Міністерство науки і освіти України

Житомирський державний технологічний університет

Кафедра інженерії програмного забезпечення

КУРСОВИЙ ПРОЕКТ

з дисципліни «Архітектура та проектування ПЗ»

на тему: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студента (ки) 3 курсу ПІ-52 групи

спеціальності 6.05010301 «Програмна інженерія»

Ксенича Євгенія Геннадійовича

(прізвище та ініціали)

Керівник: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

Члени комісії: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ І.І. Сугоняк

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.O. Данильченко

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Н. Ковальчук

м. Житомир – 2017 рік

# ЗМІСТ

ВСТУП………….………………………………………………………………….4

1. Теоретичні відомості…………………………………………………………

2. Технічне завдання……………………………………………………………

3. Опис програмного продукту…………………………………………………

4. Керівництво користувачу…………………………………………………….

5. Керівництво програмісту…………………………………………………….

ВИСНОВКИ………………………………………………………………….......32

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…………………………………..……33

ДОДАТКИ…………………………………………………………………..…....34

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ORM - Object-relational mapping - технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних».

EF - Entity Framework – технологія доступу до даних, що базується на принципах ORM.

LINQ - Language Integrated Query – мова запитів, що нагадує SQL, та реалізована у .Net.

ВСТУП

Шаблони проектування **-** ефектні способи вирішення задач проектування програмного забезпечення, які вирішують найрозповсюдженіші проблеми проектування.

На практиці використання патернів проектування відбувається дуже часто і якщо не є обов’язковим, то дуже бажаним.

Метою даного курсового проекту є здобуття практичних навичок у правильному використанні патернів проектування.

Завданням до курсового проекту є створення закінченого програмного продукту з використанням патернів проектування програмного забезпечення.

Об’єкт дослідження – закінчений програмний продукт, який буде отримано у результаті написання даного курсового проекту.

1. Теоретичні відомості.

Паттерн представляє певний спосіб проектування для вирішення часто зустрічаючихся проблем проектирования. Патерни вирішують певний список формалізованих проблем, які часто зустрічаються, і патерни представляють декілька принципів для вирішення цих проблем.

Патерни можуть бути різними за характером поведінки, основні види описані нижче.

Породжуючі патерни – патерни, які абстрагують процес створення об'єктів класів.

Структурні патерни – описують як класи і об'єкти створюють крупніші структури.

Поведінкові – визначають взаємодію між різними класами і об'єктами.

Патерни об'єктів – описують відношення між об'єктами.

Патерни дозволяють вирішувати ті проблеми, які зустрічаються найчастіше. І дані рішення являються загальноприйнятими, що дозволяє різним розробникам з легкістю розуміти програмні рішення один одного.

1. Технічне завдання.

Завдання до даного курсового проекту це створення закінченого програмного продукту з використанням як мінімум трьох патернів проектування.

Даний курсовий проект – це веб сервіс, аналог банківського додатку з можливістю псевдо введення, виведення, та переведення коштів іншим користувачам додатку.

Програмний продукт повинен бути закінченим і відповідати усім поставленим до нього вимогам.

1. Опис програмного продукту.

Програмний продукт представляє собою веб-сайт аналог банківського додатку.

Функціонал програмного продукту:

Реєстрація.

Авторизація.

Можливість депозиту коштів.

Можливість виведення коштів.

Можливість переведення коштів з одного аккаунту на інший.

Програмний продукт побудований з використанням таких технологій:

ASP .NET Core (Web API).

Angular 5.

Entity Framework Core.

1. Керівництво користувачу.

Розроблений програмний продукт – це веб сервіс, що являє собою аналог банківського додатку.

При переході на сторінку додатку ми побачимо вікно яке призначається для авторизації користувачів у системі.

На цьому вікні ми можемо бачити:

1. Поле для введення логіну.
2. Поле для введення паролю.
3. Кнопка авторизації.
4. Кнопка переходу на сторінку реєстрації.
5. Поле навігації.

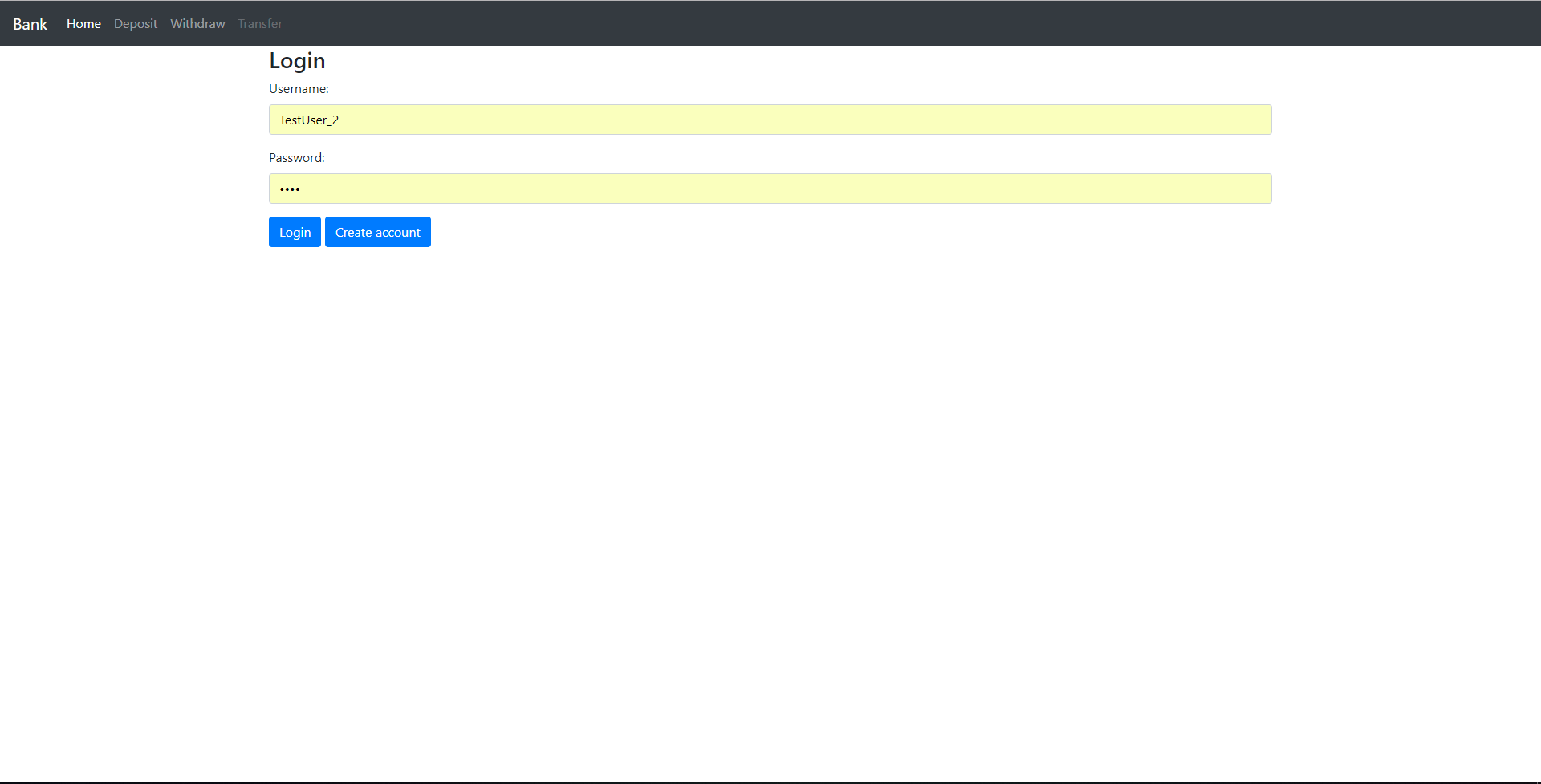


Рис. 4.1. – Вікно авторизації.

Якщо хоч 1 з двох полів для введення не буде заповнено, то кнопка авторизації буде неактивною.

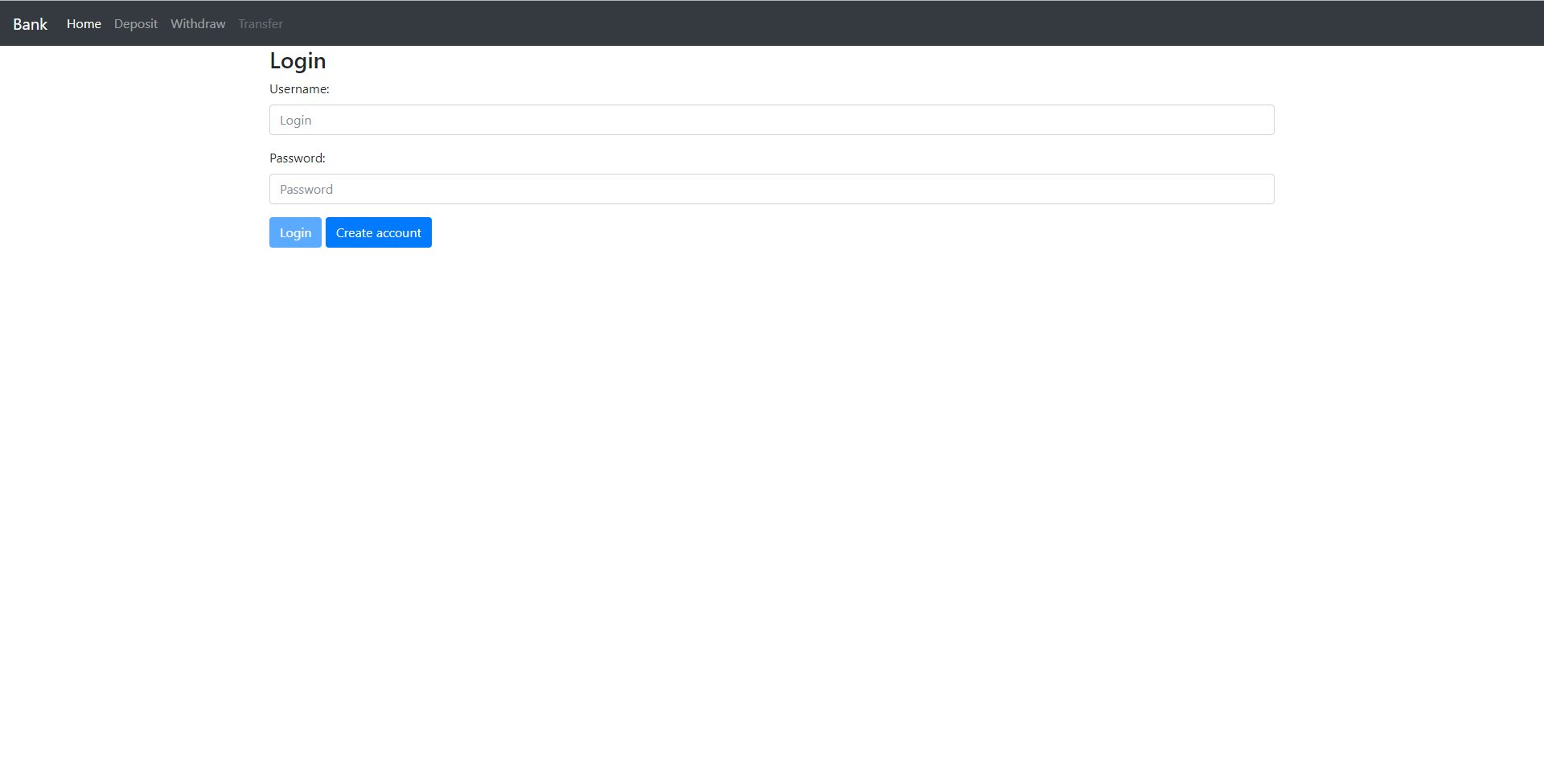


Рис. 4.2. – Вікно авторизації з не заповненими полями для введення.

Варто зазначити, що при натисканні на будь-який пункт поля навігації, у випадку якщо користувач не авторизований, нічого не буде показано, користувач буде перенаправлений на вікно авторизації.

Якщо необхідно зареєструватися, то необхідно на сторінці авторизації натиснути на кнопку переходу до сторінки реєстрації, там ми можемо побачити такі елементи як:

1. Поля для введення інформації необхідної для реєстрації.
2. Кнопка реєстрації.
3. Кнопка переходу на вікно авторизації.
4. Поле навігації.

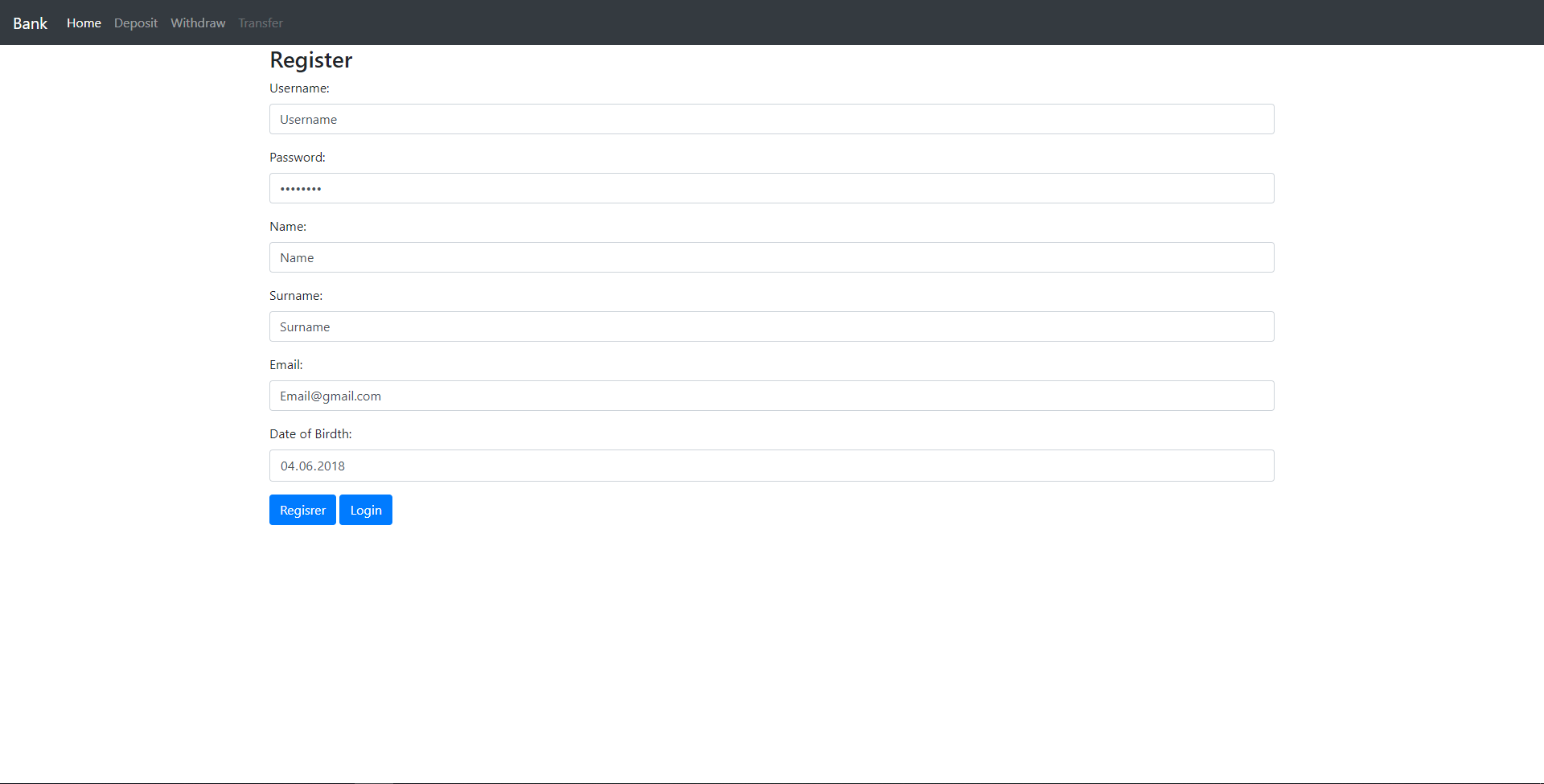


Рис. 4.3. – Вікно реєстрації.

Якщо не ввести дані у якусь з форм для введення, то вона буде підсвічена відповідно червоним кольором, та буде показано повідомлення про помилку і кнопка що відповідає за реєстрацію буде неактивною.

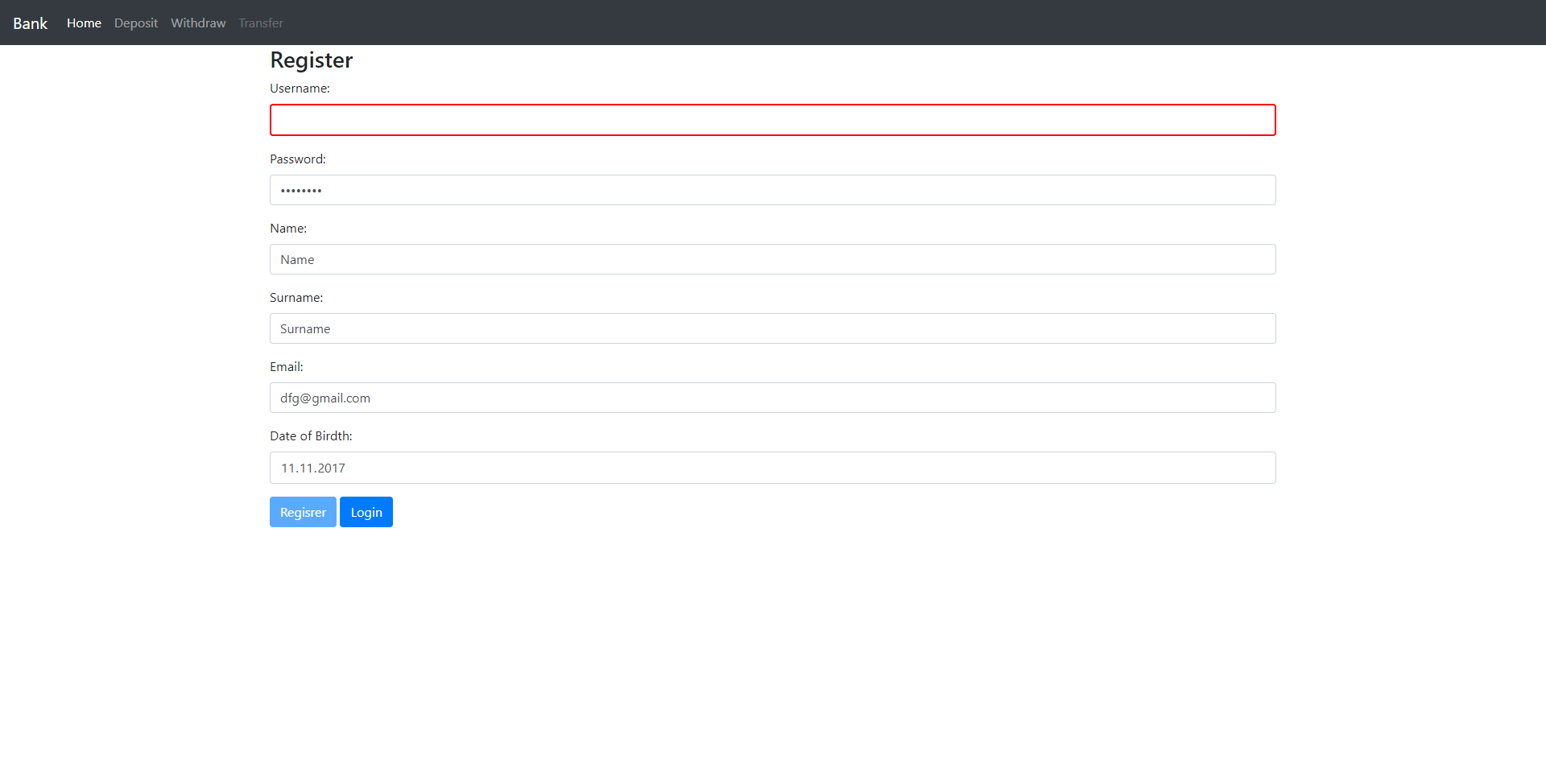


Рис. 4.4. – Вікно реєстрації з незаповненими полями для введення.

Якщо у поля будуть введені данні, які не відповідають форматові цих полів, то

відповідно кнопка для підтвердження реєстрації буде неактивною.

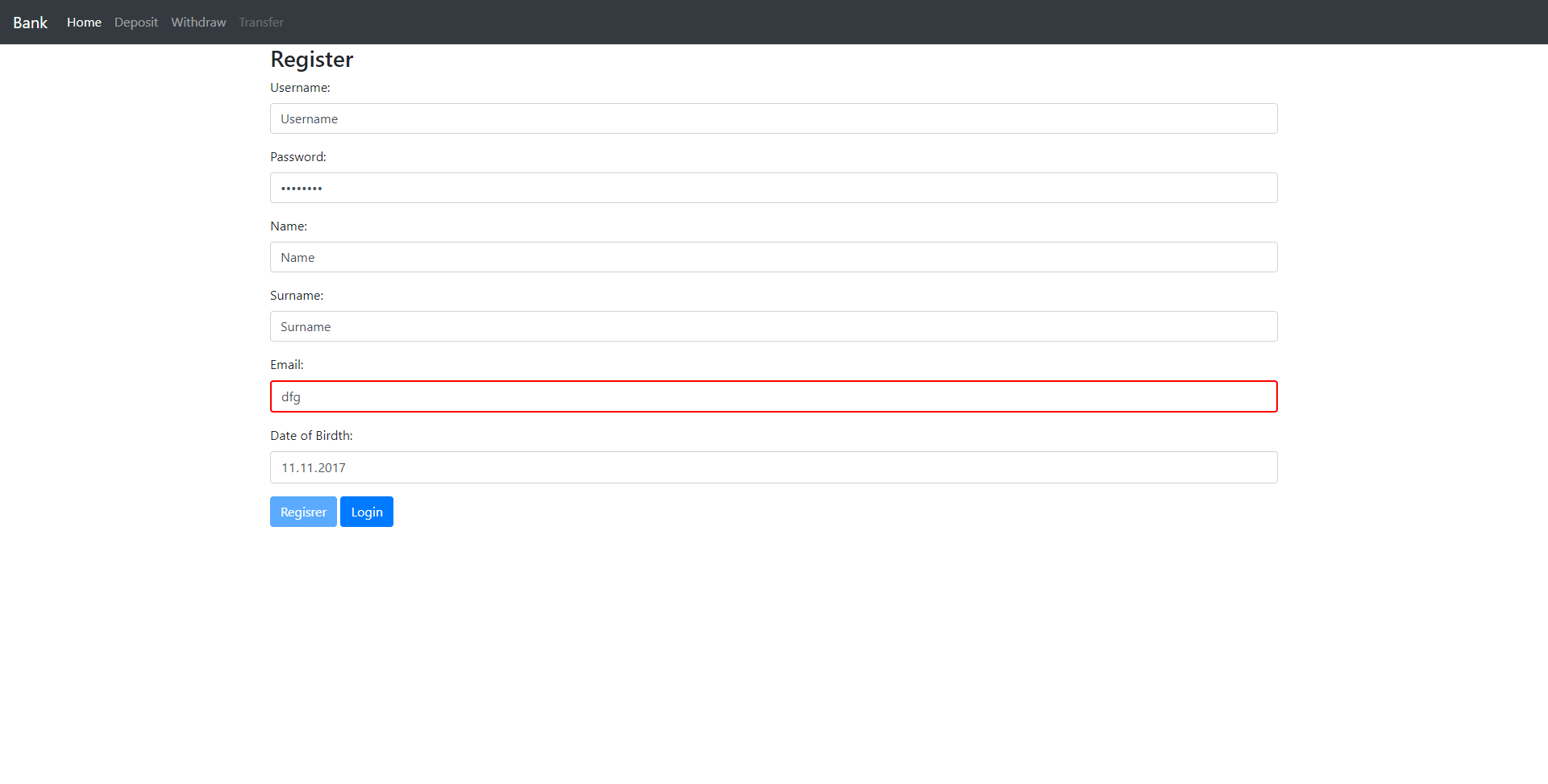


Рис. 4.5. – Вікно реєстрації неправильно заповненими полями для введення.

Якщо усі данні введені коректно, то кнопка, що відповідає за реєстрацію користувачів буде активною і користувач буде мати змогу зареєструватися. Після того, як буде натиснуто на кнопку реєстрації, користувач буде автоматично перенаправлений на сторінку авторизації. І відповідно буде мати змогу авторизуватися у системі.

Якщо користувач успішно авторизувався, то він має змогу у повному обсязі користуватися можливостями веб-додатку.

На усіх сторінках додатку є поле з навігацією. Яке у свою чергу складається з :

1. Посилання, назва додатку, натиснувши на яку користувач буде переміщений на головну сторінку.
2. Посилання переміщення на головну сторінку.
3. Посилання на сторінку депозиту.
4. Посилання на сторінку виведення коштів.
5. Посилання на сторінку переведення коштів.
6. Кнопку виходу з системи.



Рис. 4.6. – Блок навігації.

Головне вікно додатку складається з таких елементів:

1. Назва сторінки.
2. Поле у якому показано поточний баланс.
3. Вікно у якому відображаються транзакції.
4. Блок навігації.

Відповідно транзакції можуть бути:

1. Депозит.
2. Виведення коштів.
3. Переведення коштів від поточного користувача до іншого.

Переведення коштів від будь-якого іншого користувача до поточного.

Сама секція окремої транзакції складається з:

1. Назви транзакції.

Суму транзакції.

Дату транзакції.

Кольоровий індикатор який показує чи збільшувався чи зменшувався баланс. Червоний якщо баланс зменшувався, та зелений якщо збільшувався.

І у випадках коли це переведення коштів:

Користувач ініціатор.

Користувач отримувач

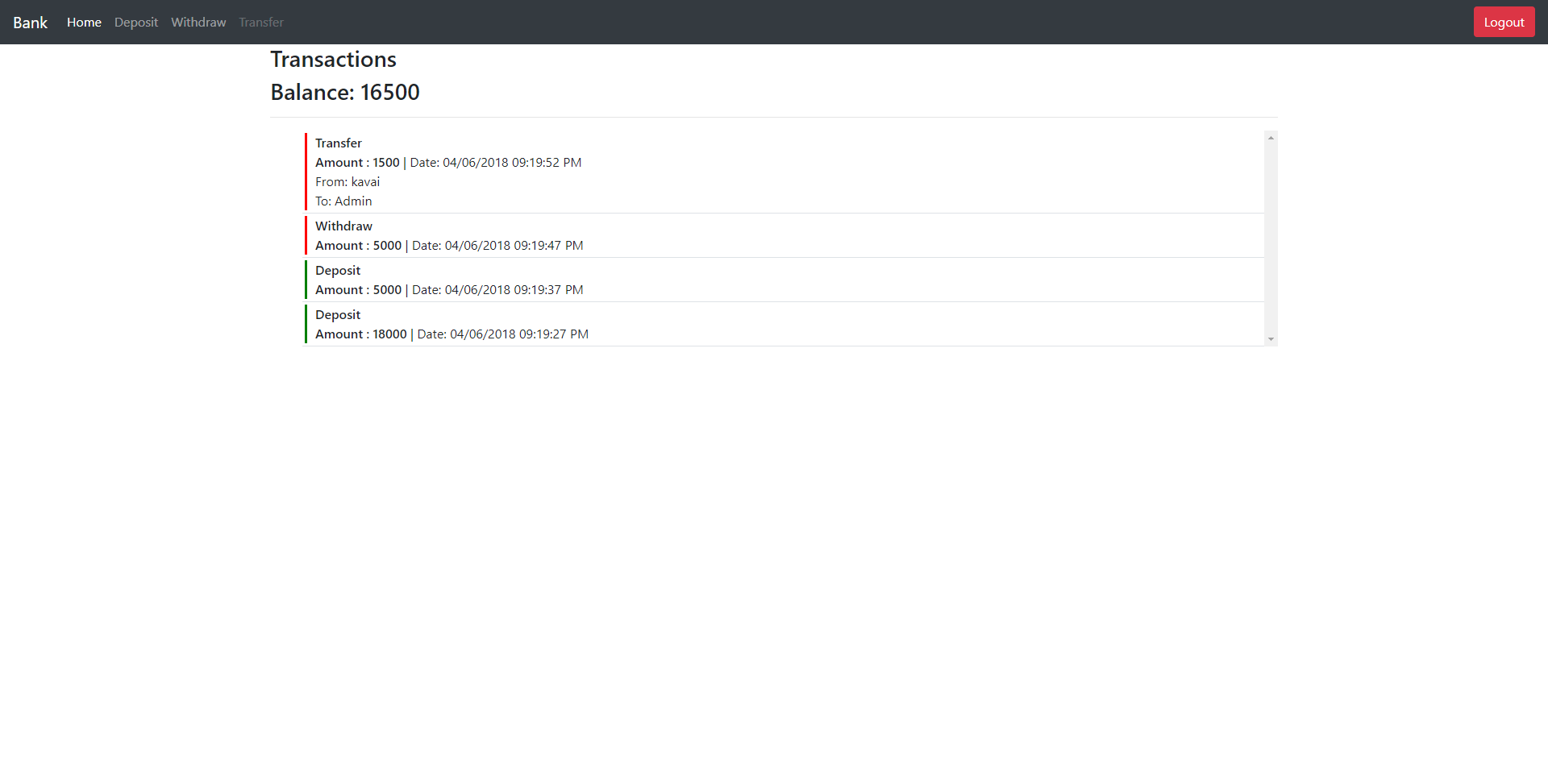


Рис. 4.7. – Вікно транзакцій.

Вікно депозиту складається з:

1. Назви сторінки.
2. Поточного балансу.
3. Вікна для введення суми депозиту.
4. Кнопка поповнення.

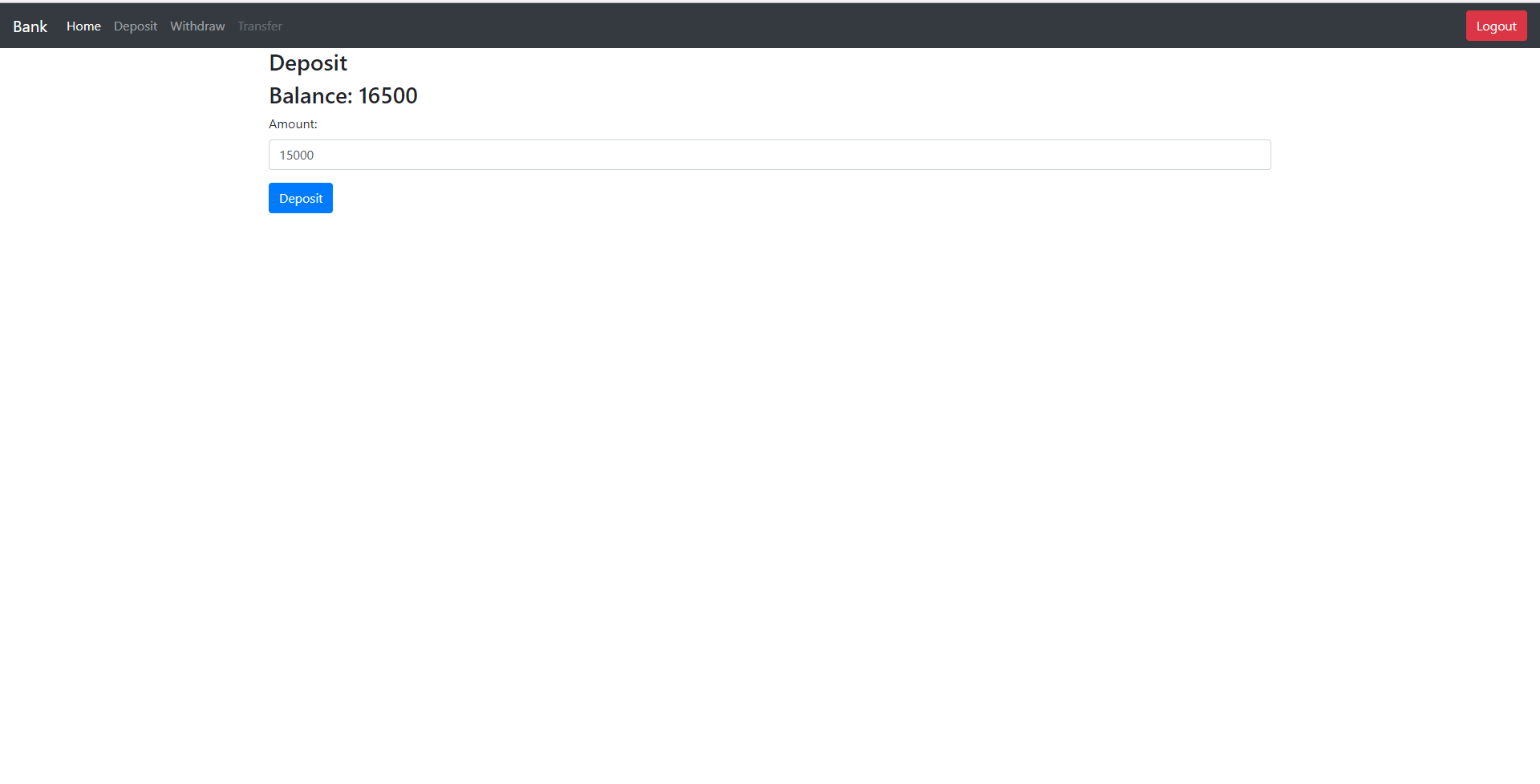


Рис. 4.8. – Вікно депозиту.

Якщо не ввести суму для поповнення, або ввести від’ємну суму то кнопка депозиту буде неактивною.

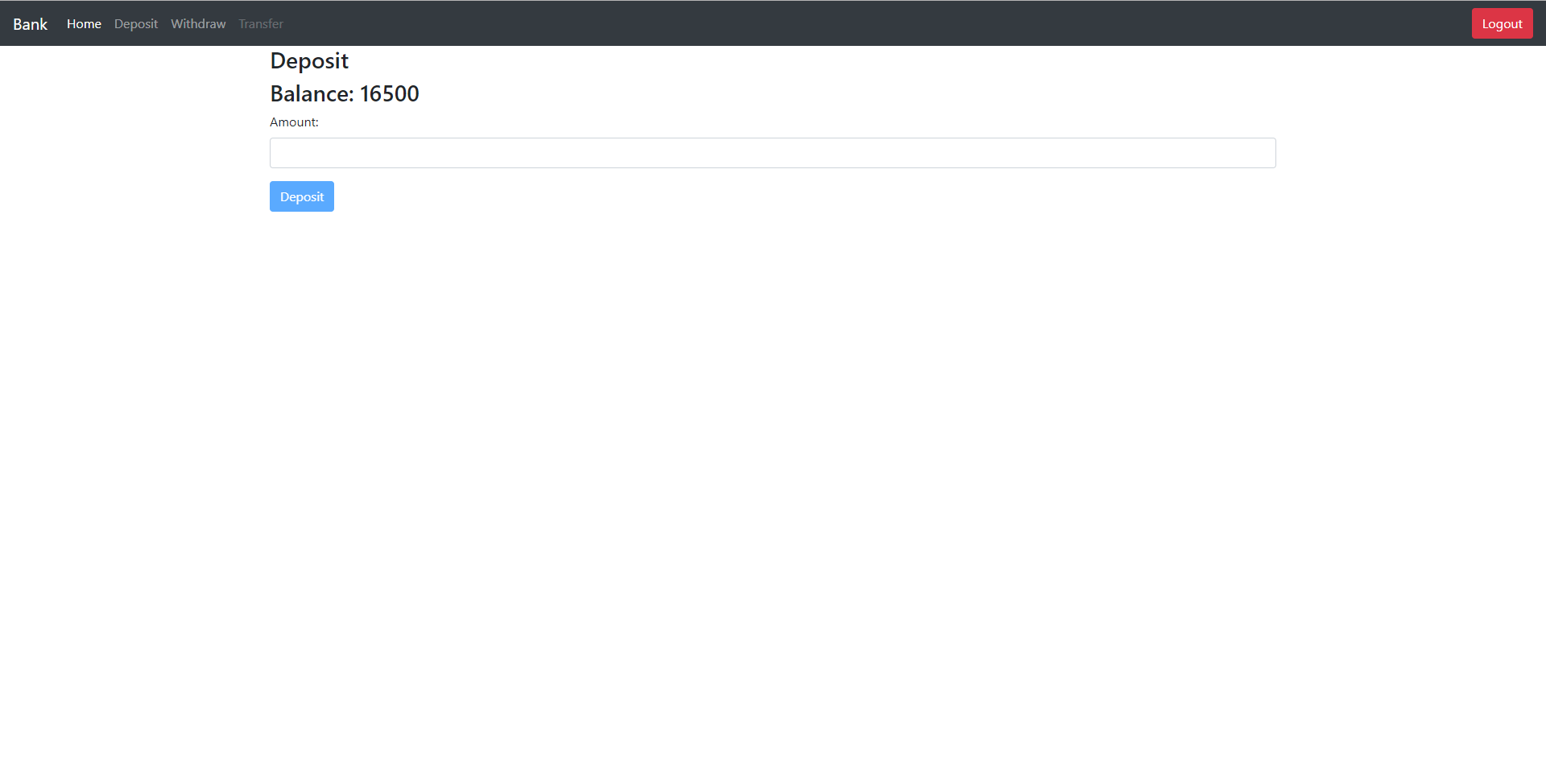


Рис. 4.9. – Вікно депозиту з не введеною сумою.

Вікно виведення коштів складається з:

1. Назви сторінки.
2. Поточного балансу.
3. Вікна для введення суми виведення коштів.
4. Кнопка виведення коштів.

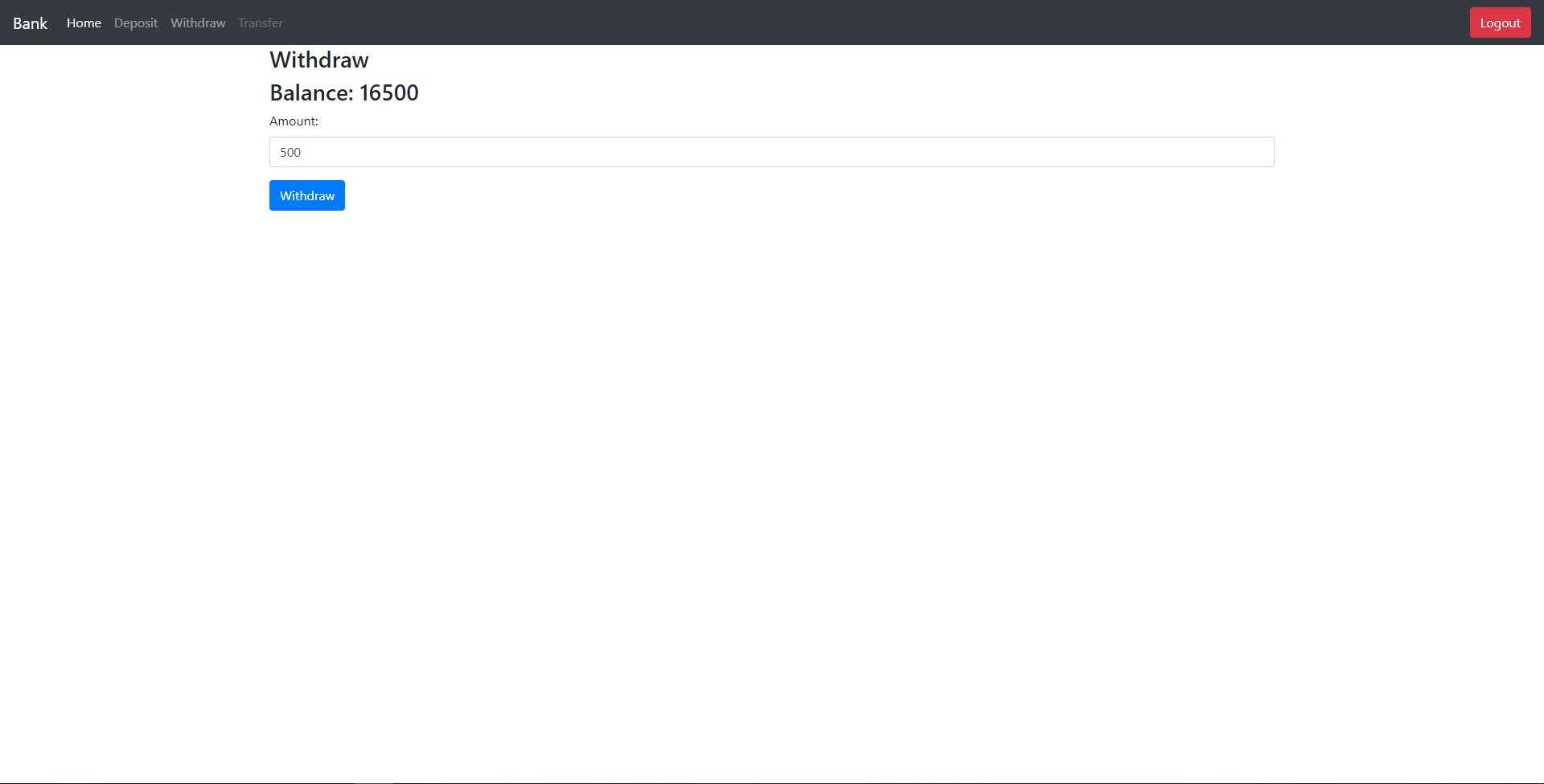


Рис. 4.10. – Вікно виведення коштів.

Якщо не ввести суму для поповнення, або ввести від’ємну суму то кнопка депозиту буде неактивною.

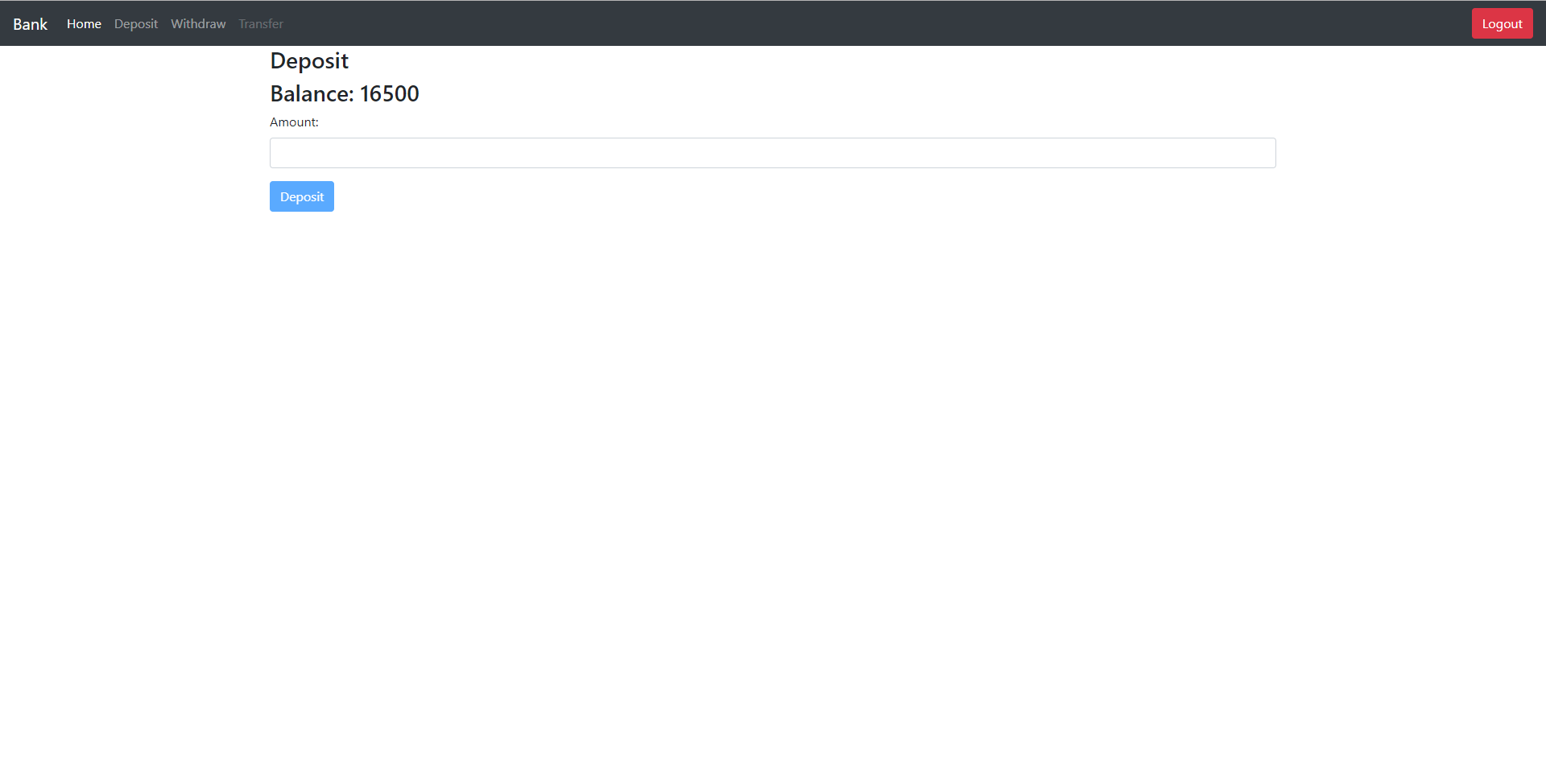


Рис. 4.11. – Вікно виведення коштів з не введеною сумою.

Вікно переведення коштів складається з:

1. Назви сторінки.
2. Поточного балансу.
3. Вікна для введення суми переведення коштів.
4. Поле для вибору якому користувачеві перевести кошти.
5. Кнопка переведення коштів.

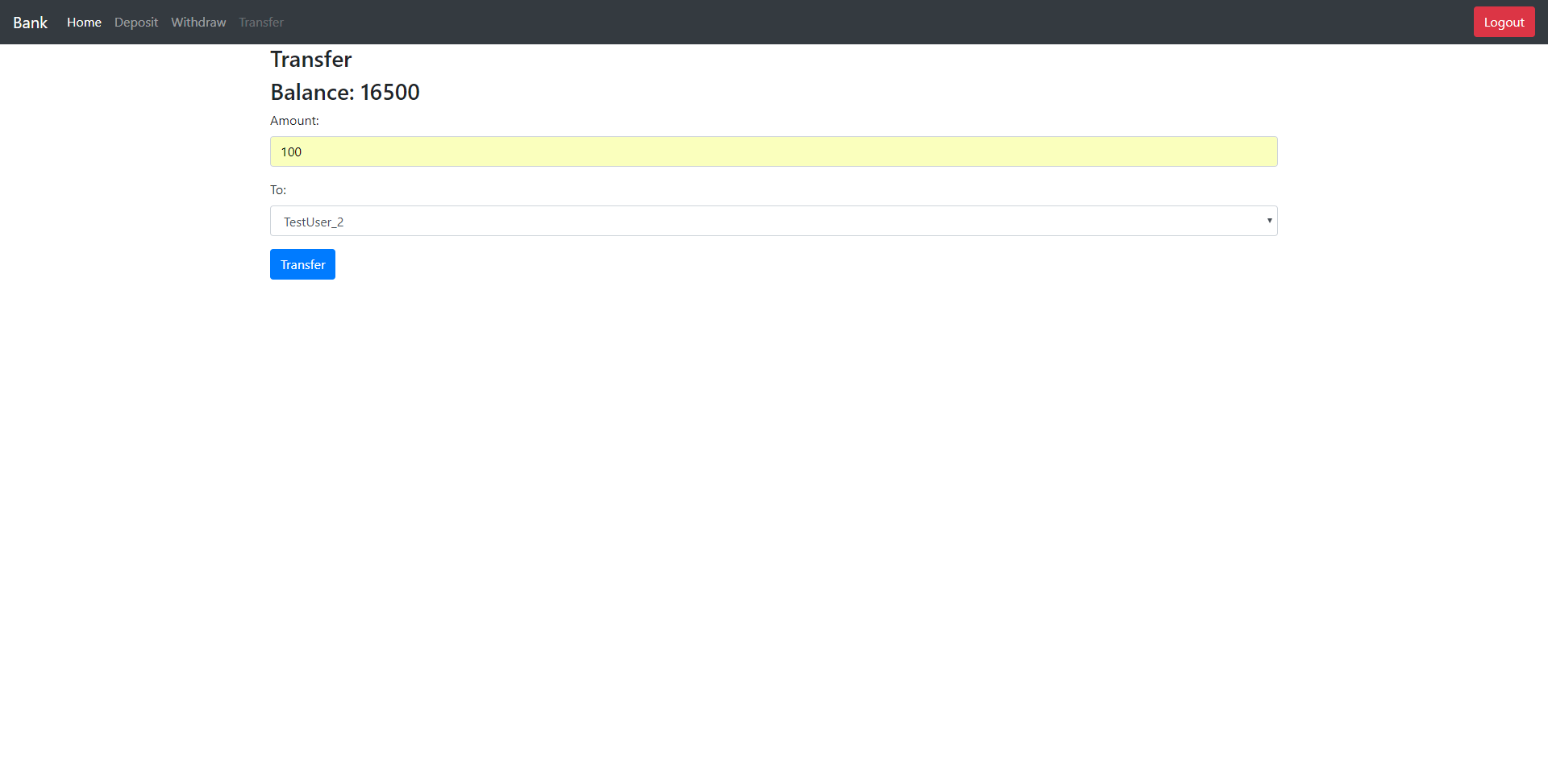


Рис. 4.12. – Вікно переведення коштів.

Якщо не ввести суму для поповнення, або ввести від’ємну суму, або не обрати користувача отримувача, то кнопка депозиту буде неактивною.

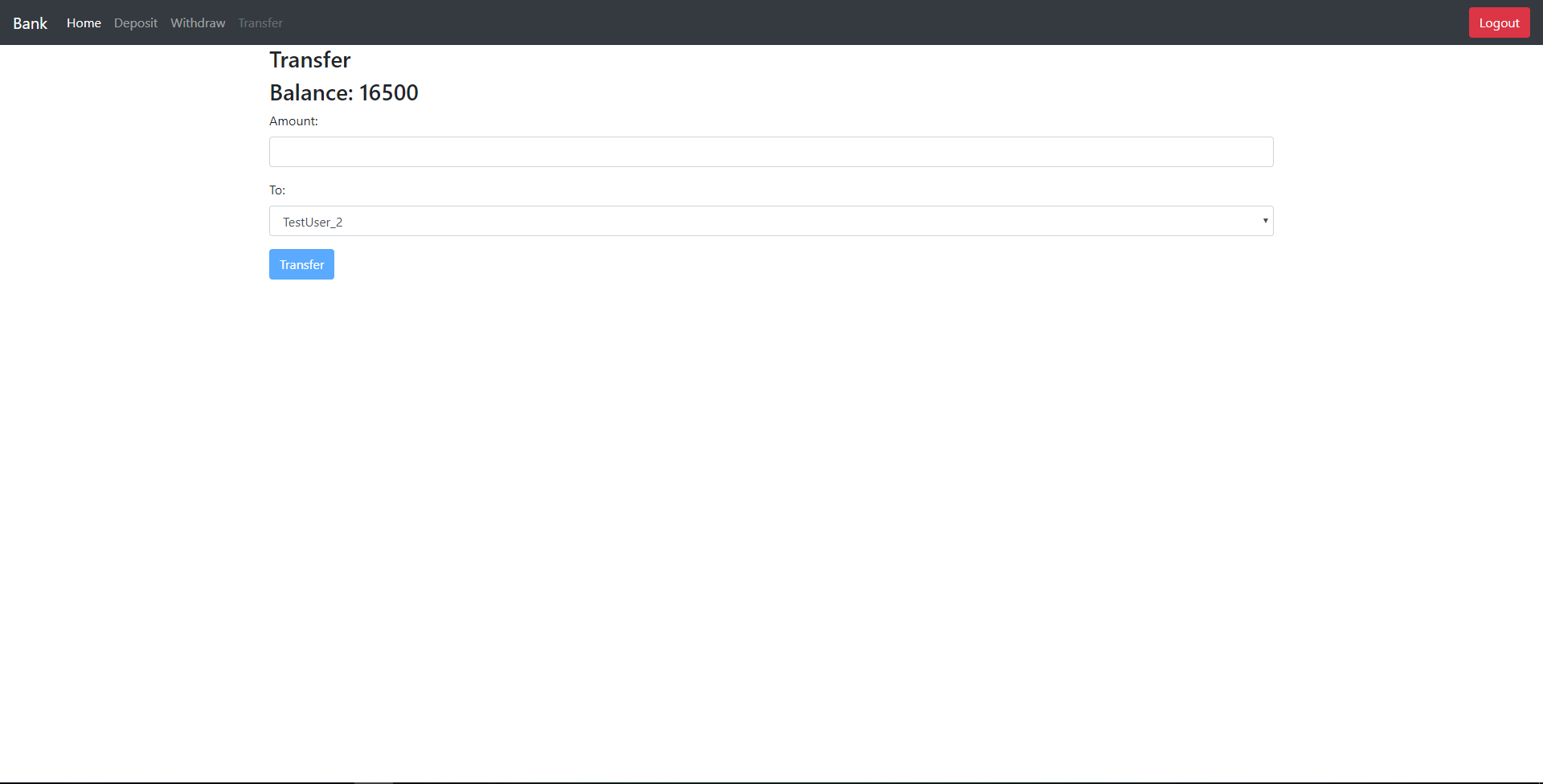


Рис. 4.13. – Вікно переведення коштів з не введеною сумою.

1. Керівництво програмісту.

Веб додаток створений на базі технологій:

1. Серверна частина: ASP .NET Core (Web API).
2. Користувацька частина: Angular 5.

Перевагою даного підходу є те, що серверна частина та користувацька частина максимально розділені і по суті являються окремими додатками. І серверну частину можна використовувати не лише з цією користувацькою частиною, а з іншими користувацькими інтерфейсами, які не обов’язково повинні бути веб додатками.

Приклад Web Api контролера:

namespace BA.Web.Controllers

{

[Produces("application/json")]

[Route("api/Users")]

public class UsersController : Controller

{

private IUserServises \_userServises;

public UsersController(IUserServises UserServises)

{

\_userServises = UserServises;

}

[HttpGet]

[Authorize]

public IEnumerable<UserView> Get()

{

var model = \_userServises.GetListForTransactions(User.Identity.Name);

if (model.Error)

{

Response.StatusCode = 400;

}

return model.Items;

}

[HttpPost]

[Route("Register")]

public IActionResult Post([FromBody]UserModel user)

{

var model = \_userServises.Register(user);

if (!model.Error)

return Ok(model.Error);

return BadRequest(model.ErrorMessage);

}

[Authorize]

[Route("GetCurrentUser")]

public UserView GetCurrentUser()

{

var model = \_userServises.GetUserViewModel(User.Identity.Name);

if (model.Error)

{

Response.StatusCode = 400;

}

return model.Item;

}

[Authorize]

[Route("Transactions")]

public IEnumerable<TransactionView> GetTransactionList()

{

var model = \_userServises.GetTransactionList(User.Identity.Name);

if (model.Error)

{

Response.StatusCode = 400;

}

return model.Items;

}

}

}

[Produces("application/json")]

[Route("api/Transaction")]

public class TransactionController : Controller

{

private ITransactionServisesRepositoryes \_transactionServises;

public TransactionController(ITransactionServisesRepositoryes transactionServises)

{

\_transactionServises = transactionServises;

}

[Authorize]

[Route("Deposit")]

public IActionResult Deposit(double amount)

{

var result = \_transactionServises.Deposit(User.Identity.Name, amount);

if (!result.Error)

return Ok(!result.Error);

return BadRequest(result.ErrorMessage);

}

[Authorize]

[Route("Withdraw")]

public IActionResult Withdraw(double amount)

{

var result = \_transactionServises.Withdraw(User.Identity.Name, amount);

if (!result.Error)

return Ok(!result.Error);

return BadRequest(result.ErrorMessage);

}

[Authorize]

[Route("Transfer")]

public IActionResult Transfer(double amount, string userReceiverName)

{

var result = \_transactionServises.Transfer(amount, User.Identity.Name, userReceiverName);

if (!result.Error)

return Ok(!result.Error);

return BadRequest(result.ErrorMessage);

}

}

Доступ до певних методів можливий лише за використання відповідних типів запитів.

Для доступу до бази даних використовується Entity Framework Core та патерну Unity of Work.

Код класу:

public class UnitOfWork : IUnitOfWork

{

private DataContext.DataContext db;

private IRepositories<User> UserRepository;

private IRepositories<Account> AccounRepository;

private ITransactionRepositories<Transaction> TransactionRepository;

public UnitOfWork(DataContext.DataContext \_context)

{

db = \_context;

}

public IRepositories<User> Users

{

get

{

if (UserRepository == null)

UserRepository = new UserRepositories(db);

return UserRepository;

}

}

public IRepositories<Account> Accounts

{

get

{

if (AccounRepository == null)

AccounRepository = new AccountRepositories(db);

return AccounRepository;

}

}

public ITransactionRepositories<Transaction> Transaction

{

get

{

if (TransactionRepository == null)

TransactionRepository = new TransactionRepositories(db);

return TransactionRepository;

}

}

public void Save()

{

db.SaveChanges();

}

private bool disposed = false;

public virtual void Dispose(bool disposing)

{

if (!this.disposed)

{

if (disposing)

{

db.Dispose();

}

this.disposed = true;

}

}

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

}

public class DataContext:DbContext

{

private static DataContext \_instance;

private DataContext()

{ }

public static DataContext getInstance()

{

if (\_instance == null)

\_instance = new DataContext();

return \_instance;

}

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

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

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

public DataContext(DbContextOptions options):base(options)

{

Database.EnsureCreated();

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

}

}

Прилад репозиторію:

public class TransactionRepositories : ITransactionRepositories<Enteties.Transaction>

{

private DataContext.DataContext db;

public TransactionRepositories(DataContext.DataContext \_context)

{

db = \_context;

}

public IEnumerable<Enteties.Transaction> GetList(string Username)

{

var TransactionsList = db.Transactions

.Include( b=>b.AccountInitiator)

.ThenInclude(c => c.User)

.Include( b=>b.AccountRecipient)

.ThenInclude(c => c.User)

.Where(c=>

(c.AccountInitiator.User.UserName == Username) ||

(c.AccountRecipient.User.UserName == Username));

return TransactionsList;

}

public Enteties.Transaction Get(string Username)

{

var ListTransactions = db.Transactions

.Include(b => b.AccountInitiator)

.ThenInclude(c => c.User)

.Include(b => b.AccountRecipient)

.ThenInclude(c => c.User)

.ToList().OrderBy(x=> x.Date).FirstOrDefault();

return ListTransactions;

}

public IEnumerable<Enteties.Transaction> GetList()

{

var ListTransactions = db.Transactions

.Include(b => b.AccountInitiator)

.ThenInclude(c => c.User)

.Include(b => b.AccountRecipient)

.ThenInclude(c => c.User)

.ToList();

return ListTransactions;

}

public void Add(Enteties.Transaction transaction)

{

db.Transactions.Add(transaction);

}

}

namespace BA.Database.Repositories

{

public class AccountRepositories: IRepositories<Account>

{

private DataContext.DataContext db;

public AccountRepositories(DataContext.DataContext \_context)

{

db = \_context;

}

public IEnumerable<Account> GetList()

{

var List = db.Accounts

.Include(b => b.Initiator)

.Include(b => b.Recipient)

.Include(b => b.User)

.ToList<Account>();

return List;

}

public Account Get(string UserName)

{

var Account = db.Accounts

.Include(b => b.Initiator)

.Include(b => b.Recipient)

.Include(b => b.User)

.Where(c => c.User.UserName == UserName).SingleOrDefault();

return Account;

}

public void Add(Account Account)

{

db.Accounts.Add(Account);

}

}

}

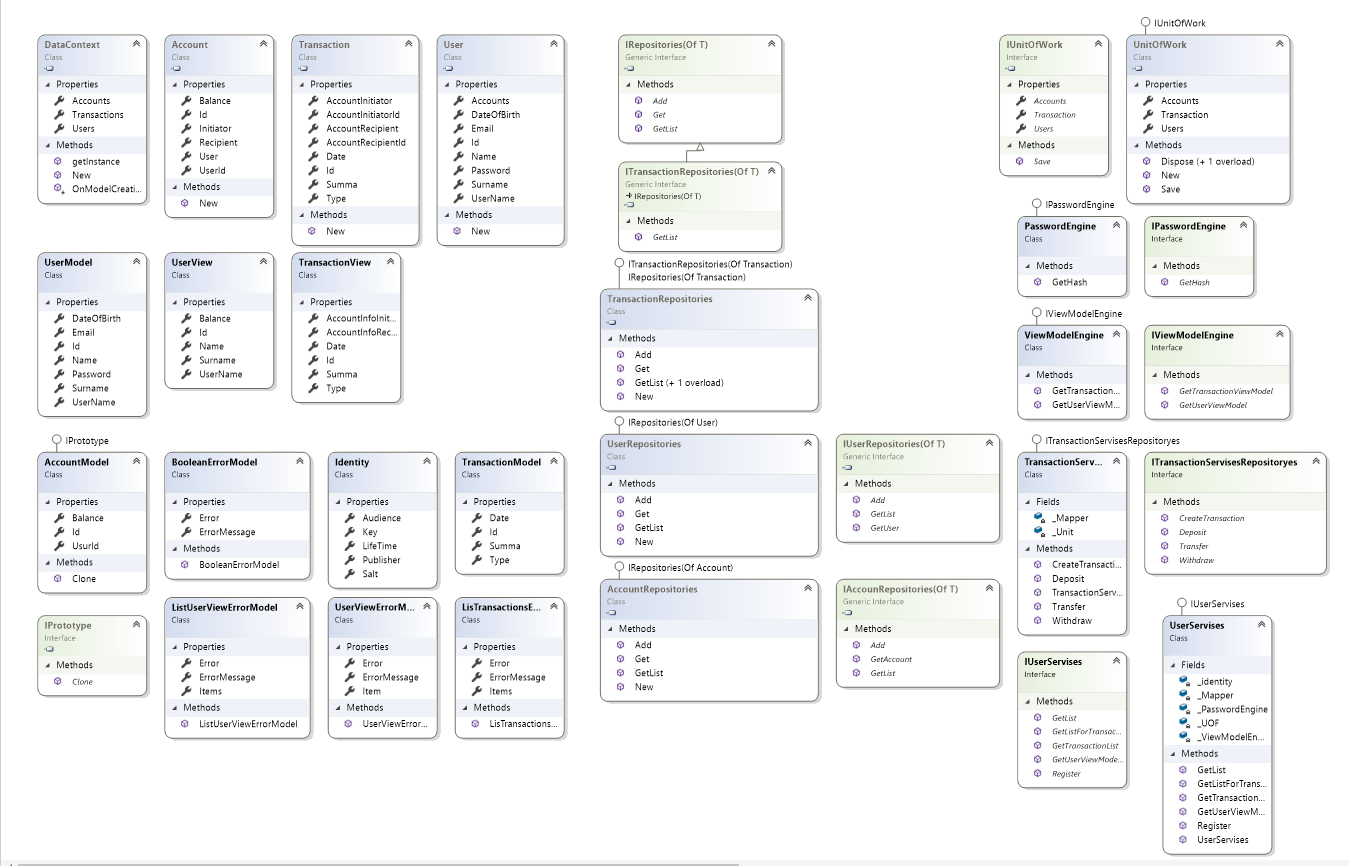


Рис. 5.1 – Діаграма програмного продукту.

У даному курсовому проекті реалізовано партерни проектуванн, основні з них:

1. Prototype.
2. Singleton.
3. Strategy.

Prototype – це патерн проектування, що дозволяє створити об’єкти на основі вже створених, по суті надаючи можливість клонування об’єктів.

Даний патерн реалізований у класі AccountModel.

Код класу:

public class AccountModel : IPrototype

{

public int Id { get; set; }

public int UsurId { get; set; }

public double Balance { get; set; }

public IPrototype Clone()

{

return new AccountModel()

{

Id = Id,

UsurId = UsurId,

Balance = Balance

};

}

}

public interface IPrototype

{

IPrototype Clone();

}

Singleton – патерн, що гарантує що для певного класу буде створено лише 1 об’єкт.

У даному курсовому проекті патерн Singleton реалізовує клас DataContext.

public class DataContext:DbContext

{

private static DataContext \_instance;

private DataContext()

{ }

public static DataContext getInstance()

{

if (\_instance == null)

\_instance = new DataContext();

return \_instance;

}

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

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

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

public DataContext(DbContextOptions options):base(options)

{

Database.EnsureCreated();

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

}

}

Також даний патерн можна реалізувати використовуючи вбудований у .Net Core механізм Dependency Injection.

services.AddSingleton<IMapper>(sp => \_mapperConfiguration.CreateMapper());

Strategy – патерн, який дозволяє легко замінити один виконуваний алгоритм іншим.

Даний патерн реалізовують класи: UnitOfWork, TransactionServises, UserServises, ViewModelEngine, PasswordEngine.

Реалізація патерну здійснюється за допомогою вбудованого механізму Dependency Injection у .NET Core.

Встановлення відповідностей здійснюється у методі ConfigureServices класу Startup.

services.AddSingleton<IMapper>(sp => \_mapperConfiguration.CreateMapper());

services.AddTransient<IUnitOfWork, UnitOfWork>();

services.AddTransient<ITransactionServisesRepositoryes, TransactionServises>();

services.AddTransient<IUserServises, UserServises>();

services.AddTransient<IViewModelEngine, ViewModelEngine>();

services.AddTransient<IPasswordEngine, PasswordEngine>();

І отримання відповідної реалізації здійснюється у конструкторах класів, що будуть використовувати дані класи.

public TransactionServises(IUnitOfWork Unit, IMapper mapper)

{

\_Mapper = mapper;

\_Unit = Unit;

}

ВИСНОВКИ

Працюючи над виконання завдання до курсового проекту я здобув практичні уміння та навички з розробки складних програмних продуктів, навчився об’єктивно оцінювати складність поставлених перед мною завдань, та виконувати поставлені завдання у поставлений термін.

Виконання даного програмного продукту допомогло мені покращити уміння та навички у мові програмування c#, ASP.NET Core, та Angular 5.

Також я здобув уміння та навички використання патернів проектування програмного забезпечення на практиці.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] ADO.NET Entity Framework [Електронний ресурс] – Доступ за посиланням: <https://ru.wikipedia.org/wiki/ADO.NET_Entity_Framework> [Дата звернення 04.06.2018.]

[2] Language Integrated Query [Електронний ресурс] – Режим доступу до ресурсу: <https://ru.wikipedia.org/wiki/Language_Integrated_Query> Дата звернення 04.06.2018.]

[3] Nick H. ASP.NET MVC Security / Harrison Nick. – Morrisville, 2015. – 85 с.

[4] Object-relational mapping [Електронний ресурс] – Режим доступу до ресурсу: <https://en.wikipedia.org/wiki/Object-relational_mapping> [Дата звернення 04.06.2018.]

[5] Руководство по Entity Framework [Електронний ресурс] – Режим доступу до ресурсу: <https://metanit.com/sharp/entityframework/> [Дата звернення 04.06.2018.]

[6] Паттерны проектирования в C# и .NET [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/patterns/ [Дата звернення 04.06.2018.]