NewUser = new UserModel()
            {
                // Id auto generated
                Username = newUser.Username,
                PasswordHash = newUser.Password.HashPassword(),
                Contact = new ContactModel()
                {
                    // Id auto generated
                    Email = newUser.Email,
                    PhoneNumber = newUser.PhoneNumber
                },
                Address = new AddressModel()
                {
                    // Id auto generated
                    Street = newUser.Street,
                    City = newUser.City,
                    PostCode = newUser.PostCode,
                    Country = newUser.Country
                }
                // CreatedAt auto generated
            };

            var registerCheck = _databaseService.RegisterUser(NewUser);

            if (registerCheck)
            {
                RefreshCache();
                return (true, "");
            }
            else
            {
                return (false, "Something went wrong during the registering process");
            }
        }

        public IList<BankAccountModel> GetBankAccountsById(int uid)
        {
            return _databaseService.GetBankAccountById(uid);
        }

        public BankAccountModel? GetBankAccountsById(string id)
        {
            return _databaseService.GetBankAccountById(id);
        }

        public IList<BankAccountModel> GetAllBankAccounts()
        {
            return _databaseService.GetAllBankAccounts();
        }

        public IList<RolesModel> GetRolesById(int uid)
        {
            return _databaseService.GetRolesById(uid);
        }

        public IList<TransactionModel> GetAllTransactions()
        {
            return _databaseService.GetTransactions();
        }

        public IList<TransactionModel> GetTransactionsByAccountId(int accountId)
        {
            return _databaseService.GetTransactions(accountId);
        }

    }
}
```

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.
```
namespace BankWebApp.Tools;

public static class ClaimTools
{
    public static int ToInt32(this string str) => int.Parse(str);
}
```

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.

```
using System.Runtime.Intrinsics.Arm;
using System.Security.Cryptography;
using System.Text;

namespace BankWebApp.Tools;

public static class PasswordHashes
{
    public static string HashPassword(this string text)
    {
        return BCrypt.Net.BCrypt.HashPassword(text);
    }

    public static bool VerifyPassword(string unhashedp, string p2)
    {
        return BCrypt.Net.BCrypt.Verify(unhashedp, p2);
    }
}
```

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

Index

INDEX