Namespace BankBackend.Models

Classes

Account

Transaction

User

Enums

AccountType

Class Account

Namespace: BankBackend.Models

Assembly: BankBackend.dll

```
public class Account
```

Inheritance

object ← Account

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

Account()

```
public Account()
```

Properties

AccountId

```
[Key]
public int AccountId { get; set; }
```

Property Value

int

Balance

```
public double Balance { get; set; }
```

Property Value

double

PrimaryUserId

```
[ForeignKey("UserId")]
public int PrimaryUserId { get; set; }
```

Property Value

int

Type

```
public AccountType Type { get; set; }
```

Property Value

AccountType

Users

```
public List<User> Users { get; set; }
```

Property Value

List<User>

Enum AccountType

Namespace: BankBackend.Models

Assembly: BankBackend.dll

```
public enum AccountType
```

Fields

CHECKING = 1

CLOWN = 2

SAVINGS = 0

Class Transaction

Namespace: BankBackend.Models

Assembly: BankBackend.dll

```
public class Transaction
```

Inheritance

object ← Transaction

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

Transaction()

```
public Transaction()
```

Transaction(Account, Account, double)

```
public Transaction(Account fromAccount, Account toAccount, double amount)
```

Parameters

`fromAccount` Account

`toAccount` Account

`amount` double

Transaction(Account, double)

```
public Transaction(Account account, double amount)
```

Parameters

`account` Account

`amount` double

Properties

Amount

```
public double Amount { get; }
```

Property Value

double

FromAccount

```
public Account FromAccount { get; }
```

Property Value

Account

Time

```
public DateTime Time { get; }
```

Property Value

DateTime

ToAccount

```
public Account? ToAccount { get; }
```

Property Value

Account

TransactionId

```
[Key]
public int TransactionId { get; set; }
```

Property Value

int

Class User

Namespace: BankBackend.Models

Assembly: BankBackend.dll

```
public class User
```

Inheritance

object ← User

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

User()

```
public User()
```

User(string, string)

```
public User(string password, string name)
```

Parameters

`password` string

`name` string

Properties

Accounts

```
public List<Account> Accounts { get; set; }
```

Property Value

List<Account>

Name

```
public string Name { get; set; }
```

Property Value

string

Password

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

Property Value

string

UserId

```
[Key]
public int UserId { get; set; }
```

Property Value

int

Namespace BankBackend.Repository

Classes

BankContext

BankRepository

Interfaces

IBankRepository

Class BankContext

Namespace: BankBackend.Repository

Assembly: BankBackend.dll

```
public class BankContext : DbContext, IInfrastructure<IServiceProvider>,
IDbContextDependencies, IDbSetCache, IDbContextPoolable, IResettableService,
IDisposable, IAsyncDisposable
```

Inheritance

object ← DbContext ← BankContext

Implements

IInfrastructure<IServiceProvider>, IDbContextDependencies, IDbSetCache, IDbContextPoolable, IResettableService, IDisposable, IAsyncDisposable

Inherited Members

DbContext.Set<TEntity>() , DbContext.Set<TEntity>(string) , DbContext.OnConfiguring(DbContextOptionsBuilder) , DbContext.ConfigureConventions(ModelConfigurationBuilder) , DbContext.OnModelCreating(ModelBuilder) , DbContext.SaveChanges() , DbContext.SaveChanges(bool) , DbContext.SaveChangesAsync(CancellationToken) , DbContext.SaveChangesAsync(bool, CancellationToken) , DbContext.Dispose() , DbContext.DisposeAsync() , DbContext.Entry<TEntity>(TEntity) , DbContext.Entry(object) , DbContext.Add<TEntity>(TEntity) , DbContext.AddAsync<TEntity>(TEntity, CancellationToken) , DbContext.Attach<TEntity>(TEntity) , DbContext.Update<TEntity>(TEntity) , DbContext.Remove<TEntity>(TEntity) , DbContext.Add(object) , DbContext.AddAsync(object, CancellationToken) , DbContext.Attach(object) , DbContext.Update(object) , DbContext.Remove(object) , DbContext.AddRange(params object[]) , DbContext.AddRangeAsync(params object[]) , DbContext.AttachRange(params object[]) , DbContext.UpdateRange(params object[]) , DbContext.RemoveRange(params object[]) , DbContext.AddRange(IEnumerable<object>) , DbContext.AddRangeAsync(IEnumerable<object>, CancellationToken) , DbContext.AttachRange(IEnumerable<object>) , DbContext.UpdateRange(IEnumerable<object>) , DbContext.RemoveRange(IEnumerable<object>) , DbContext.Find(Type, params object[]) , DbContext.FindAsync(Type, params object[]) ,

DbContext.FindAsync(Type, object[], CancellationToken) , DbContext.Find<TEntity>(params object[]) , DbContext.FindAsync<TEntity>(params object[]) , DbContext.FindAsync<TEntity>(object[], CancellationToken) , DbContext.FromExpression<TResult>(Expression<Func<IQueryable<TResult>>>) , DbContext.Database , DbContext.ChangeTracker , DbContext.Model , DbContext.ContextId , DbContext.SavingChanges , DbContext.SavedChanges , DbContext.SaveChangesFailed , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

BankContext(DbContextOptions<BankContext>)

```
public BankContext(DbContextOptions<BankContext> options)
```

Parameters

`options` DbContextOptions<BankContext>

Properties

Accounts

```
public DbSet<Account> Accounts { get; }
```

Property Value

DbSet<Account>

Transactions

```
public DbSet<Transaction> Transactions { get; }
```

Property Value

DbSet<Transaction>

Users

```
public DbSet<User> Users { get; }
```

Property Value

DbSet<User>

Class BankRepository

Namespace: BankBackend.Repository

Assembly: BankBackend.dll

```
public class BankRepository : IBankRepository
```

Inheritance

object ← BankRepository

Implements

IBankRepository

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

BankRepository(string)

```
public BankRepository(string connectionString)
```

Parameters

`connectionString` string

Methods

AddAccountToUser(Account, int)

adds account with `account` to the account list of the user with `userId`

```
public User? AddAccountToUser(Account account, int userId)
```

Parameters

`account` Account

`userId` int

Returns

User

the updated user with the added account, null if the user does not exist

AddUserToAccount(User, int)

adds user with `userId` to the account list of the account with `account`

```
public Account? AddUserToAccount(User user, int accountId)
```

Parameters

`user` User

`accountId` int

Returns

Account

the updated account with the new user, null if account does not exist

CreateAccount(Account)

uploads the account to database

```
public Account CreateAccount(Account account)
```

Parameters

`account` Account

Returns

Account

the account that was just created

CreateTransaction(Transaction)

uploads the transaction to database

```
public Transaction CreateTransaction(Transaction transaction)
```

Parameters

`transaction` Transaction

Returns

Transaction

the transaction that was just created

CreateUser(User)

uploads the user to database

```
public User CreateUser(User user)
```

Parameters

`user` User

Returns

User

the user that was just created

DeleteAccountById(int)

deletes the account with `accountId`

```
public Account? DeleteAccountById(int accountId)
```

Parameters

`accountId` int

Returns

Account

the account that was just deleted, null if the account does not exist

DeleteAccountUserByUserId(int, int)

deletes the user with `userId` from the account with `accountId`

```
public User? DeleteAccountUserByUserId(int accountId, int userId)
```

Parameters

`accountId` int

`userId` int

Returns

User

the user that was just deleted, null if the account does not exist, or if the account does not have an user with `userId`

DeleteTransactionByTransactionId(int)

deletes the transaction with `transactionId`

```
public Transaction? DeleteTransactionByTransactionId(int transactionId)
```

Parameters

`transactionId` int

Returns

Transaction

the transaction that was just deleted, null if the transaction does not exist

DeleteUserAccountByAccountId(int, int)

deletes the account with `accountId` from the user with `userId`

```
public Account? DeleteUserAccountByAccountId(int userId, int accountId)
```

Parameters

`userId` int

`accountId` int

Returns

Account

the account that was just deleted, null if user does not exist, or the user does not have an account with `accountId`

DeleteUserById(int)

deletes the user with `userId`

```
public User? DeleteUserById(int userId)
```

Parameters

`userId` int

Returns

User

the user that was just deleted, null if the user does not exist

GetAccountByAccountId(int)

find an account with `accountId`

```
public Account? GetAccountByAccountId(int accountId)
```

Parameters

`accountId` int

Returns

Account

the account with `accountId`, null if account does not exist

GetAccountsByUserId(int)

find all the accounts of the user with `userId`

```
public List<Account>? GetAccountsByUserId(int userId)
```

Parameters

`userId` int

Returns

List<Account>

a list containing all the accounts, null if user does not exist

GetAllAccounts()

find all existing accounts in the database

```
public List<Account> GetAllAccounts()
```

Returns

List<Account>

list containing all existing accounts, empty list if non exists

GetAllTransactions()

find all existing transactions in the database

```
public List<Transaction> GetAllTransactions()
```

Returns

List<Transaction>

list containing all existing transactions, empty list if non exists

GetAllUsers()

find all existing users in the database

```
public List<User> GetAllUsers()
```

Returns

List<User>

a list containing all existing users, empty list if non exists

GetPrimaryAccountsByUserId(int)

find all the accounts that the user with `accountId` is a primary user

```
public List<Account>? GetPrimaryAccountsByUserId(int userId)
```

Parameters

`userId` int

Returns

List<Account>

a list containing all the primary accounts, null if user does not exist

GetTransactionByTransactionId(int)

find a transaction with `transactionId`

```
public Transaction? GetTransactionByTransactionId(int transactionId)
```

Parameters

`transactionId` int

Returns

Transaction

the transaction with `transactionId`, null if transaction does not exist

GetTransactionsByFromAccount(int)

```
public List<Transaction> GetTransactionsByFromAccount(int fromAccountId)
```

Parameters

`fromAccountId` int

Returns

List<Transaction>

GetTransactionsByToAccountId(int)

```
public List<Transaction> GetTransactionsByToAccountId(int toAccountId)
```

Parameters

`toAccountId` int

Returns

List<Transaction>

GetUserByUserId(int)

find a user with `userId`

```
public User? GetUserByUserId(int userId)
```

Parameters

`userId` int

Returns

User

the user with the Id, null if user does not exist

GetUsersByAccountId(int)

find all the users of the account with `accountId`

```
public List<User>? GetUsersByAccountId(int accountId)
```

Parameters

`accountId` int

Returns

List<User>

a list containing all the users, null if account does not exist

UpdateBalance(int, double)

updates the balance of the account with `account` to `balance`

```
public Account? UpdateBalance(int accountId, double balance)
```

Parameters

`accountId` int

`balance` double

Returns

Account

the updated account with the new balance, null if the account does not exist

UpdateName(int, string)

updates the name of the user with `userId` to `name`

```
public User? UpdateName(int userId, string name)
```

Parameters

`userId` int

`name` string

Returns

User

the updated user with the new name, null if the user does not exist

UpdatePassword(int, string)

updates the password of user with `userId` to `password`

```
public User? UpdatePassword(int userId, string password)
```

Parameters

`userId` int

`password` string

Returns

User

the updated user with the new password, null if the user does not exist

UpdatePrimaryUser(int, int)

replaces the primary user of the account with `account` with `userId`

```
public Account? UpdatePrimaryUser(int accountId, int userId)
```

Parameters

`accountId` int

`userId` int

Returns

Account

the updated account with the new primary user, null if account doesnot exist or if user does not exist

Exceptions

NotImplementedException

Interface IBankRepository

Namespace: BankBackend.Repository

Assembly: BankBackend.dll

```
public interface IBankRepository
```

Methods

AddAccountToUser(Account, int)

```
User? AddAccountToUser(Account account, int userId)
```

Parameters

`account` Account

`userId` int

Returns

User

AddUserToAccount(User, int)

```
Account? AddUserToAccount(User user, int accountId)
```

Parameters

`user` User

`accountId` int

Returns

Account

CreateAccount(Account)

```
Account CreateAccount(Account account)
```

Parameters

`account` Account

Returns

Account

CreateTransaction(Transaction)

```
Transaction CreateTransaction(Transaction transaction)
```

Parameters

`transaction` Transaction

Returns

Transaction

CreateUser(User)

```
User CreateUser(User user)
```

Parameters

`user` User

Returns

User

DeleteAccountById(int)

```
Account? DeleteAccountById(int accountId)
```

Parameters

`accountId` int

Returns

Account

DeleteAccountUserByUserId(int, int)

```
User? DeleteAccountUserByUserId(int accountId, int userId)
```

Parameters

`accountId` int

`userId` int

Returns

User

DeleteTransactionByTransactionId(int)

```
Transaction? DeleteTransactionByTransactionId(int transactionId)
```

Parameters

`transactionId` int

Returns

Transaction

DeleteUserAccountByAccountId(int, int)

```
Account? DeleteUserAccountByAccountId(int userId, int accountId)
```

Parameters

`userId` int

`accountId` int

Returns

Account

DeleteUserById(int)

```
User? DeleteUserById(int userId)
```

Parameters

`userId` int

Returns

User

GetAccountByAccountId(int)

```
Account? GetAccountByAccountId(int accountId)
```

Parameters

`accountId` int

Returns

Account

GetAccountsByUserId(int)

```
List<Account>? GetAccountsByUserId(int userId)
```

Parameters

`userId` int

Returns

List<Account>

GetAllAccounts()

```
List<Account> GetAllAccounts()
```

Returns

List<Account>

GetAllTransactions()

```
List<Transaction> GetAllTransactions()
```

Returns

List<Transaction>

GetAllUsers()

```
List<User> GetAllUsers()
```

Returns

List<User>

GetPrimaryAccountsByUserId(int)

```
List<Account>? GetPrimaryAccountsByUserId(int userId)
```

Parameters

`userId` int

Returns

List<Account>

GetTransactionByTransactionId(int)

```
Transaction? GetTransactionByTransactionId(int transactionId)
```

Parameters

`transactionId` int

Returns

Transaction

GetTransactionsByFromAccount(int)

```
List<Transaction> GetTransactionsByFromAccount(int fromAccountId)
```

Parameters

`fromAccountId` int

Returns

List<Transaction>

GetTransactionsByToAccountId(int)

```
List<Transaction> GetTransactionsByToAccountId(int toAccountId)
```

Parameters

`toAccountId` int

Returns

List<Transaction>

GetUserByUserId(int)

```
User? GetUserByUserId(int userId)
```

Parameters

`userId` int

Returns

User

GetUsersByAccountId(int)

```
List<User>? GetUsersByAccountId(int accountId)
```

Parameters

`accountId` int

Returns

List<User>

UpdateBalance(int, double)

```
Account? UpdateBalance(int accountId, double balance)
```

Parameters

`accountId` int

`balance` double

Returns

Account

UpdateName(int, string)

```
User? UpdateName(int userId, string name)
```

Parameters

`userId` int

`name` string

Returns

User

UpdatePassword(int, string)

```
User? UpdatePassword(int userId, string password)
```

Parameters

`userId` int

`password` string

Returns

User

UpdatePrimaryUser(int, int)

```
Account? UpdatePrimaryUser(int accountId, int userId)
```

Parameters

`accountId` int

`userId` int

Returns

Account

Namespace Bank_Backend.Migrations

Classes

Initial

A base class inherited by each EF Core migration.

Class Initial

Namespace: Bank_Backend.Migrations

Assembly: BankBackend.dll

A base class inherited by each EF Core migration.

```
[DbContext(typeof(BankContext))]
[Migration("20240805163459_Initial")]
public class Initial : Migration
```

Inheritance

object ← Migration ← Initial

Inherited Members

Migration.InitialDatabase , Migration.TargetModel , Migration.UpOperations , Migration.DownOperations , Migration.ActiveProvider , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

See Database migrations for more information and examples.

Methods

BuildTargetModel(ModelBuilder)

Implemented to build the TargetModel.

```
protected override void BuildTargetModel(ModelBuilder modelBuilder)
```

Parameters

`modelBuilder` ModelBuilder

The ModelBuilder to use to build the model.

Remarks

See Database migrations for more information and examples.

Down(MigrationBuilder)

Builds the operations that will migrate the database 'down'.

```
protected override void Down(MigrationBuilder migrationBuilder)
```

Parameters

`migrationBuilder` MigrationBuilder

The MigrationBuilder that will build the operations.

Remarks

That is, builds the operations that will take the database from the state left in by this migration so that it returns to the state that it was in before this migration was applied.

This method must be overridden in each class that inherits from Migration if both 'up' and 'down' migrations are to be supported. If it is not overridden, then calling it will throw and it will not be possible to migrate in the 'down' direction.

See Database migrations for more information and examples.

Up(MigrationBuilder)

Builds the operations that will migrate the database 'up'.

```
protected override void Up(MigrationBuilder migrationBuilder)
```

Parameters

`migrationBuilder` MigrationBuilder

The MigrationBuilder that will build the operations.

Remarks

That is, builds the operations that will take the database from the state left in by the previous migration so that it is up-to-date with regard to this migration.

This method must be overridden in each class that inherits from Migration.

See Database migrations for more information and examples.