МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Розробка інформаційного порталу "Спортивний бридж" за допомогою Javalin та PostgreSQL»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Шаповалов С.П.**

Студента групи ІН – 63 — **Лопатка К.Р.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2020 г.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-63 спеціальності "Інформатика" денної форми навчання Лопатки Кирила Романовича.

Тема: "**Розробка інформаційного порталу "Спортивний бридж" за допомогою Javalin та PostgreSQL** "

Затверджена наказом по СумДУ

№_____ от _____2020 г.

Зміст пояснювальної записки: 1) аналітичний огляд веб-ресурсів; 2) аналіз мов програмування, субд, фреймворку 3) проектування субд, проектування інтерфейсу 5)розробка інформаційного порталу; 6) аналіз отриманих результатів.

Дата видачі завдання "_____"_____2020 г.

Керівник випускної роботи _____ Шаповалов С.П.

Завдання приняв до виконання _____ Лопатка К.Р.

РЕФЕРАТ

Записка: 79 сторінки, 24 рис., 2 таблиці, 15 джерел, 2 додатки.

Об’єкт дослідження — вузько-орієнтований веб-ресурс

Мета роботи — розробка інформаційного порталу спортивний брідж

Методи дослідження — інформації аналіз, моделювання СУБД

Результати — проведений аналіз літератури, спроектована бд, яка була реалізована на PostgreSQL и реалізований інформаційний портал за допомогою Java, фреймворку Javalin, html, css, js.

ВЕБ-РЕСУРС, СПОРТИВНИЙ БРИДЖ, КЛУБ, JAVA, JAVALIN, POSTGRESQL, ТЕСТУВАННЯ

ЗМІСТ

ВСТУП

На сьогоднішній день багато людей грають в різноманітні ігри, щоб провести свій вільний час. Є дуже багато різноманітних ігор, які зосереджені на силовій активності: футбол, волейбол і баскетбол. А є ігри, які зосереджені на інтелектуальній активності мозку: шахи, де потрібно наперед обдумувати свої кроки, хтось вирішує сканворди, щоб перевірити свої знання та ерудицію або розв’язують судоку на швидкість, щоб перевірити свою уважність та швидкість прийняття рішень за певний час.

Є ігри, для яких недостатньо знати тільки правила, але ще й потрібно знати деякі особливості, яка вона приховує в собі.

Однією з таких ігр є спортивний брідж. Саме в цій грі дуже великий поріг входження. Тобто, щоб в ній бути дуже успішним та отримувати задоволення від гри, ти повинен знати багато інформації. В ній заборонено розмовляти під час гри, і щоб поділитися інформацію зі своїм партнером використовують систему торгівлі. Або щоб попередити свого напарника під час гри, що в тебе закінчується карти певної масті. Всю інформацію, яку потрібно тобі буде вивчити, щоб отримувати задоволення від гри можно будет знайти на різних веб-сайтах, тобто не існує такого інформаційного порталу, де було би зібрана вся інформація або вона є, але там настільки мало інформації або взагалі вона подана так, що у тебе з’явиться ще більше питань і в тебе відпаде бажання почати грати всю цікаву, неймовірну гру.

Мета роботи - розробка інформаційного порталу “Спортивний брідж” на якому буду зібрана вся потрібна інформація у вигляді постів, яку потребують нові гравці, щоб вони змогли розібратися з цією складною грою. Крім того, на веб-сайті можно буде створювати пости в яких користувач може ділиться

своїми ситуаціями, які з ним сталися під час звичайних ігор або турнірів. Або які новини з’явилися в світі спортивного бріджу.

Завданнями роботи є: розробити інформаційного портал на якому буде зібрана вся потрібна інформація, користувачі можуть реєструватися, створювати пости, коментувати їх.

1 ОГЛЯД ВІДОМИХ РІШЕНЬ

Існують деякі інформаційні портали які схожі між собою, мають схожий функціонал та тематику, тому вони були обрані на дослідження, щоб підкреслити кращі сторони і використати та реалізувати в своїй роботі.

1.1 Аналіз існуючих інформаційних порталів

1.1.1 Bridgebum

Інтернет ресурс, на якому зібрано багато інформації. Він дуже популярний серед бриджістів. На ньому можно знайти будь-яку інформацію, яку потрібно. При заходжені на сайті подається інформація з останніми оновленими постами. В шапці веб-ресурсу розташовані найважливіший функціонал для користування. На веб-сайті існує функція пошуку, яка спрощує користуванням сайтом. Мапа сайту побудована так, що людина, яка ніколи в житті не користувалося інтернетом зможе з легкістю використовувати його в своїх цілях.[5]

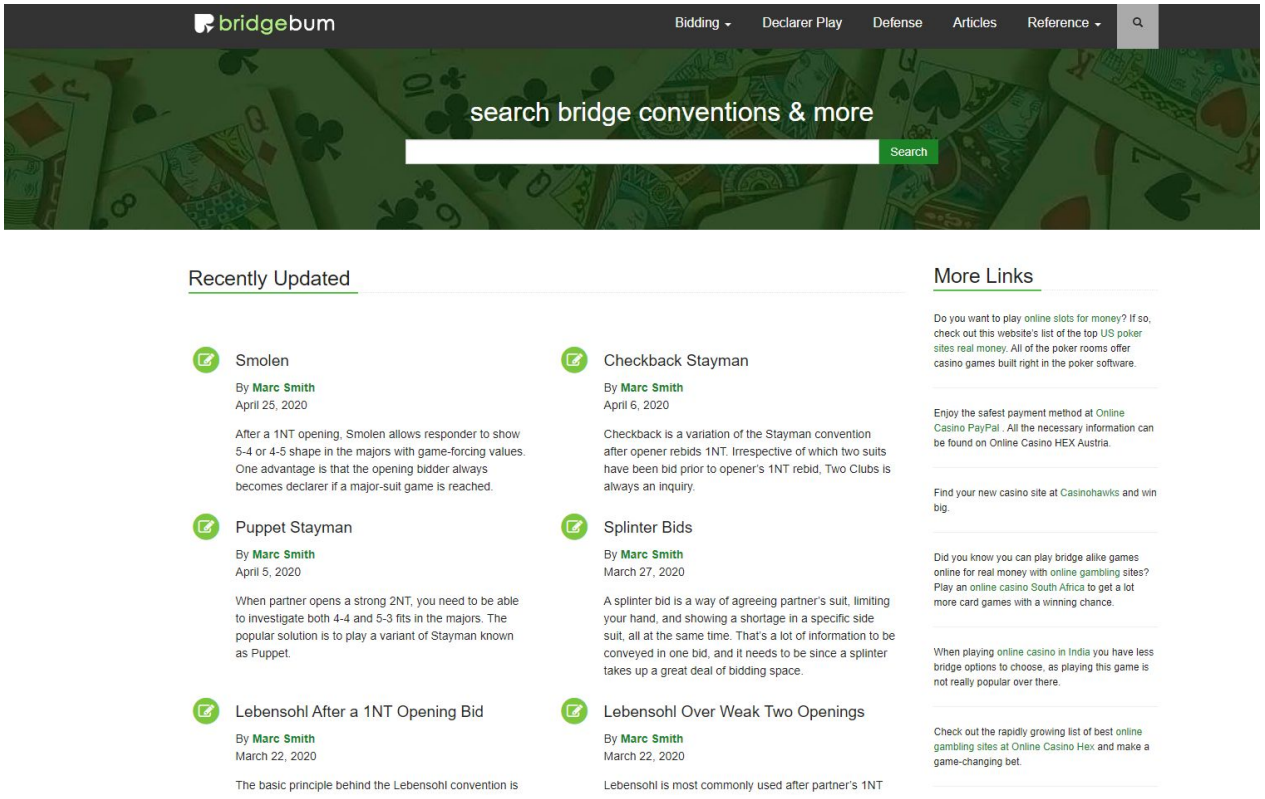

Рисунок 1.1.1 – Зовнішний вигляд веб-сайту Bridgebum

1.1.2 Bridgeclub

На стартовій сторінці розташовані три основних функціоналу сайту: мапа новин, матеріали, які допоможуть ознайомитися з усією зібраною інформацією та мапа сайту. На веб-порталі існує перехід на форум, на якому користувачі можуть створювати топіки за допомогою яких можна здійснювати спілкування та обмінюватися думками з іншими користувачами.[6]

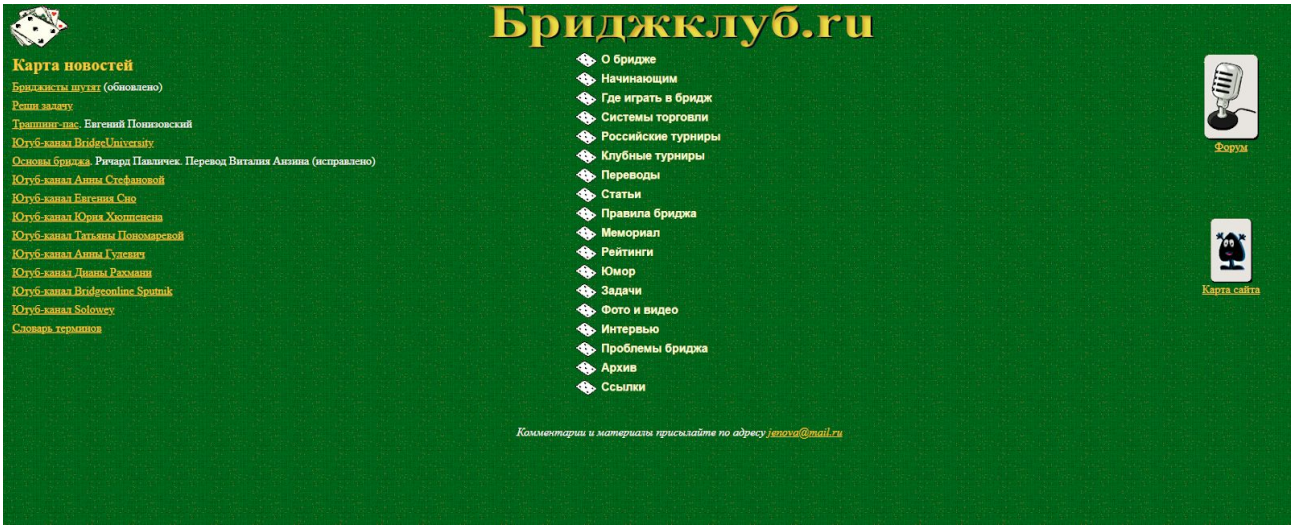

Рисунок 1.1.2 – Зовнішний вигляд веб-сайту Bridgeclub

1.1.3 Bridgewinners

Інформаційний портал, на якому можно створювати аккунт користувача самостійно, що дає йому змогу йому публікувати запитання або пости, а не звертатися за цим до когось з управлінням сайта. На ньому існують логічні завдання на тематику спортивного бриджу, які можно вирішувати будь-якому зареєстрованому користувачу або спитати допомоги. Кожний пост можно коментувати. Що дає змогу отримувати фідбек автору.[7]

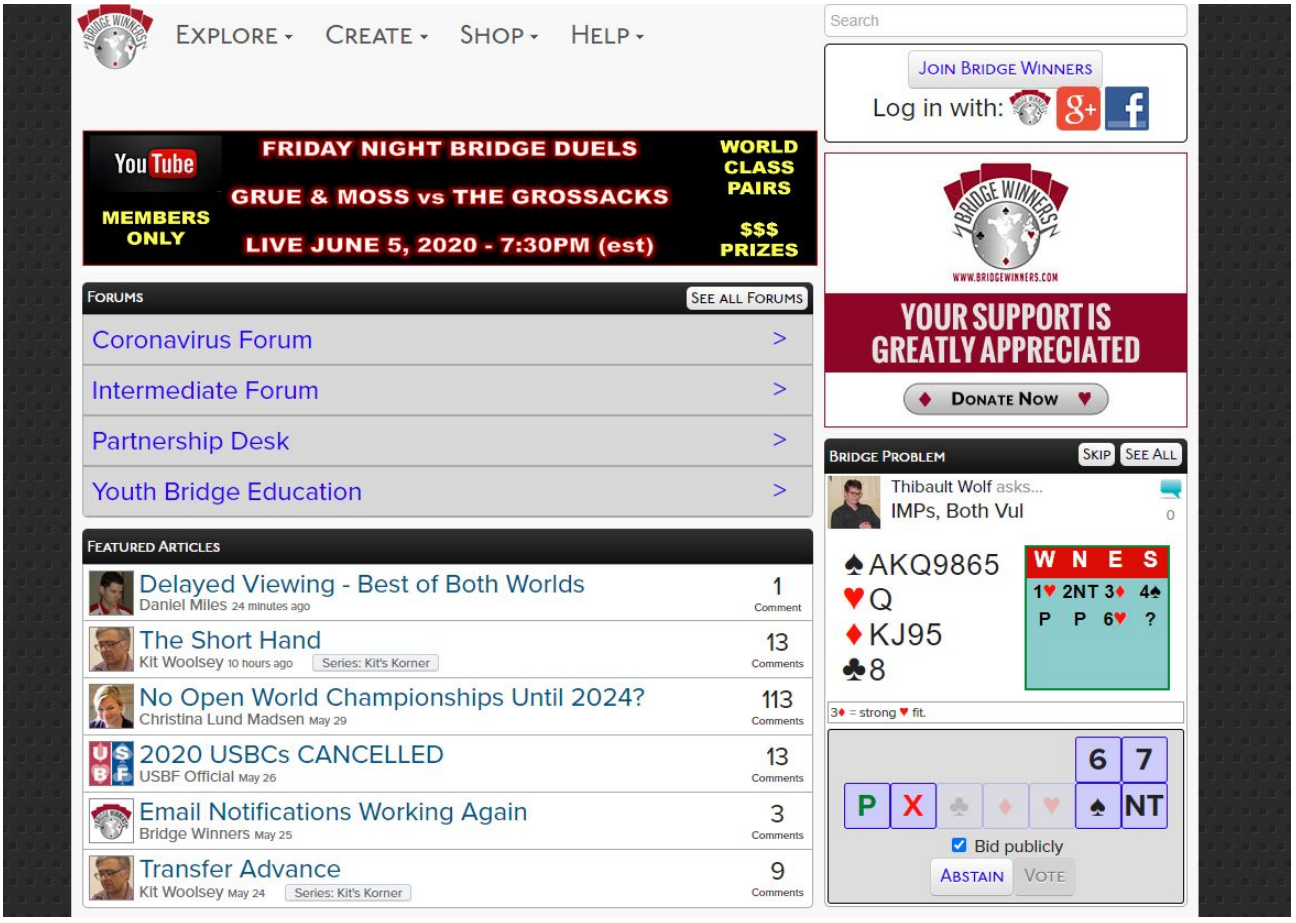

Рисунок 1.3 – Зовнішний вигляд веб-сайту Bridgewinners

1.2 Порівняльна характеристика існуючих інформаційних порталів.

Під час оглядів інформаційних порталів було створена таблиця переваг та недоліків, що проаналізувати її та результаті отриманих даних зробити власний інформаційний портал.

Таблиця 1.1

Інформаційний додаток	Переваги	Недоліки
Bridgebum	1. Зручний інтерфейс 2. Гарно підібраний матеріал 3. Присутня функція пошуку 4. Є телефона адаптація	1. Сайт повністю на англійській мові, тому для тих користувачів, які не знають це буде дуже великою проблемою для користування

		2. Немає функції реєстрації для користувачів, весь матеріал викладається одним користувачем 3. Немає функції комментування, тому буде складно отримувати фідбек.
Bridgeclub	1. Гарно підібраний матеріал 2. Є мапа новин і сайту, яка поліпшує користування порталом 3. Форум, на якому користувачі можуть спілкуватись	1. Відсутня функція пошуку 2. Немає функції реєстрації
Bridgewinners	1. Існує функція пошуку 2. Є функція реєстрації для нових користувачів 3. Є можливість коментувати пости 4. Гарно підібраний матеріал	1. Сайт повністю на англійській мові, тому для тих користувачів, які не знають це буде дуже великою проблемою для користування

Отримуючи результат після аналізу наведений вище прикладів було вирішено розробити власний інформаційний портал.

1.3 Постановка задачі

Інформаційний портал буде мати вигляд блог-сайт на якому будь-який користувач зможе зареєструватися і публікувати пости. Якщо користувач забуде або забажає змінити свій пароль, то в нього буде ця можливість. Кожен пост буде проходити верифікацію. Завдяки цьому контент, який будет публікуватися проходити фільтрацію. Все це спричинить для з'явлення якісного контенту сайта. Кожен пост можно буде оцінювати, тобо будет система лайків, щоб можно було відстежувати якість. Кожен пост можно буде коментувати, щоб автор посту та інші користувачі мали змогу обмінюватися думками, а автор отримувати фідбек від своєї роботи. Буде реалізована функція пошуку, щоб прискорити знаходження певної інформації.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Мова програмування Java

Існують дуже багато мов програмування. Вони поділяються на об’єктно - орієнтовані та процедурні мови програмування. Різниця між ними в тому, що для рішення невеликих задач використовують процедурний спосіб програмування. Проте при розробці великих проектів класи й методи мають великі переваги, а саме:

1. Класи надають зручний механізм кластеризації методів;
2. Легке розподілення роботи між програмістами;
3. Присутня інкапсуляція: класи приховують деталі уявлення даних від будь-якого коду, окрім своїх методів;

Тому, при написанні простого веб-серверу використовуючи процедурний метод програмування програмісту знадобиться приблизно 2000 процедур, в той час об’єктно-орієнтованим методом вийде приблизно 20 класів. Да якщо програміст допустить помилку, то йому буде легше знайти її серед 20 класів, ніж серед 2000 процедур.[14] Тому вибір пав на об’єктно-орієнтовану мову програмування. Одна з таких мов - Java.

Java - один із найпопулярніших мов програмування.

Java - строго типізований об'єктно-орієнтована мова програмування, розроблений компанією Sun Microsystems (в подальшому придбаної компанією Oracle). Розробка ведеться співтовариством, організованим через Java Community Process, мова і основні реалізують його технології поширюються за ліцензією GPL. Права на торговельну марку належать корпорації Oracle.

Програми на Java транслюються в байт-код Java, який виконується віртуальною машиною Java (JVM) - програмою, обробній байтовий код і передавальної інструкції обладнанню як інтерпретатор.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.[4]

Отже, можно підвести підсумки, чому саме було обрана мова програмування Java:

1. Гнучка система безпеки;
2. Об'єктно-орієнтована мова програмування;
3. Багато літератури, яка допомагає розібратися з усіма аспектами її.
4. Развиті фреймовркі;
5. Величезна кількість бібліотек, які поліпшують роботу.
6. Простота коду та його ефективність.

2.2 Вибір СУБД

База даних (БД) - це організована структура, призначена для зберігання, зміни і обробки взаємозалежної інформації, переважно великих обсягів. Бази даних активно використовуються для динамічних сайтів зі значними обсягами даних - часто це інтернет-магазини, портали, корпоративні сайти. Такі сайти зазвичай розроблені за допомогою серверного мови програмування (як приклад, PHP) або на основі CMS (як приклад, WordPress), і не мають готових сторінок з даними за аналогією з HTML-сайтами.

В контексті баз даних варто розглянути поняття СУБД. Система управління базами даних (СУБД) - це комплекс програмних засобів, необхідних для створення структури нової бази, її наповнення,відображення інформації і редагування вмісту.

А здійснює цей доступ до даних СУБД за допомогою спеціальної мови - SQL.

Існує дуже багато СУБД. Найбільш поширені є Oracle, MySql, PostgreSQL, Microsoft SQL. Саме ці СУБД найчастіше використовуються серед тупу клієнт-сервер.

PostgreSQL надає безліч різних можливостей, досить надійна і має хороші характеристики по продуктивності. Вона працює практично на всіх UNIX-платформах, включаючи UNIX-подібні системи, такі як FreeBSD і Linux. Її можна застосовувати на Windows NT Server і Windows 2000 Server, а для розробки годяться навіть такі системи Microsoft для робочих станцій, як ME. Крім того, PostgreSQL вільно поширюється і має відкритий вихідний код.

PostgreSQL має всі ті функції, які маю інші платні СУБД, а також тими функціями, що роблять її унікальною серед інших.

Функціїї PostgreSQL:

1. Транзакції;
2. Вкладені запити;
3. Уявлення;
4. Типи, обумовленими користувачем;
5. Посильна цілісність.

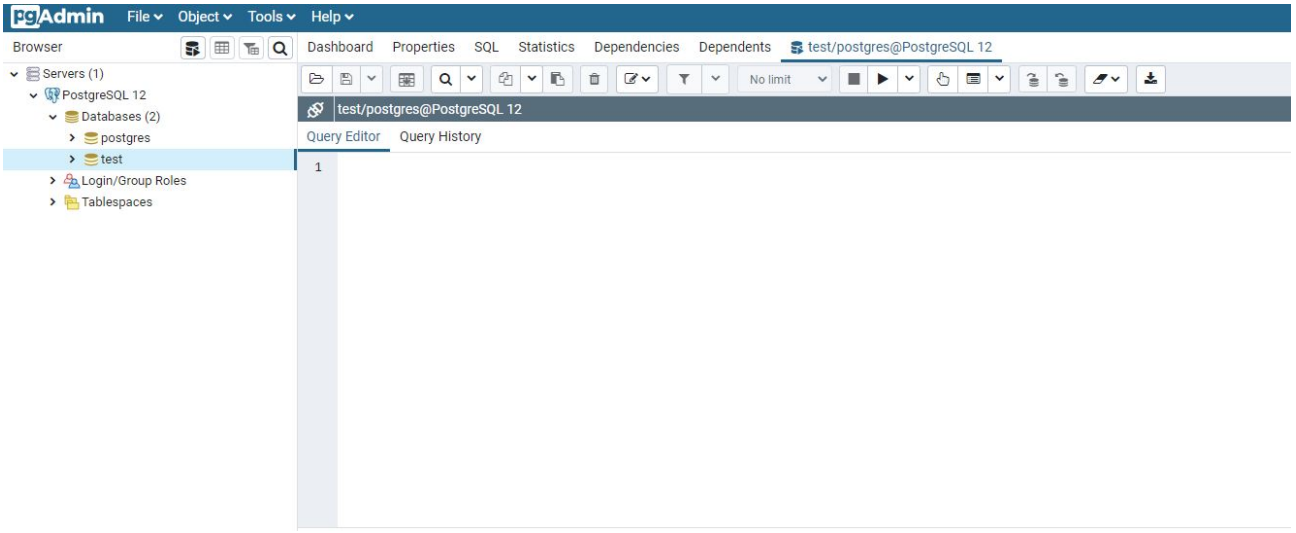

Рисунок 2.2.1 – Зовнішний вигляд PostgreSQL для адміністрування бази даних

Проаналізувавши переваги СУБД PostgreSQL вона була обрана.[2]

2.3 Вибір фреймворку

Фреймворк - програмна платформа, яка визначає структуру програмної системи; програмне забезпечення, що полегшує розробку і об'єднання різних компонентів великого програмного проекту.

Фреймворк визначається як безліч конкретних і абстрактних класів, а також визначень способів їх взаємини. Конкретні класи зазвичай реалізують взаємовідносини між класами. Абстрактні класи являють собою точки розширення, в яких каркаси можуть бути використані або адаптовані.

Одним з таких фреймворку - Javalin. Це простий веб-фреймворк для мов програмування Java та Kotlin.

Основні його переваги:

1. Простий.

   На відміну від інших веб-рамок Java та Kotlin, Javalin має дуже мало понять, які вам потрібно вивчити. Ви ніколи не розширюєте класи і рідко реалізуєте інтерфейси.

2. Інтероперабельний

   Інші веб-рамки Java та Kotlin зазвичай пропонують окрему версію для кожної мови. Javalin розробляється з урахуванням сумісності, тому програми будуються однаково як у Java, так і в Kotlin.

3. Гнучкий

   Javalin розроблений таким чином, щоб бути простим і блокуючим, оскільки це найпростіша модель програмування. Однак якщо в результаті встановити майбутнє, Javalin переходить в асинхронний режим.

4. Інтеграція з OpenAPI

   Багато легких веб-рамок Java та Kotlin не підтримують OpenAPI, але Javalin має повну інтеграцію, включаючи інтерфейс Swagger і повторно налаштований для відображення генеруючих документів.

Підводячи підсумки, можно зрозуміти, що даний фреймворк дуже добре вписується в концепцію розробки інформаційного порталу. Він полегшить та заощадить час на разробку. [1]

2.5 Вибір середовища розробки

Оскільки була обрана мова програмування Java, то і середовище потрібно, яке зможе на максимум реалізувати потенціал. Наприклад: NetBeans, IntelliJ IDEA, Eclipse. Було обрано IntelliJ IDEA 2020 Community edition, як середу програмування.

IntelliJ IDEA - інтегроване середовище розробки програмного забезпечення для багатьох мов програмування, зокрема Java, JavaScript, Python, розроблена компанією JetBrains.

Починаючи з версії 9.0, середа доступна в двох редакціях: Community Edition і Ultimate Edition. Community Edition є повністю вільною версією, доступною під ліцензією Apache 2.0, в ній реалізована повна підтримка Java SE, Kotlin, Groovy, Scala, а також інтеграція з найбільш популярними системами

управління версіями. В редакції Ultimate Edition, доступною під комерційною ліцензією, реалізована підтримка Java EE, UML-діаграм, підрахунок покриття коду, а також підтримка інших систем управління версіями, мов та фреймворків.

В IntelliJ IDEA можливо приєднатись до бази даних і здійснювати адміністрування її. [9]

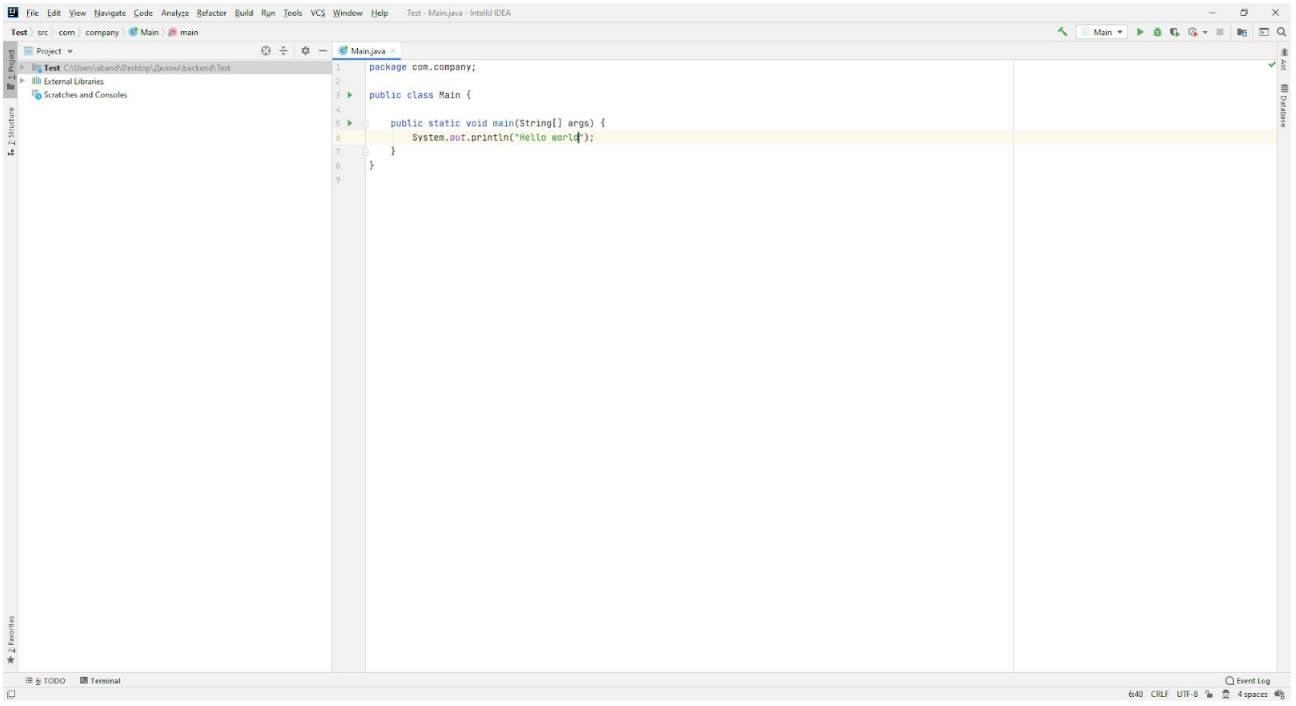

Рисунок 2.5.1 – Зовнішний вигляд IntelliJ IDEA

2.6 Front-end сайту

Для розробки Front-end сайту буде використано html, css, js.

HTML (від англ. HyperText Markup Language - «мова гіпертекстової розмітки») - стандартизований мову розмітки документів у Всесвітній павутині. Більшість веб-сторінок містять опис розмітки на мові HTML (або XHTML). Мова HTML інтерпретується браузерами; отриманий в результаті інтерпретації форматований текст відображається на екрані монітора комп'ютера або мобільного пристрою.[3]

CSS - формальна мова опису зовнішнього вигляду документа, написаного з використанням мови розмітки.

Переважно використовується як засіб опису, оформлення зовнішнього вигляду веб-сторінок, написаних за допомогою мов розмітки HTML і XHTML, але може також застосовуватися до будь-яких XML-документах, наприклад, до SVG або XUL.[10]

JavaScript - мультіпарадігменний мову програмування. Підтримує об'єктно-орієнтована, імперативний і функціональний стилі.

JavaScript використовується як вбудований мова для програмного доступу до об'єктів додатків. Найбільш широке застосування знаходить в браузерах як мова сценаріїв для додавання інтерактивності веб-сторінок. [11]

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Проектування бази даних

Одним з важливих етапів розробки це проектування бази даних. Проектування було здійснено за допомогою CaseStudio, тому що в ньом обрати під яку саме СУБД проектується і в кінці згенерувати скрипт, який допоможе створити таблиці та відношення між ними. На етапі проектування бази даних було вирішено, який саме функціонал будет на інформаційному порталі. Тому спроектована база даних дає змогу зрозуміти, які функцію потрібно буде реалізувати, як вони будуть пов'язані між собою та як буде будуватися архітектура веб-сайту.[15]

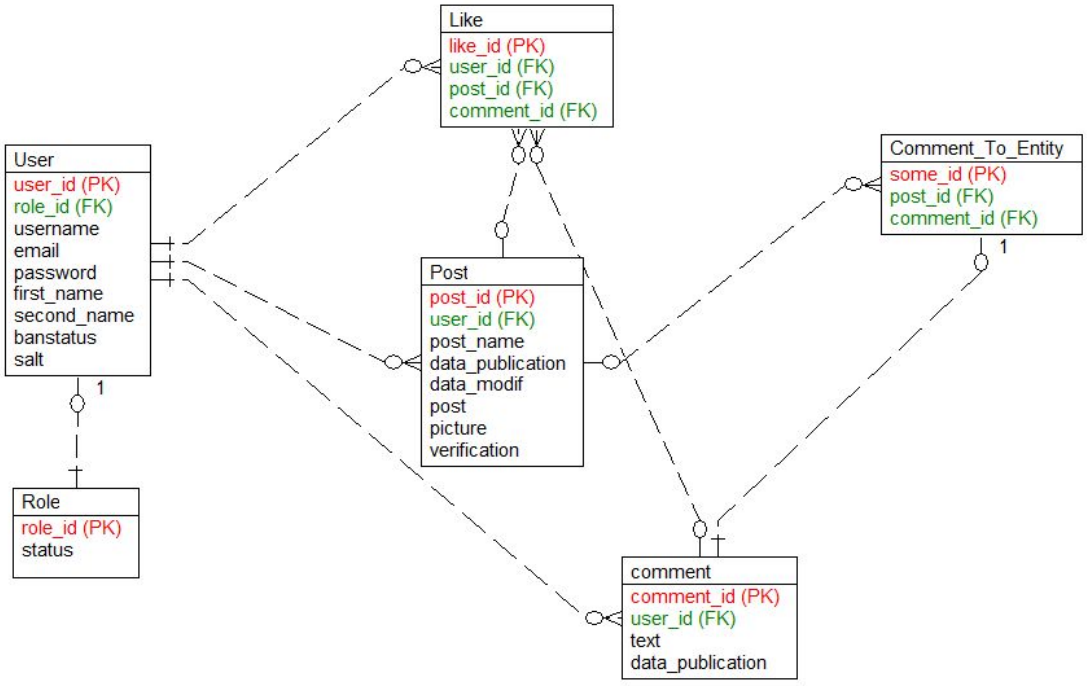

Рисунок 3.1 – ER діаграма інформаційного порталу спортивний брідж

Представлення логіки Інформаційного порталу у вигляді таблиці:

Таблиця 3.1

Таблиця	Поле	Тип	Ключ	Обмеження	Зміст
Users	user_id	Serial	Primary key	not null	Номер користувача
	username	Varchar(50)		unique, not null	Юзернейм користувача
	email	Varchar(50)		unique, not null	Пошта користувача
	password	Varchar(50)		not null	Пароль користувача
	first_name	Varchar(50)		not null	Ім’я користувача
	second_name	Varchar(50)		not null	Прізвище користувача
	banstatus	Numeric(1)		not null default 0	Статус користувача
	salt	Char(20)		not null	Сума для шифрування користувача
	role_id	Integer	Foreign key	not null	Номер ролі
Posts	post_id	Serial	Primary key	not null	Номер ролі
	user_id	Integer	Foreign key	not null	Автор поста
	post_name	Varchar(50)		not null	Заголовок

					поста
	data_publication	Timestamp		not null	Дата публікації
	data_modif	Timestamp		not null	Дата зміни публикації
	post	Varchar(255)		not null	Інформація
	picture	Varchar(255)		not null	Зображення посту
	verification	Numeric(1)		not null default 0	Верифікація посту
Likes	like_id	Serial	Primary key	not null	Номер лайку
	user_id	Integer	Foreign key	not null	Автор лайку
	post_id	Integer	Foreign key	not null	Номер посту
	comment_id	Integer	Foreign key	not null	Номер коментаря до посту
Roles	role_id	Serial	Primary key	not null	Номер ролі
	status	Varchar(50)		not null	Роль користувача
Comment_to_entity	some_id	Serial	Primary key	not null	Номер відношення коментаря
	post_id	Integer	Foreign key	not null	Номер посту
	comment_id	Integer	Foreign key	not null	Номер коментаря посту

Практична реалізація БД наведено у Додатку А.

Реалізація запитів, які будуть реалізовувати функціонал сайту [12]:

1. Авторизація користувача

   select username,user_id from users where email = ? and password = ? ;

2. Перевірка пошти (якщо користувач зареєстрований, то його пошта вже буду в базі даних)

   select username from users where email = ?;

3. Забанить користувача

   update users set banstatus = 1 where user_id = 4 and EXISTS(select user_id from users where role_id = 2 and user_id = ?);

4. Зміна паролю

   update users set password = '5555', salt= ? where user_id = 2;

5. Відображення постів на інформаційному порталі

   select p.post_id,p.user_id,p.post_name,p.data_publication,p.data_modif, p.picture, l.likes_count from posts p left join (select post_id,count(*) as likes_count from likes group by post_id) l on (p.post_id = l.post_id) where p.verification = 1 order by data_modif desc

6. Вміст посту

   select p.post_id,p.user_id,p.post_name,p.data_publication,p.data_modif,p.post, p.picture,l.likes_count from posts p left join (select post_id,count(*) as likes_count from likes group by post_id) l on (p.post_id = l.post_id) where p.post_id = ? and p.verification = 1;

7. Відобразити пости, які знаходяться на верифікації (Для адміну)

   selectp.post_id, p.user_id, p.post_name,p.data_publication,p.data_modif, p.picture, l.likes_count from posts p left join (select post_id,count(*) as likes_count from likes group by post_id) l on (p.post_id = l.post_id) where

p.verification = 0 and EXISTS(select user_id from users where role_id = 2 and user_id = ?) order by p.data_modif

8. Верифікація

   update posts set verification = 1 where post_id = ? and EXISTS(select user_id from users where role_id = 2 and user_id = ?);

9. Створити пост

   insert into posts values(null,36,'About',sysdate,sysdate, 'friends','link',0);

10. Оновлення посту

    update posts set post_name = ?, post = ?, data_modif = sysdate, picture = ?, verification = 0 where post_id = ? and (user_id in(select user_id from users where role_id = 2)or user_id = ?);

11. Оцінювання посту

    insert into likes select null,user_id,post_id,null from dual where not EXISTS(select 1 from likes where user_id = ? and post_id = ?);

3.2 Проектування інтерфейсу.

Значну роль буде відігравати графічний інтерфейс. Будь-який користувач повинен з легкістю користуватися інформаційним порталом. Після проектування бази даних будет легко спроектований графічний інтерфейс. Інтерфейс був реалізований за допомогою draw.io [13]

3.2.1 Функція авторизації

Для користування сайтом потрібно буде пройти авторизацію. Структура авторизації:

1. Поле для юзернейму;
2. Поле для паролю;
3. Кнопка для входу;
4. Посилання для реєстрації;

5. Посилання на відновлення паролю

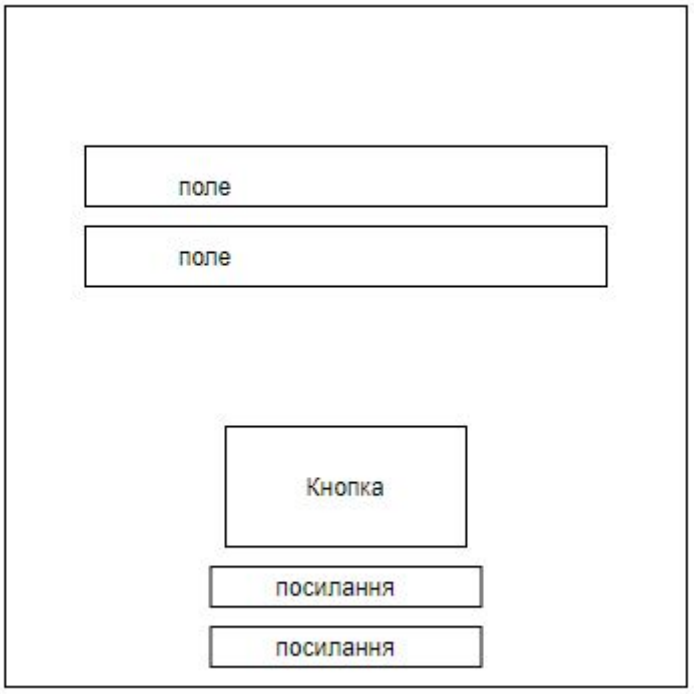

Рисунок 3.2.1 – Вікно авторизації

3.2.2 Функція регістрації

Для нових користувачів потрібна будет функція реєстрації. Структура сторінки:

1. Поле для юзернейму;
2. Поле для пошти;
3. Поле для паролю;
4. Поле для імені;
5. Поле для прізвища.

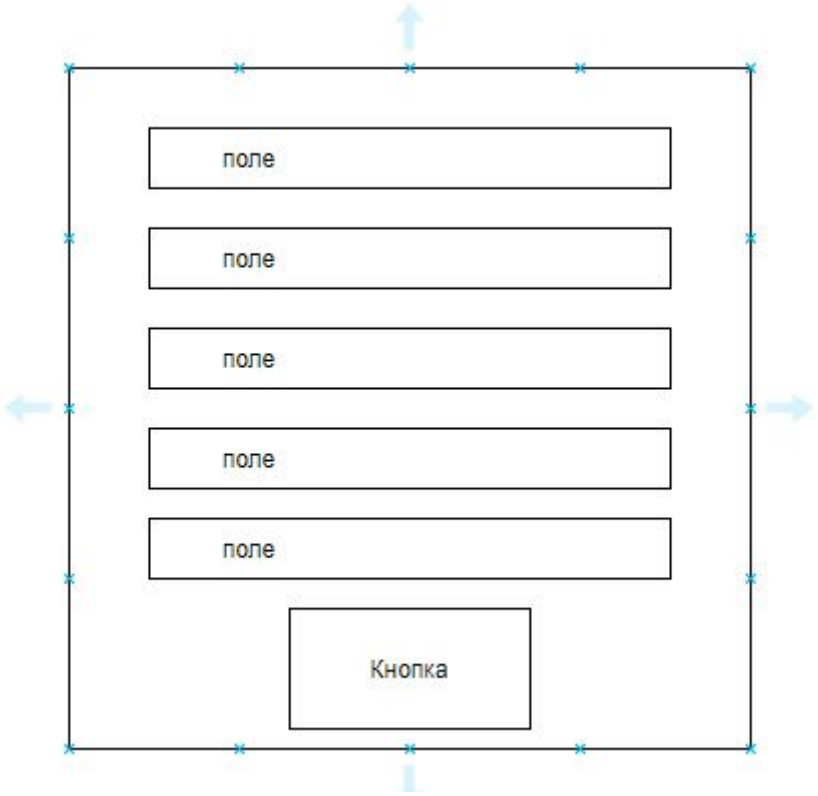

Рисунок 3.2.2 – Вікно реєстрації

3.2.3 Головна сторінка

На головній сторінці будуть відображатися всі пости. Функція пошуку, на посту буде кнопка для встановлення лайків та перехід в глубину посту. Кнопки в шапці сайту, які слугують для реалізації функціоналу: зміна паролю, адмін панель, створити пост та вихід.

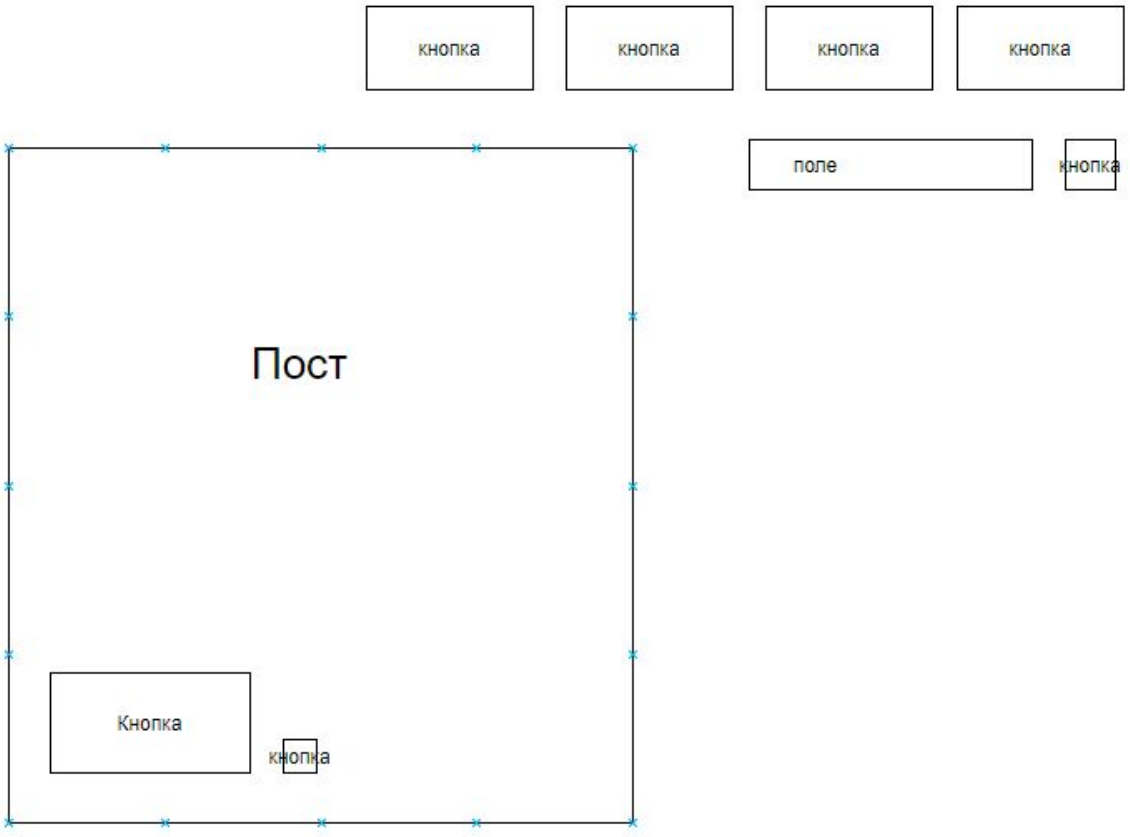

Рисунок 3.2.3 – Вікно реєстрації

3.2.4 Створення посту

На сторінці створенню посту:

1. Поле для назви посту;
2. Поле для заповнення посту інформацією;
3. Кнопка, для загрузки зображення;
4. Кнопка створення посту.

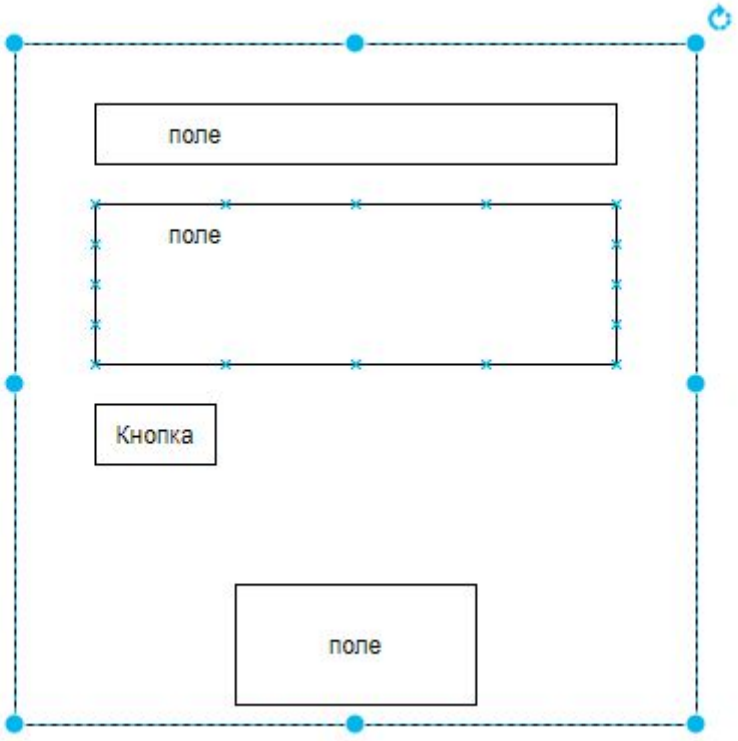

Рисунок 3.2.4 – Вікно створення посту

3.2.5 Відновлення паролю.

Якщо користувач забуде свій пароль, то в нього буде можливість його відновити. Структура сторінки для відновлення паролю:

1. Поле для введення пошти;
2. Кнопка для відправки нового паролю.

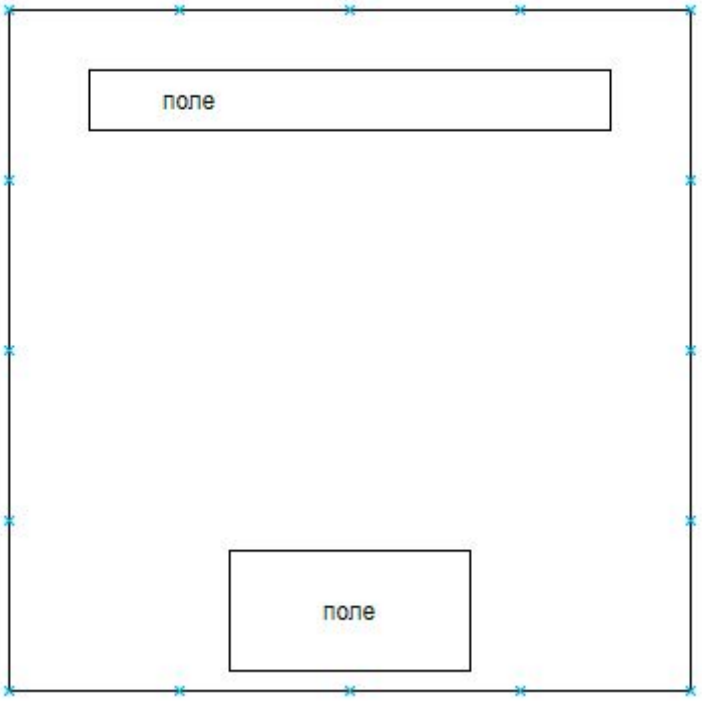

Рисунок 3.2.5 – Вікно відновлення посту

3.2.6 Зміна паролю.

Кожен користувач може змінювати свій пароль. Структура сторінки для зміни паролю:

1. Поле для вводу старого паролю;
2. Поле для вводу нового паролю;
3. Кнопка для зміни.

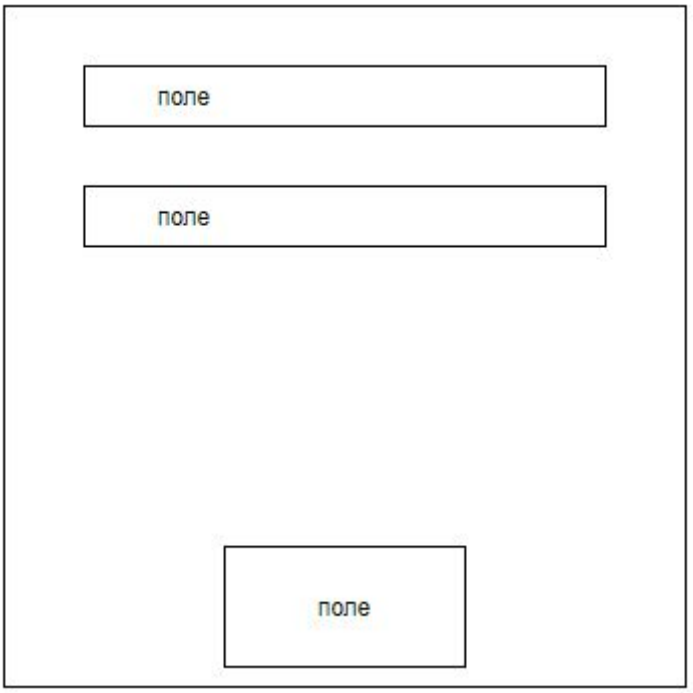

Рисунок 3.2.6 – Вікно для зміни паролю

3.2.7 Адмін сторінка.

Сторінка для адміністратора, на якій має змогу зайти тільки адмін. На цій сторінці адмін буде перевіряти пости та й публікувати їх. Структура:

1. Кнопка для пошуку;
2. Поле для вводу інформації для пошуку;
3. Кнопка для виходу з аккаунту;
4. Кнопка для зміни паролю;
5. Кнопка для переходу в сам пост;
6. Кнопка для верифікації посту.

Рисунок 3.2.7 – Вікно адмін сторінки

3.3 Розробка додатку.

Структура проекту інформаційного порталу:

1. Папка api

   В цій папці зберігаються api файли. API - складова частина серверу, яка отримує запити та відправляє їх.

2. Папка logs

   В цій папці знаходиться текстовий файл, в який записуються логи роботи додатку.

3. Файл BridgeGameApp

   Це модульний файл, який зберігає інформацію о модулі розвитку, котрий може бути в Java або компоненти Maven. Зберігає шляхи до модулів, залежності та інше параметри.

4. Файл pom.xml

   Інформація для програмного проекту, підтримуючий Maven. При використані Maven перевіряє, чи містить цей файл усі потрібні дані та чи всі дані синтаксично вірно записані.

5. Файл properties.txt

   Містить параметри для старту роботи програми

6. Папка Src містить в собі 2 паки, одна з них відповідає на back-end (це папка java), а інша папка за front-end (папка resources)

Старт програми починається з класу BridgeGameApp. В ньому знаходиться метод public static void main(String[] args).

Далі відбувається підключення нашого фреймворку Javalin - Javalin app = Javalin.*create*();

Запускаємо додаток на певному порту. Порт зчитується з файлу за properties.txt за допомогою статичного методу getPort з класу Property. App.start(Property.*getPort*())

Підключення до бази даних здійснюється за допомогою jdbc.

Після цього викликаються методи:

1. SessionSecurity sessionSecurity = new SessionSecurityControllerImpl();

   Відповідає за надійність сесій.

2. Notification mailNotification = new NotificationImpl();

   Відповідає за надсилання повідомлень на пошту.

3. MailSecurity mailSecurity = new MailSecurityControllerImpl();

   Відповідає за надійність пошти.

4. PasswordSecurity passwordSecurity = new PasswordSecurityController();

   Відповідає за шифрування паролю.

5. MainController mainAppController = new MainAppControllerImpl(app, daoConnection, sessionSecurity, mailNotification, mailSecurity,

passwordSecurity);

Виклає усі контроллери.

Після усіх пройдених дій запускається додаток. Результатом цього ми бачимо повідомлення.[8]

Рисунок 3.3.1 – Вдалий запуск додатку

Лістинг програми буде знаходиться в Додатку Б.

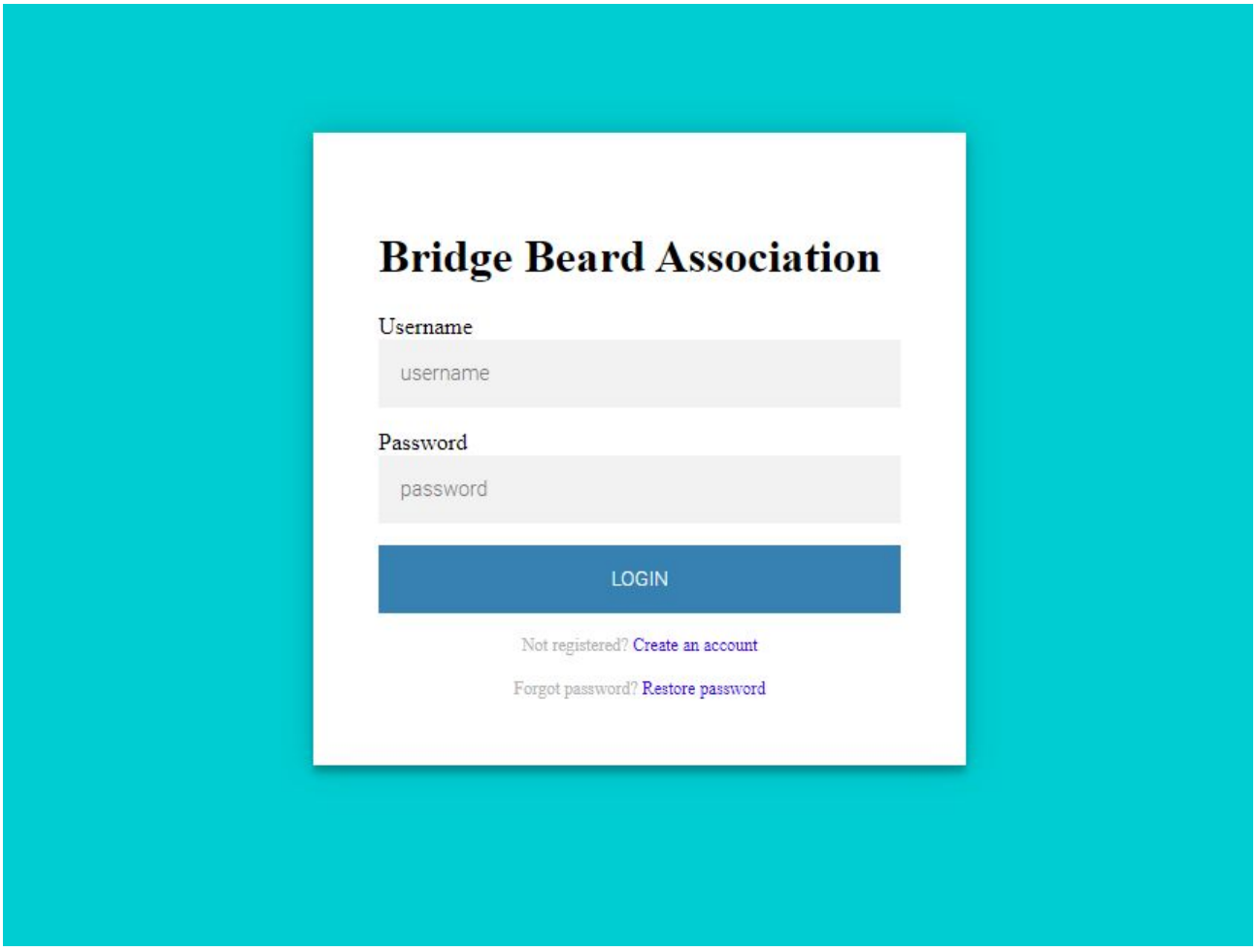

Рисунок 3.3.2 – Стартове вікно авторизації

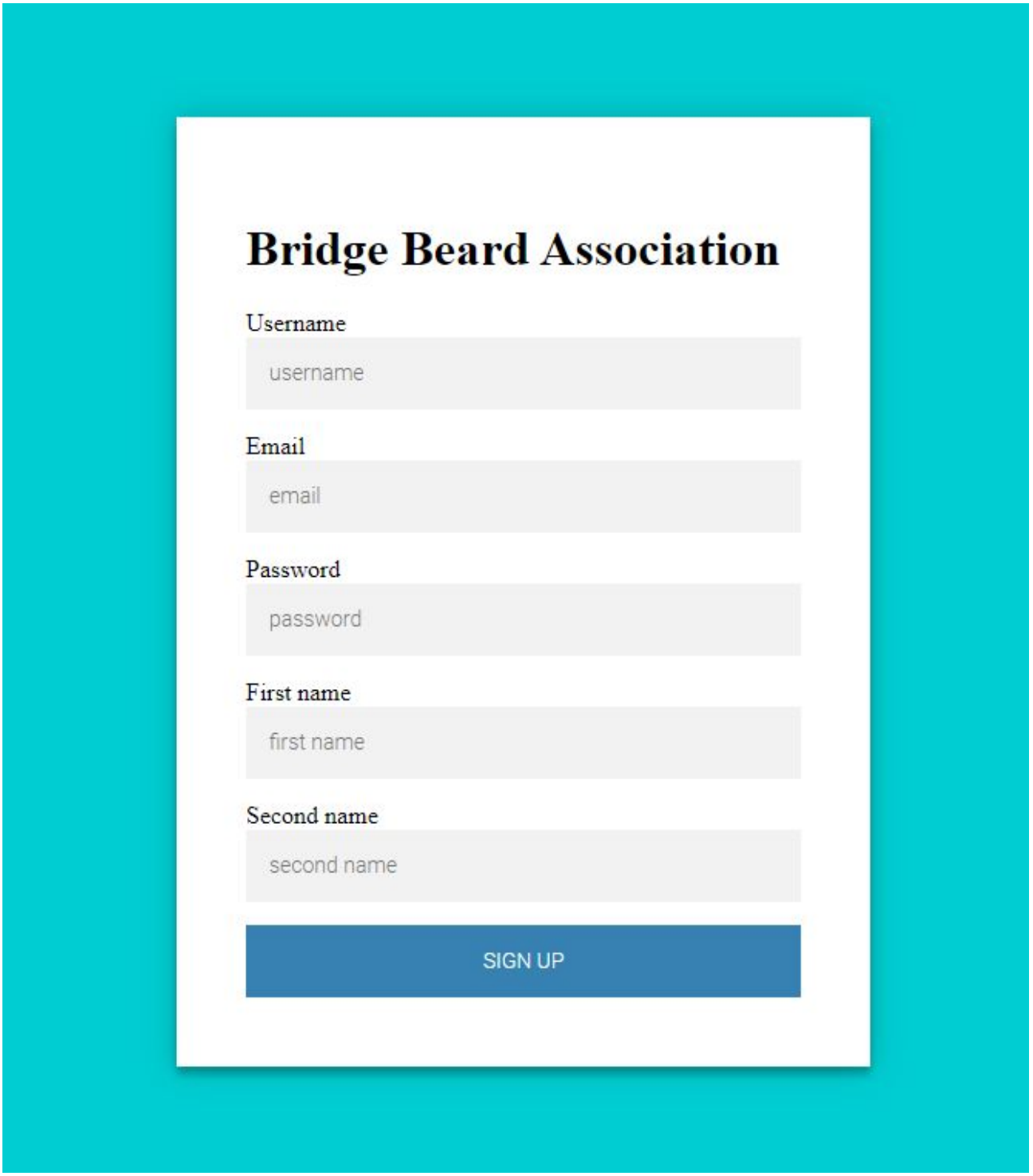

Рисунок 3.3.3 – Вікно реєстрації

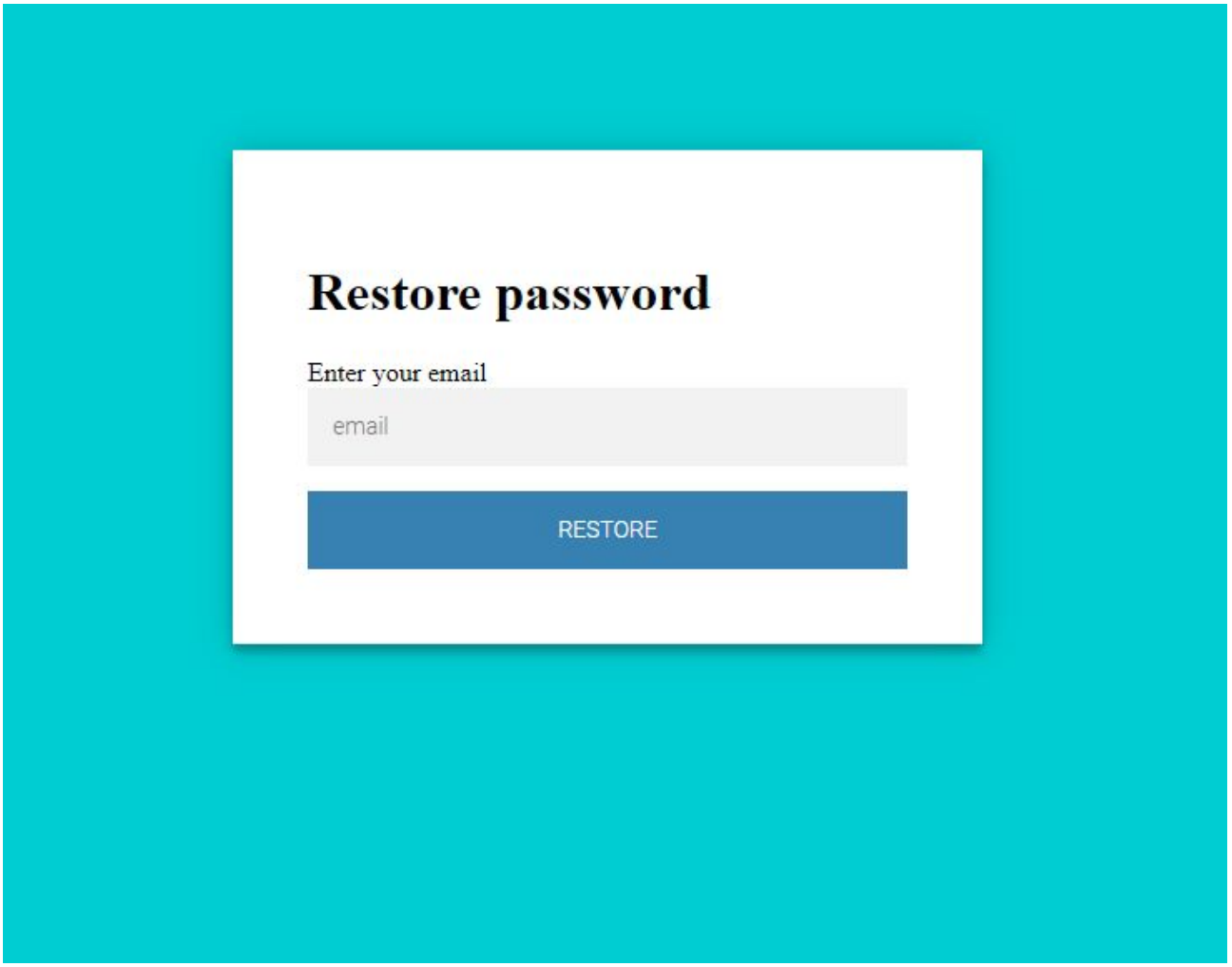

Рисунок 3.3.4 – Вікно відновлення паролю

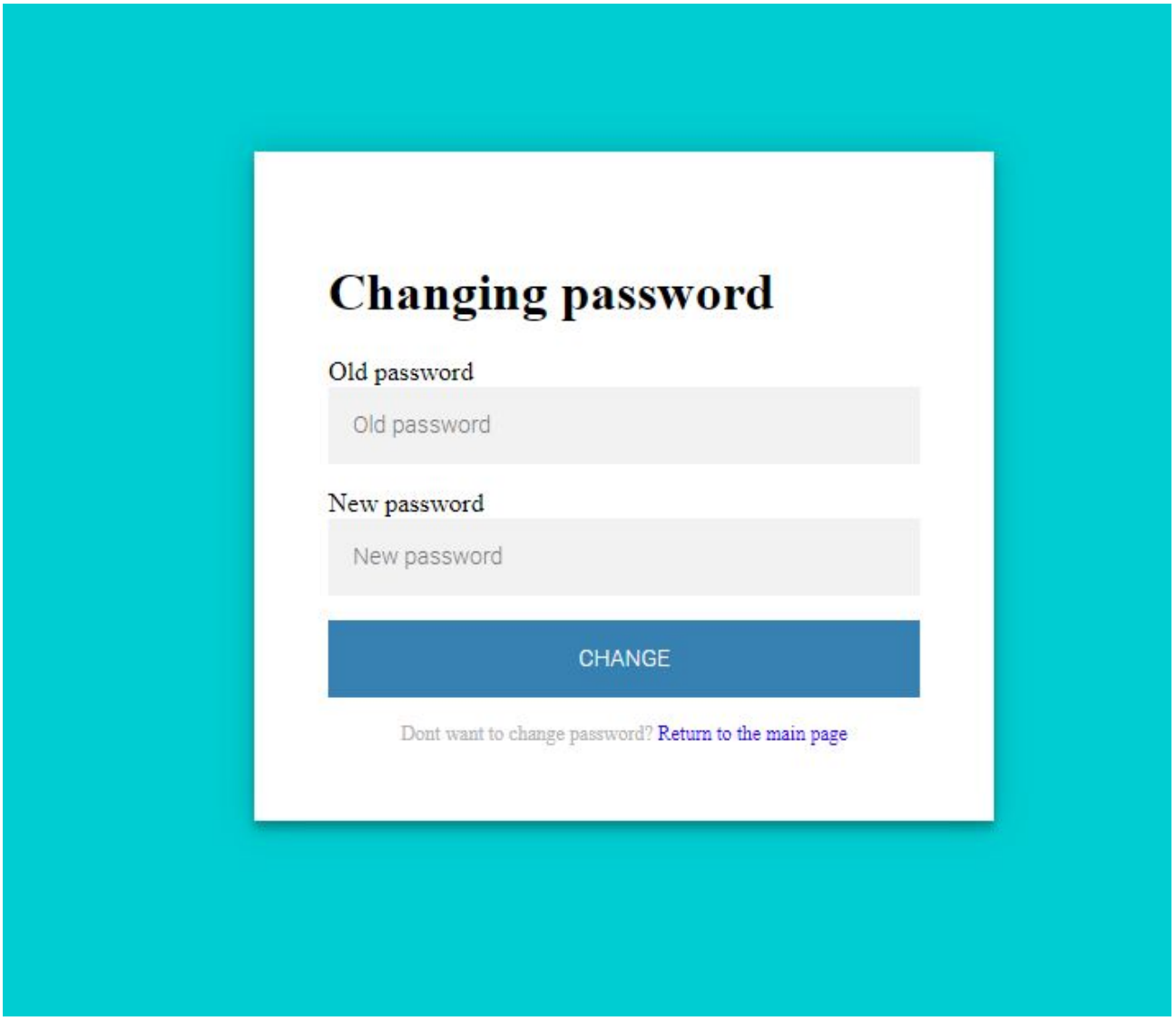

Рисунок 3.3.4 – Вікно зміни паролю

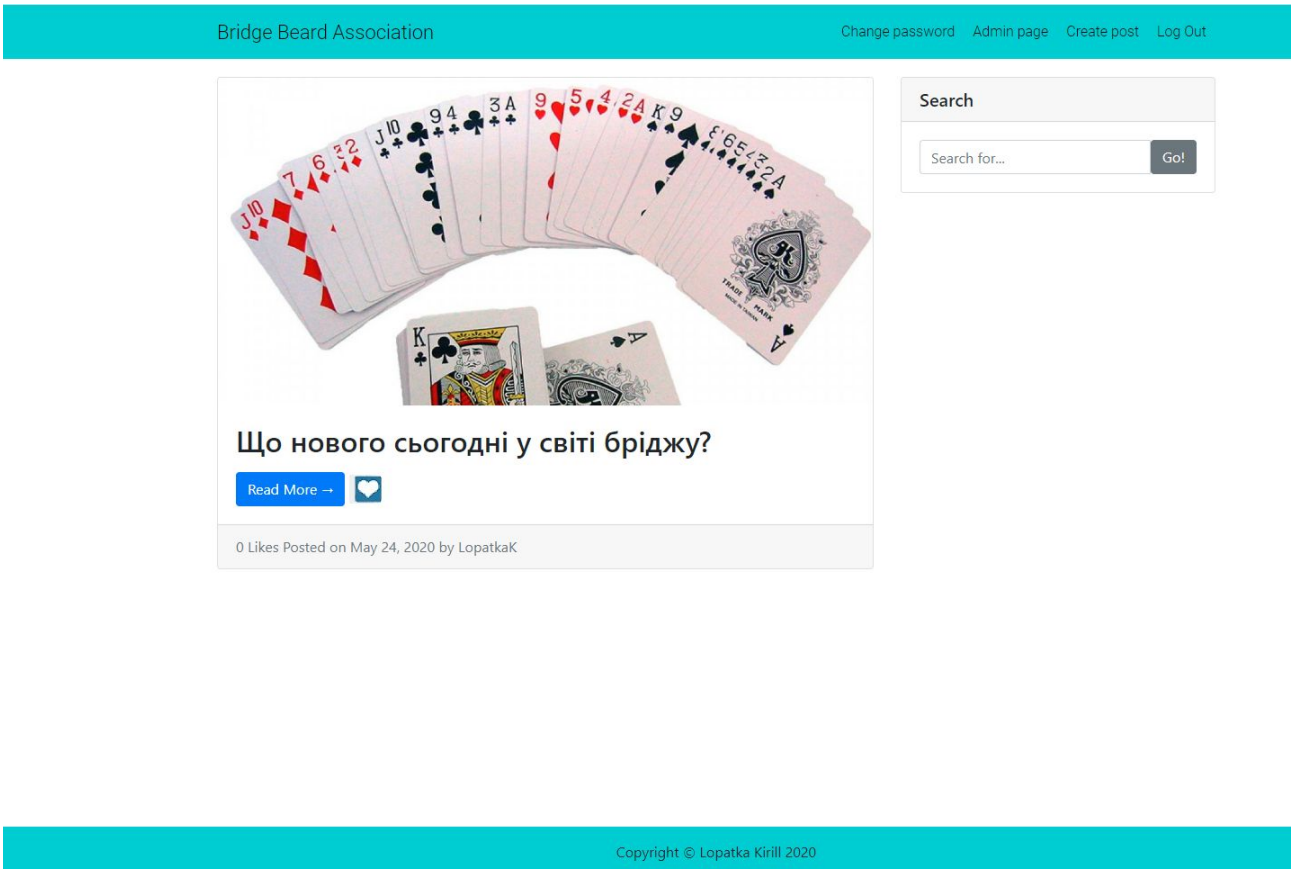

Рисунок 3.3.5 – Початкова сторінка

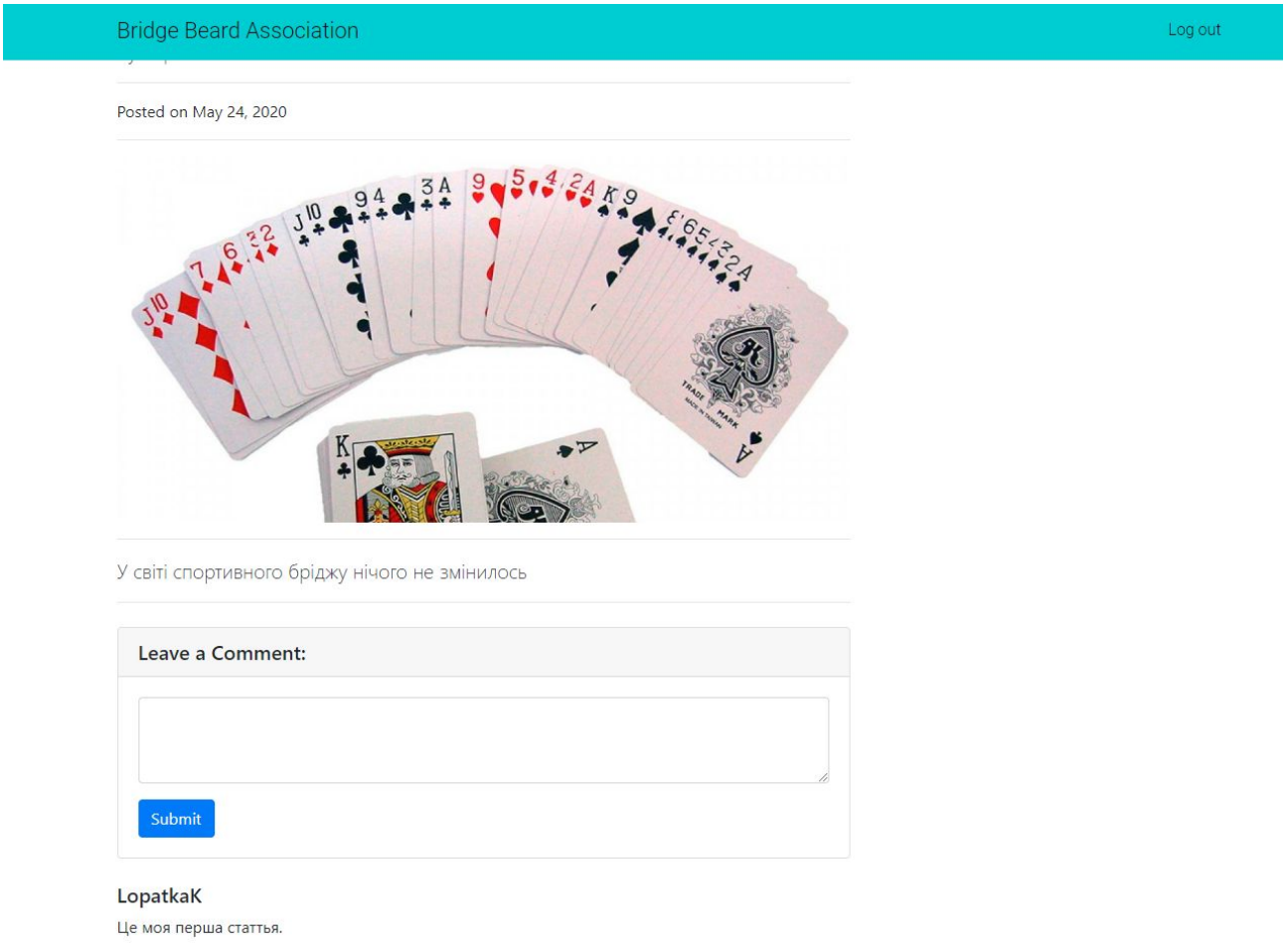

Рисунок 3.3.6 – Вміст посту

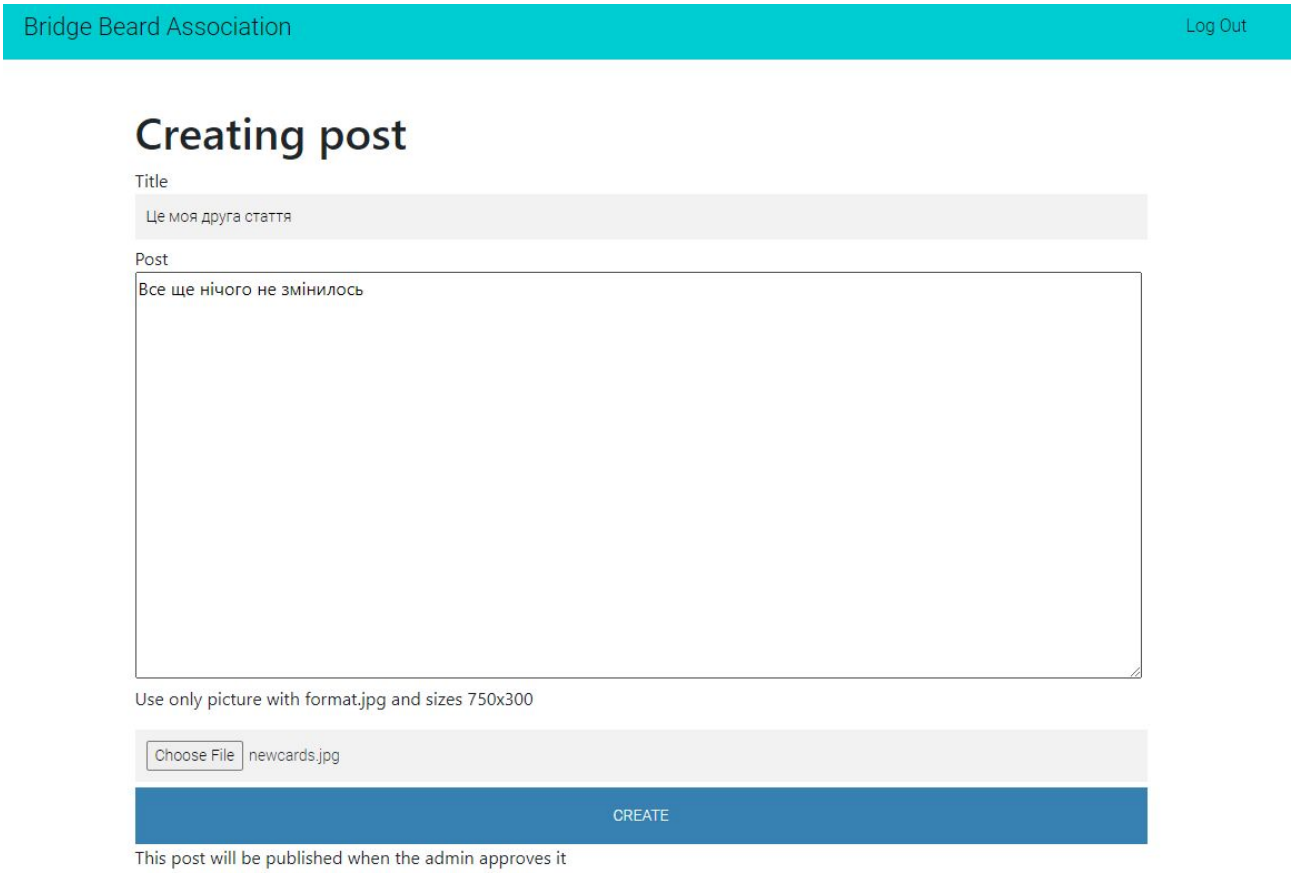

Рисунок 3.3.7 – Створення посту

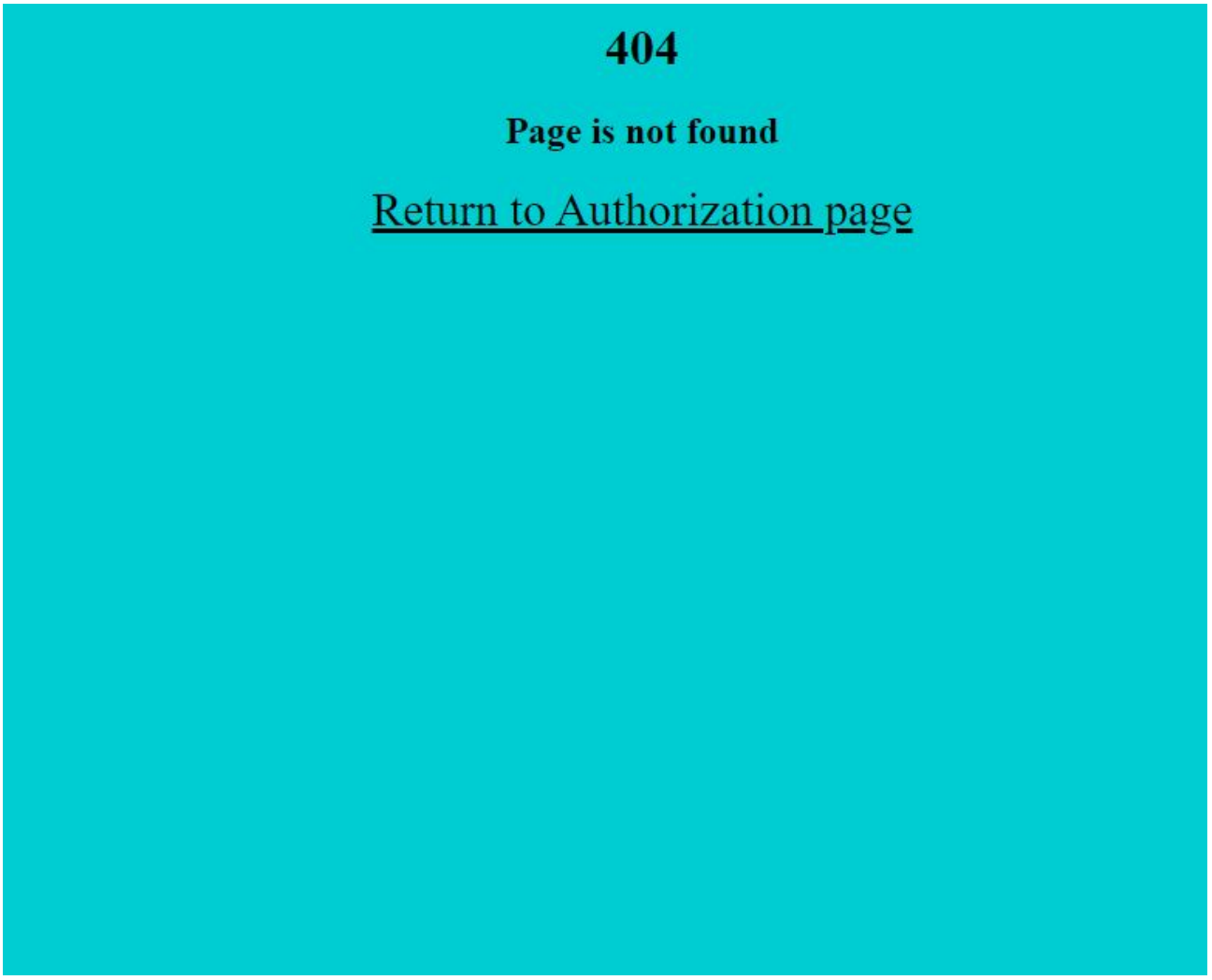

Рисунок 3.3.8 – Перехід на неіснуючу сторінку

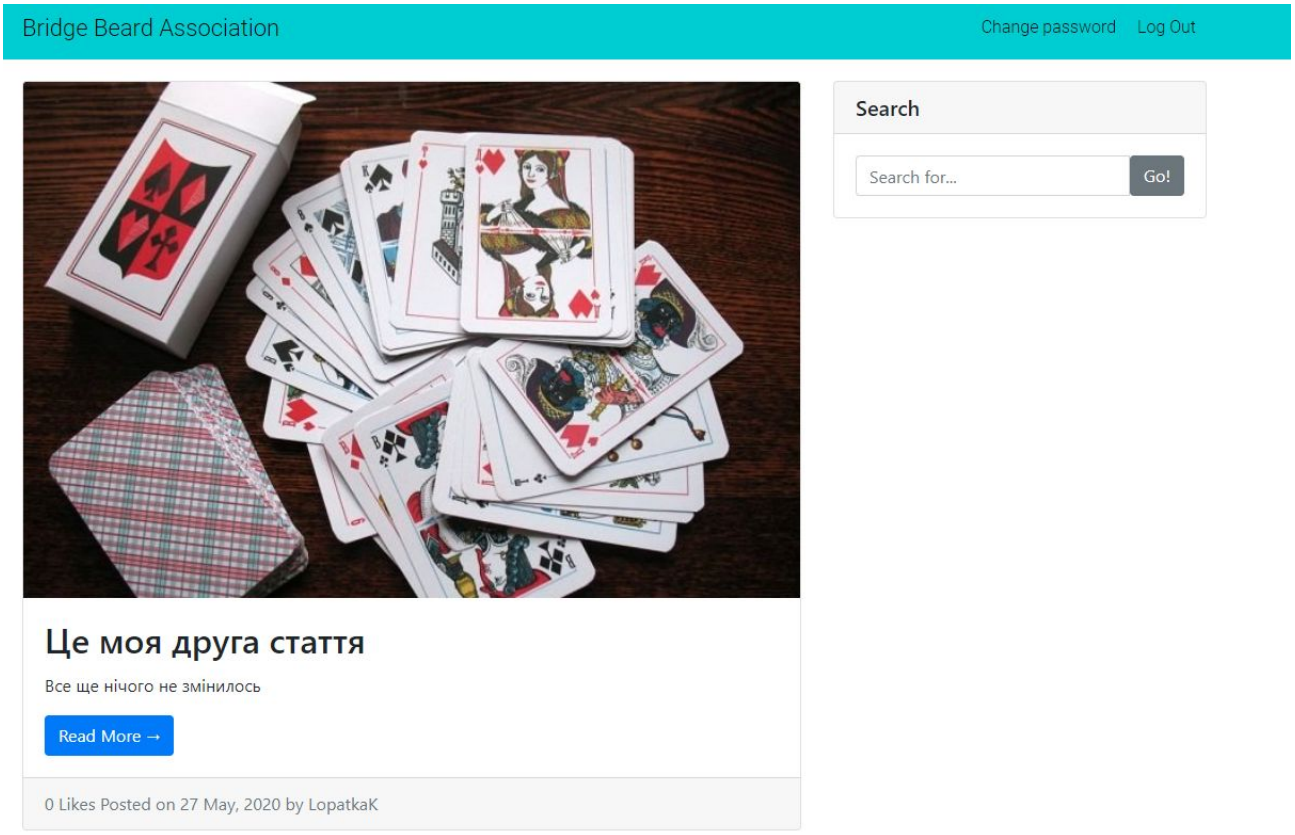

Рисунок 3.3.9 – Адмін сторінка

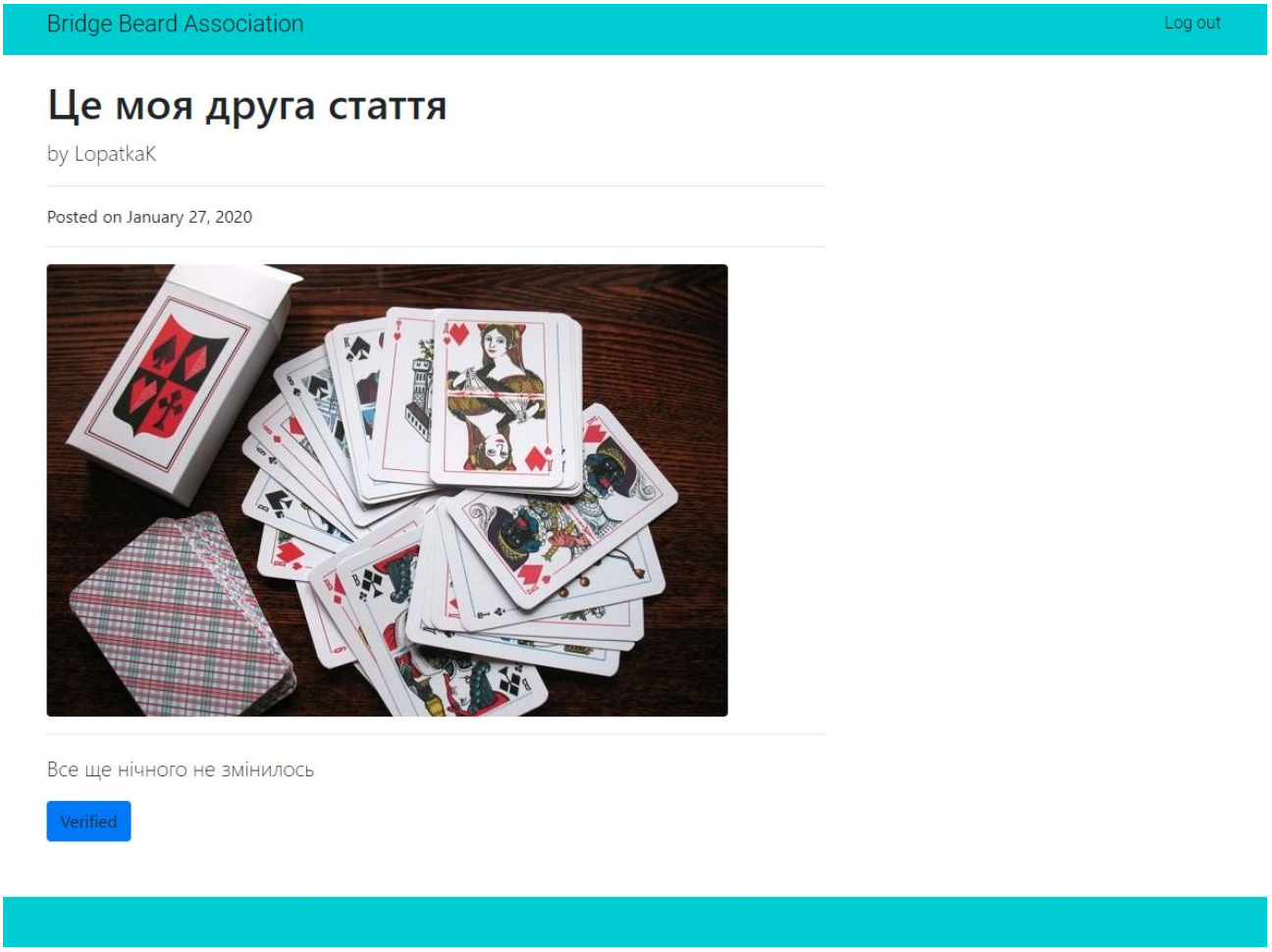

Рисунок 3.3.10 – Вміст поста через адмін права

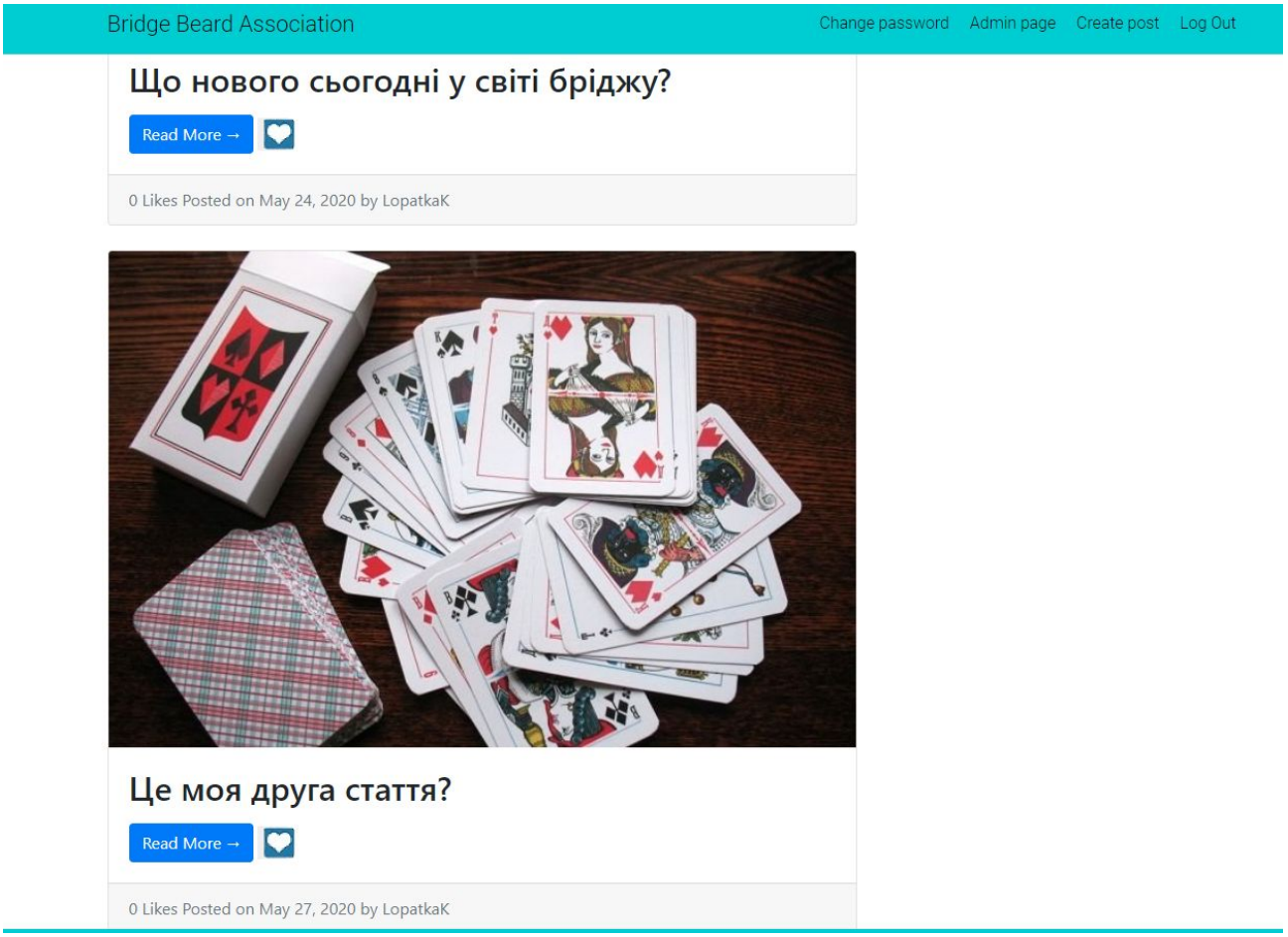

Рисунок 3.3.11 – З’явлення нового посту після його верифікації

ВИСНОВКИ

Під час виконання даної роботи було розглянуто та проаналізовано існуючих рішень, з яких було узято краще та використано як темою цієї роботи.

Було реалізовано:

1. Функції реєстрації та авторизації, які дають можливість усім бажаючим, хто прагне створювати цікаві пости.
2. Зручний інтерфейс, який не потребує великих знань в області інтернету.
3. Функцію коментування постів. Завдяки комментуванню постів автор та інші користувачі можуть обмінюватись думками.
4. Реалізована функція відновлення або зміни паролю, яка є необхідною на інформаційному порталі.
5. Функція пошуку, за допомогою якої можливо миттєво знаходити потрібну інформацію

Під час роботи було ознайомлено з різними системами управління баз даних. Завдяки цьому було розроблена и реалізована база даних.

Набув та поглибив навички працювання з фреймворком Javalin. Було повторено html, css, js.

Підводячи підсумки всієї зробленої роботи було реалізовано інформаційний портал спортивний брідж.

В роботі було використано всі навички, які були здобуті під час навчання в університеті.

СПИСОК ЛІТЕРАТУРИ

1. Javalin documentation [Електронний ресурс] – Режим доступу до ресурсу: https://javalin.io/documentation.
2. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/.
3. HTML [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/html/default.asp
4. Шилдт Г. Java 8. Полное руководство. 9-е издание / Герберт Шилдт., 2017. – (Вильямс).
5. Bridgebum [Електронний ресурс] – Режим доступу до ресурсу: https://www.bridgebum.com/.
6. Bridgeclub [Електронний ресурс] – Режим доступу до ресурсу: http://www.bridgeclub.ru/
7. Bridgewinners [Електронний ресурс] – Режим доступу до ресурсу: http://bridgewinners.com/.
8. stackoverflow [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com.
9. IntelliJ IDEA [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/ru-ru/idea/documentation/.
10. CSS [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/css/default.asp.
11. Java Script [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/js/default.asp.
12. SQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/sql/default.asp.
13. Draw.io [Електронний ресурс] – Режим доступу до ресурсу: https://app.diagrams.net/
14. Маклафлин Б. Объектно-ориентированный анализ и проектирование / Б. Маклафлин, Г. Поллайс, Д. Уэст., 2013. – (Питер). – (Head First O'Reilly).
15. Учимся проектированию Entity Relationship — диаграмм [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/440556/.

Додаток 1. Практична реалізація БД

```
Create table Users
(
      user_id serial  primary key,
      username Varchar(50) NOT NULL UNIQUE,
      email Varchar(50) NOT NULL UNIQUE,
      password Varchar(50) NOT NULL,
      first_name Varchar(50) NOT NULL,
      second_name Varchar(50) NOT NULL,
      banstatus NUMERIC(1) default 0 NOT NULL ,
      salt Char(20),
      role_id Integer NOT NULL
);


--Для Постс.
Create table Posts (
      post_id serial primary key,
      user_id Integer NOT NULL,
      post_name Varchar(50) NOT NULL,
      data_publication Timestamp NOT NULL,
      data_modif TIMESTAMP NOT NULL,
      post Varchar(255) NOT NULL,
      picture Varchar(255) NOT NULL,
      verification NUMERIC(1) default 0 NOT NULL
);

--Комменты
Create table comments (
    comment_id serial primary key,
    user_id Integer NOT NULL,
    text Varchar(255) NOT NULL,
     data_publication TIMESTAMP NOT NULL
);
```

```
--для Лайков
Create table Likes (
    like_id serial primary key,
    user_id Integer NOT NULL,
    post_id Integer,
    comment_id Integer
);
--Для форума
Create table topics (
    forum_id serial primary key,
    category_id Integer NOT NULL,
    user_id Integer NOT NULL,
    topic_name Varchar(50) NOT NULL,
     data_publication TIMESTAMP NOT NULL
)
--Для ролей
Create table Roles (
    role_id serial primary key,
    status varchar(50) NOT NULL
)
--Для куда_относиться_коммент.
Create table Comment_To_Entity (
    some_id serial primary key,
    post_id Integer,
    comment_id Integer NOT NULL,
    forum_id Integer
)
Alter table Posts add  foreign key (user_id) references Users(user_id);
Alter table Comments add  foreign key (user_id) references Users
(user_id);
Alter table Likes add  foreign key (user_id) references Users
(user_id);
Alter table Users add  foreign key (role_id) references Roles
(role_id);
```

```
Alter table Likes add  foreign key (post_id) references Posts (post_id)
ON DELETE CASCADE;
Alter table Comment_to_entity add  foreign key (post_id) references
Posts(post_id) ON DELETE CASCADE;
Alter table Likes add  foreign key (comment_id) references comments
(comment_id);
Alter table Comment_to_entity add  foreign key (comment_id) references
comments (comment_id)ON DELETE CASCADE;
```

Додаток 2. Лістинг програми коду

BridgeGameApp

```
package bridgeGameApp;

import bridgeGameApp.controller.MailSecurity;
import bridgeGameApp.controller.MainController;
import bridgeGameApp.controller.PasswordSecurity;
import bridgeGameApp.controller.SessionSecurity;
import bridgeGameApp.controller.impl.MailSecurityControllerImpl;
import bridgeGameApp.controller.impl.MainAppControllerImpl;
import bridgeGameApp.controller.impl.PasswordSecurityController;
import bridgeGameApp.controller.impl.SessionSecurityControllerImpl;
import bridgeGameApp.model.dataBase.DAOConnection;
import bridgeGameApp.model.dataBase.postgresql.PostgresqlDAO;
import bridgeGameApp.service.Notification;
import bridgeGameApp.service.NotificationImpl;
import bridgeGameApp.service.Property;
import io.javalin.Javalin;
import org.apache.log4j.Logger;

public class BridgeGameApp {
   private static Logger logger =
Logger.getLogger(BridgeGameApp.class);

   public static void main(String[] args) {
       logger.info("START app");
       Javalin app = Javalin.create();
       app.start(Property.getPort());

       app.config.addStaticFiles("pages");

       DAOConnection daoConnection = PostgresqlDAO.getInstance();
```

```
        SessionSecurity sessionSecurity = new
SessionSecurityControllerImpl();
        Notification mailNotification = new NotificationImpl();
        MailSecurity mailSecurity = new MailSecurityControllerImpl();
        PasswordSecurity passwordSecurity = new
PasswordSecurityController();
        MainController mainAppController = new
MainAppControllerImpl(app,
                daoConnection, sessionSecurity, mailNotification,
mailSecurity, passwordSecurity);
        mainAppController.startApp();
    }
}
```

BridgeControllerImpl

```
package bridgeGameApp.controller.impl;

import bridgeGameApp.controller.SessionSecurity;
import bridgeGameApp.model.dataBase.DAOConnection;
import bridgeGameApp.model.entity.User;
import io.javalin.Javalin;
import org.apache.log4j.Logger;

import java.util.List;

public class AdminControllerImpl extends BaseAbstractControllerImpl {
    private Logger logger = Logger.getLogger(AdminControllerImpl.class);
    private Javalin app;
    private DAOConnection daoConnection;
    private SessionSecurity sessionSecurity;

    public AdminControllerImpl(Javalin app, DAOConnection daoConnection,
                               SessionSecurity sessionSecurity) {
        super(sessionSecurity);
        this.app = app;
        this.daoConnection = daoConnection;
    }
```

```
    @Override
    public void execute() {
        app.post("/admin/status", context -> {
            Integer userId = getUserIdFromSession(context);
            if (userId != null) {
                logger.info("/admin/status");
                if (daoConnection.checkAdminStatus(userId)) {
                    sendResult(context, "true");
                } else {
                    sendResult(context, "false");
                }
            } else {
                sendSessionDoesNotExistResult(context);
            }
        });
        app.post("/admin/changeRole", context -> {
            Integer userId = getUserIdFromSession(context);
            if (userId != null) {
                logger.info("/admin/changeRole");
                String changedUserId = requestParam("userId", context);
                String roleId = requestParam("roleId", context);
                if
(daoConnection.setRoleForUser(Integer.parseInt(changedUserId),
                        userId, Integer.parseInt(roleId))) {
                    sendResult(context, "true");
                } else {
                    sendResult(context, 500, "false");
                }
            } else {
                sendSessionDoesNotExistResult(context);
            }
        });
        app.post("/admin/changeStatus", context -> { // add auto log out
            Integer userId = getUserIdFromSession(context);
            if (userId != null) {
```

```
                logger.info("/admin/changeStatus");
                String changedUserId = requestParam("userId", context);
                String statusId = requestParam("statusId", context);
                if
(daoConnection.setBaneStatusForUser(Integer.parseInt(changedUserId),
                        userId, Integer.parseInt(statusId))) {
                    sendResult(context, "true");
                } else {
                    sendResult(context, 500, "false");
                }
            } else {
                sendSessionDoesNotExistResult(context);
            }
        });
        app.get("/admin/adminList", context -> {
            logger.info("/admin/adminList");
            List<User> adminList = daoConnection.getAdminIdList();
            if (adminList != null && adminList.size() > 0) {
                context.json(adminList);
            } else {
                sendResult(context, 500, "false");
            }
        });
        app.post("/admin/verifyPost", context -> {
            Integer userId = getUserIdFromSession(context);
            if (userId != null) {
                logger.info("/admin/verifyPos");
                String postId = requestParam("postId", context);
                if (daoConnection.verifyPost(Integer.parseInt(postId),
userId)) {
                    sendResult(context, "true");
                } else {
                    sendResult(context, 500, "false");
                }
            } else {
                sendSessionDoesNotExistResult(context);
```

```
            }
        });
    }
}
```

BaseAbstractControllerImpl

```
package bridgeGameApp.controller.impl;

import bridgeGameApp.controller.BaseController;
import bridgeGameApp.controller.SessionSecurity;
import bridgeGameApp.service.Property;
import io.javalin.http.Context;
import org.apache.log4j.Logger;

public abstract class BaseAbstractControllerImpl implements
BaseController {
    private org.apache.log4j.Logger logger =
Logger.getLogger(BaseAbstractControllerImpl.class);
    private SessionSecurity sessionSecurity;

    public BaseAbstractControllerImpl(SessionSecurity sessionSecurity) {

        this.sessionSecurity = sessionSecurity;
    }

    public String requestParam(String paramName, Context context) {
        String value = context.formParam(paramName);
        if (value == null) {
            value = context.queryParam(paramName);
        }
        if(value != null){
            value=value.trim();
        }
        return value;
    }

    public void sendResult(Context context, int httpCode, String body) {
```

```
        context.status(httpCode);
        context.result(body);
    }

    public void sendResult(Context context, String body) {
        context.status(200);
        context.result(body);
    }

    public void sendSessionDoesNotExistResult(Context context) {
        sendResult(context, 500, "Session does not exist");
    }

    public boolean checkSessionStatus(Context context) {
        String userSessionKey =
context.sessionAttribute("userSessionKey");
        if (userSessionKey != null &&
sessionSecurity.checkValidSessionKey(userSessionKey)) {
            logger.info("checkSessionStatus " +
sessionSecurity.getUserIdFromStorage(userSessionKey) + " true");
            return true;
        } else {
            logger.info("checkSessionStatus " + userSessionKey + "
false");
            return false;
        }
    }

    public Integer getUserIdFromSession(Context context) {
        String userSessionKey =
context.sessionAttribute("userSessionKey");
        if (userSessionKey != null &&
sessionSecurity.checkValidSessionKey(userSessionKey)) {
            return sessionSecurity.getUserIdFromStorage(userSessionKey);
        } else {
```

```
            logger.info("checkSessionStatus " + userSessionKey + "
false");
            return null;
        }
    }

    public void sessionLogOut(Context context) {
        if (checkSessionStatus(context)) {
            String userSessionKey =
context.sessionAttribute("userSessionKey");

sessionSecurity.removeFromStorageUserSessionKey(userSessionKey);
            context.sessionAttribute("userSessionKey", null);
            context.result("true");
            sendResult(context, "true");
        } else {
            sendResult(context, 500, "false");
        }
    }

    public boolean validateForPassword(String password) {
        if (password != null && password.trim().length() >
Property.getLoginPasswordLengthMin()
                && password.trim().length() <
Property.getLoginPasswordLengthMax()) {
            return true;
        } else {
            return false;
        }
    }

    public boolean validateForUsername(String username) {
        if (username != null && username.trim().length() >=
Property.getLoginUsernameLengthMin()
                && username.trim().length() <=
Property.getLoginUsernameLengthMax()) {
```

```
            return true;
        } else {
            return false;
        }
    }

    public boolean validateForFistName(String firstName) {
        if (firstName != null && firstName.trim().length() >=
Property.getLoginFirstLengthMin()
                && firstName.trim().length() <=
Property.getLoginFirstLengthMax()) {
            return true;
        } else {
            return false;
        }
    }
    public boolean validateForSecondName(String secondName) {
        if (secondName != null && secondName.trim().length() >=
Property.getLoginSecondLengthMin()
                && secondName.trim().length() <=
Property.getLoginSecondLengthMax()) {
            return true;
        } else {
            return false;
        }
    }

    public boolean validateForPostTitle(String title) {
        if (title != null && title.trim().length() <=
Property.getPostNameMax()) {
            return true;
        } else {
            return false;
        }
    }
```

```
    public boolean validateForPostText(String secondName) {
        if (secondName != null && secondName.trim().length() <=
Property.getPostTextMax()) {
            return true;
        } else {
            return false;
        }
    }

    public boolean validateForCategoryText(String name) {
        if (name != null && name.trim().length() <=
Property.getCategoryTextMax()) {
            return true;
        } else {
            return false;
        }
    }

    public boolean validateForForumText(String text) {
        if (text != null && text.trim().length() <=
Property.getTopicTextLengthMax()) {
            return true;
        } else {
            return false;
        }
    }

    public boolean validateForComment(String text) {
        if (text != null && text.trim().length() <=
Property.getCommentLengthMax()) {
            return true;
        } else {
            return false;
        }
    }
}
```

MailControllerImpl

```
package bridgeGameApp.controller.impl;

import bridgeGameApp.controller.MailSecurity;
import bridgeGameApp.service.Property;

import java.time.LocalDateTime;
import java.util.concurrent.ConcurrentHashMap;

public class MailSecurityControllerImpl implements MailSecurity,
Runnable {
   private static ConcurrentHashMap<String, Integer> mailStorage = new
ConcurrentHashMap();
   private static ConcurrentHashMap<String, Integer>
mailCountMessageStorage = new ConcurrentHashMap();

   @Override
   public Integer getCodeForVerify(String email) {
       if (mailStorage.containsKey(email)) {
           return mailStorage.get(email);
       }
       return null;
   }

   @Override
   public Integer getNewCodeForVerify(String email) {
       removeFromStorage(email);
       int key = generateKey();
       mailStorage.put(email, key);
       return key;
   }

   @Override
   public boolean removeFromStorage(String email) {
       if (mailStorage.containsKey(email)) {
           mailStorage.remove(email);
```

```
            return true;
        }
        return false;
    }

    private int generateKey() {
        int random = 0;
        do {
            random = (int) (Math.random() * Integer.MAX_VALUE / 10000) +
10000;
        } while (mailStorage.containsValue(random));
        return random;
    }

    @Override
    public boolean checkAddContEmailAvailable(String email) {
        boolean flag = true;
        int value = 0;
        int COUNT_EMAIL_IN_DAY = Property.getCountEmailInDay();
        if (mailCountMessageStorage.containsKey(email)) {
            value = mailStorage.get(email);
            if (value > COUNT_EMAIL_IN_DAY) {
                flag = false;
            }
        }
        mailCountMessageStorage.put(email, ++value);

        return flag;
    }

    @Override
    public void run() {
        LocalDateTime lastTime = LocalDateTime.now();
        int TIME_INTERVAL_FOR_CHECKING_MAIL_IN_HOURS =
Property.getTimeIntervalForEmailCountCheck();
        while (true) {
```

```
            if
(lastTime.plusHours(TIME_INTERVAL_FOR_CHECKING_MAIL_IN_HOURS).isBefore(
LocalDateTime.now())) {
                lastTime = LocalDateTime.now();
                mailCountMessageStorage = new ConcurrentHashMap();
                TIME_INTERVAL_FOR_CHECKING_MAIL_IN_HOURS =
Property.getTimeIntervalForEmailCountCheck();
            }
        }
    }
}
```

MainAppController

```
package bridgeGameApp.controller.impl;

import bridgeGameApp.controller.*;
import bridgeGameApp.model.dataBase.DAOConnection;
import bridgeGameApp.service.Notification;
import io.javalin.Javalin;
import org.apache.log4j.Logger;

import java.util.ArrayList;
import java.util.List;


public class MainAppControllerImpl implements MainController {
    private Logger logger = Logger.getLogger(MainController.class);
    private Javalin app;
    private DAOConnection daoConnection;
    private List<BaseController> controllerInterfaceList = new
ArrayList<>();

    private SessionSecurity sessionSecurity;
    private Notification mailNotification;
    private MailSecurity mailSecurity;
    private PasswordSecurity passwordSecurity;
```

```
    public MainAppControllerImpl(Javalin javalin, DAOConnection
daoConnection, SessionSecurity sessionSecurity,
                                 Notification mailNotification,
MailSecurity mailSecurity,
                                 PasswordSecurity passwordSecurity) {
        this.app = javalin;
        this.daoConnection = daoConnection;
        this.sessionSecurity = sessionSecurity;
        this.mailNotification = mailNotification;
        this.mailSecurity = mailSecurity;
        this.passwordSecurity = passwordSecurity;
    }

    @Override
    public void startApp() {
        app.get("/", context -> {
            logger.info("app.get '/', pages/Authorization.html");
            context.render("pages/Authorization.html");
        });
        app.error(404, ctx -> {
            logger.info("page not found");
            ctx.render("pages/404.html");

        });
          try {
            createControllerList();
            checkSessionTiming();
            checkEmailCount();
            executeControllerList();
        } catch (Exception ex) {
            logger.error(ex.getStackTrace());
        }
    }

    private void createControllerList() {
```

```
        controllerInterfaceList.add(new UserControllerImpl(app,
daoConnection, sessionSecurity,
                mailNotification, mailSecurity, passwordSecurity));
        controllerInterfaceList.add(new AdminControllerImpl(app,
daoConnection, sessionSecurity));
        controllerInterfaceList.add(new PostControllerImpl(app,
daoConnection, sessionSecurity));
        controllerInterfaceList.add(new CommentControllerImpl(app,
daoConnection, sessionSecurity));
        controllerInterfaceList.add(new CategoryControllerImpl(app,
daoConnection, sessionSecurity));
        controllerInterfaceList.add(new ForumControllerImpl(app,
daoConnection, sessionSecurity));
    }

    private void executeControllerList() {
        for (BaseController controllerInterface :
controllerInterfaceList) {
            controllerInterface.execute();
        }
    }

    private void checkSessionTiming() {
        Thread thread = new Thread((Runnable) sessionSecurity);
        thread.start();
    }

    private void checkEmailCount() {
        Thread thread = new Thread((Runnable) mailSecurity);
        thread.start();
    }
}
```

PaswordSecurityController

```
package bridgeGameApp.controller.impl;

import bridgeGameApp.controller.PasswordSecurity;
```

```
import bridgeGameApp.service.Property;

import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.KeySpec;
import java.util.Base64;
import java.util.Random;

public class PasswordSecurityController implements PasswordSecurity {
   @Override
   public String getEncryptedPassword(String password, String salt)
           throws NoSuchAlgorithmException, InvalidKeySpecException {
       String algorithm = "PBKDF2WithHmacSHA1";
       int derivedKeyLength = 160; // for SHA1
       int iterations = 20000; // NIST specifies 10000

       byte[] saltBytes = Base64.getDecoder().decode(salt.trim());
       KeySpec spec = new PBEKeySpec(password.toCharArray(), saltBytes,
iterations, derivedKeyLength);
       SecretKeyFactory f = SecretKeyFactory.getInstance(algorithm);

       byte[] encBytes = f.generateSecret(spec).getEncoded();
       return Base64.getEncoder().encodeToString(encBytes);
   }

   @Override
   public String getNewSalt() throws NoSuchAlgorithmException {
       SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
       byte[] salt = new byte[8];
       random.nextBytes(salt);
       return Base64.getEncoder().encodeToString(salt);
   }
```

```
    @Override
    public String generateNewPassword() {
        int leftLimit = 48; // numeral '0'
        int rightLimit = 122; // letter 'z'
        int targetStringLength = Property.getLoginPasswordLengthMax() -
1;
        Random random = new Random();

        String generatedString = random.ints(leftLimit, rightLimit + 1)
                .filter(i -> (i <= 57 || i >= 65) && (i <= 90 || i >=
97))
                .limit(targetStringLength)
                .collect(StringBuilder::new,
StringBuilder::appendCodePoint, StringBuilder::append)
                .toString();
        return generatedString;
    }
}
```

PostControllerImpl

```
package bridgeGameApp.controller.impl;

import bridgeGameApp.controller.SessionSecurity;
import bridgeGameApp.model.dataBase.DAOConnection;
import bridgeGameApp.model.entity.Comment;
import bridgeGameApp.model.entity.Post;
import bridgeGameApp.service.Property;
import io.javalin.Javalin;
import io.javalin.http.Context;
import org.apache.log4j.Logger;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.time.LocalDateTime;
```

```
import java.util.List;

public class PostControllerImpl extends BaseAbstractControllerImpl {
   private Logger logger = Logger.getLogger(PostControllerImpl.class);
   private Javalin app;
   private DAOConnection daoConnection;

   public PostControllerImpl(Javalin app, DAOConnection daoConnection,
                             SessionSecurity sessionSecurity) {
       super(sessionSecurity);
       this.app = app;
       this.daoConnection = daoConnection;
   }

   @Override
   public void execute() {
       app.get("/post/postTitles", context -> {
           List<Post> postList = daoConnection.getVerPostListTitles();
           if (postList != null && postList.size() > 0) {
               context.json(postList);
           } else {
               sendResult(context, 500, "false");
           }
       });
       app.get("/post/postInfo", context -> {
           String id = requestParam("postId", context);
           Post postList =
daoConnection.getVerPost(Integer.parseInt(id));
           if (postList != null) {
               context.json(postList);
           } else {
               sendResult(context, 500, "false");
           }
       });

       app.get("/post/commentList", context -> {
```

```
            String id = requestParam("postId", context);
            List<Comment> commentList =
daoConnection.getCommentListForPost(Integer.parseInt(id));
            if (commentList != null && commentList.size() > 0) {
                context.json(commentList);
            } else {
                sendResult(context, 500, "false");
            }
        });
        app.get("/post/notVerify/admin", context -> {
            Integer userId = getUserIdFromSession(context);
            if (userId != null) {
                List<Post> postListForAdmin =
daoConnection.getNotVerificationPostsForAdmin(userId);
                if (postListForAdmin != null && postListForAdmin.size()
> 0) {
                    context.json(postListForAdmin);
                } else {
                    sendResult(context, 500, "false");
                }
            } else {
                sendSessionDoesNotExistResult(context);
            }
        });
        app.get("/post/notVerify/user", context -> {
            Integer userId = getUserIdFromSession(context);
            if (userId != null) {
                List<Post> postListForAdmin =
daoConnection.getNotVerificationPosForUser(userId);
                if (postListForAdmin != null && postListForAdmin.size()
> 0) {
                    context.json(postListForAdmin);
                } else {
                    sendResult(context, 500, "false");
                }
            } else {
```

```
                sendSessionDoesNotExistResult(context);
            }
        });

        app.post("/post/create", context -> {
            Integer userId = getUserIdFromSession(context);
            if (userId != null) {
                String title = requestParam("title", context);
                String text = requestParam("post", context);
              //  String picture = requestParam("picture", context);
                String path = imageLoad(context); // check
                if (validateForPostTitle(title) &&
validateForPostText(text)) {
                    Post post = new Post(userId, title, text, path);
                    Integer postId = daoConnection.addPost(post);
                    if (postId != null) {
                        sendResult(context, String.valueOf(postId));
                    } else {
                        sendResult(context, 500, "false");
                    }
                } else {
                    logger.error("Input params error");
                    sendResult(context, 500, "Input params error");
                }
            } else {
                sendSessionDoesNotExistResult(context);
            }
        });
        app.post("/post/update", context -> {
            Integer userId = getUserIdFromSession(context);
            if (userId != null) {
                String postId = requestParam("postId", context);
                String title = requestParam("title", context);
                String text = requestParam("post", context);
                String picture = requestParam("picture", context);
```

```
                if (validateForPostTitle(title) &&
validateForPostText(text)) {
                    Post post = new Post(Integer.parseInt(postId),
userId, title, text, picture);
                    if (daoConnection.updatePost(post)) {
                        sendResult(context, "true");
                    } else {
                        sendResult(context, 500, "false");
                    }
                } else {
                    logger.error("Input params error");
                    sendResult(context, 500, "Input params error");
                }
            } else {
                sendSessionDoesNotExistResult(context);
            }
        });
        app.post("/post/delete", context -> {
            Integer userId = getUserIdFromSession(context);
            if (userId != null) {
                String postId = requestParam("postId", context);
                if (daoConnection.removePost(Integer.parseInt(postId),
userId)) {
                    sendResult(context, "true");
                } else {
                    sendResult(context, 500, "false");
                }
            } else {
                sendSessionDoesNotExistResult(context);
            }
        });
        app.get("/post/find/author", context -> {
            String userId = requestParam("userId", context);
            List<Post> postList =
daoConnection.getPostListByAuthor(Integer.parseInt(userId));
            if (postList != null && postList.size() > 0) {
```

```
                context.json(postList);
            } else {
                sendResult(context, 500, "false");
            }
        });
        app.get("/post/find/name", context -> {
            String name = requestParam("name", context);
            List<Post> postList = daoConnection.getPostByName(name);
            if (postList != null && postList.size() > 0) {
                context.json(postList);
            } else {
                sendResult(context, 500, "false");
            }
        });
        app.get("/post/find/sort/publicationDate", context -> {
            List<Post> postList = daoConnection.getPostListByDate();
            if (postList != null && postList.size() > 0) {
                context.json(postList);
            } else {
                sendResult(context, 500, "false");
            }
        });
        app.post("/post/like/add", context -> {
            Integer userId = getUserIdFromSession(context);
            if (userId != null) {
                String postId = requestParam("postId", context);
                if (daoConnection.addPostLike(Integer.parseInt(postId),
userId)) {
                    sendResult(context, "true");
                } else {
                    sendResult(context, 500, "false");
                }
            } else {
                sendSessionDoesNotExistResult(context);
            }
        });
```

```
    }

    private String imageLoad(Context ctx) {
        InputStream inputStream =
ctx.uploadedFile("picture").getContent();
        String name = ctx.uploadedFile("picture").getFilename();
        name = LocalDateTime.now().toString().
                replace("-", "").
                replace(":", "") + name;

        File file = new File(Property.getPath() // does not work for jar
file
                + File.separator
                + "target" + File.separator
                + "classes" + File.separator
                + "pages" + File.separator
                + "images" + File.separator
                + name);
        logger.info(Property.getPath()
                + File.separator
                + "target" + File.separator
                + "classes" + File.separator
                + "pages" + File.separator
                + "images" + File.separator
                + name);
        BufferedImage imBuff = null;
        try {
            imBuff = ImageIO.read(inputStream);
            ImageIO.write(imBuff, "jpg", file);
            ImageIO.write(imBuff, "png", file);
            return name;
        } catch (IOException e) {
            logger.info(e);
        }
        return null;
    }
```

```
}
```

UserControllerImpl

```
package bridgeGameApp.controller.impl;

import bridgeGameApp.controller.MailSecurity;
import bridgeGameApp.controller.PasswordSecurity;
import bridgeGameApp.controller.SessionSecurity;
import bridgeGameApp.model.dataBase.DAOConnection;
import bridgeGameApp.model.entity.User;
import bridgeGameApp.service.Notification;
import bridgeGameApp.service.Property;
import io.javalin.Javalin;
import org.apache.log4j.Logger;

import java.security.GeneralSecurityException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;

public class UserControllerImpl extends BaseAbstractControllerImpl {
   private Logger logger = Logger.getLogger(UserControllerImpl.class);
   private Javalin app;
   private DAOConnection daoConnection;
   private SessionSecurity sessionSecurity;
   private Notification mailNotification;
   private MailSecurity mailSecurity;
   private PasswordSecurity passwordSecurity;

   public UserControllerImpl(Javalin app, DAOConnection daoConnection,
SessionSecurity sessionSecurity,
                             Notification mailNotification,
MailSecurity mailSecurity,
                             PasswordSecurity passwordSecurity) {
       super(sessionSecurity);
       this.app = app;
       this.daoConnection = daoConnection;
       this.sessionSecurity = sessionSecurity;
```

```
        this.mailNotification = mailNotification;
        this.mailSecurity = mailSecurity;
        this.passwordSecurity = passwordSecurity;
    }

    @Override
    public void execute() {
        app.post("/user/login", context -> {
            if (checkSessionStatus(context)) {
                sendResult(context, 500, "User has been logged in");
                return;
            }
            String username = requestParam("username", context);
            String password = requestParam("password", context);
            logger.info("post /user/login username" + username);
            if (validateForUsername(username) &&
validateForPassword(password)) {
                try {
                    String salt = daoConnection.getUserSalt(username);
                    String calcHash =
passwordSecurity.getEncryptedPassword(password, salt);
                    User user = daoConnection.getUser(username,
calcHash);
                    if (user != null) {
                        if
(daoConnection.checkBanStatusForUser(user.getUserId())) {
                            context.sessionAttribute("userSessionKey",

sessionSecurity.putToStorageUserSessionKey(user.getUserId()));
                            logger.info("user found");
                            sendResult(context,
String.valueOf(user.getUserId()));
                        } else {
                            logger.info("User id " + user.getUserId() +
" and email "
```

```
                                    + user.getUserId() + " has been
banned");
                            sendResult(context, 500, "User with email "
                                    + user.getEmail() + " has been
banned by admin");
                        }
                    } else {
                        throw new NullPointerException("User user is
null");
                    }
                } catch (GeneralSecurityException | NullPointerException
e) {
                    logger.error(e);
                    logger.info("User not found. Incorrect username or
password");
                    sendResult(context, 500, "User not found. Incorrect
username or password"
                            + "\nPlease, try one more time");
                }
            } else {
                sendResult(context, 500, "Incorrect length of username
or password");
        }
        });
        app.post("/user/logout", context -> {
            sessionLogOut(context);
        });
        app.get("/user/currentUser", context -> {
            Integer id = getUserIdFromSession(context);
            if (id != null) {
                logger.info("/user/currentUse " + id);
                User user = daoConnection.getUser(id);
                if (user != null) {
                    logger.info("user " + user.toString());
                    context.json(user);
                }
```

```
            } else {
                logger.info("User not found");
                sendResult(context, 500, "User not found");
            }
        });
        app.get("/user/userInfo", context -> {
            String userId = requestParam("id", context);
            logger.info("/user/userInfo " + userId);
            User user = daoConnection.getUser(Integer.parseInt(userId));
            if (user != null) {
                logger.info("user " + user.toString());
                context.json(user);
            } else {
                logger.info("User not found");
                sendResult(context, 500, "User not found");
            }
        });
        app.get("/session/status", context -> {
            if (checkSessionStatus(context)) {
                sendResult(context, "true");
            } else {
                sendResult(context, "false");
            }
        });
        app.post("/user/username", context -> {
            logger.info("/user/username");
            String username = requestParam("username", context);
            logger.info("username " + username);
            boolean statusUsername =
daoConnection.checkNotExistUserName(username);
            logger.info(statusUsername);
            sendResult(context, String.valueOf(statusUsername));
        });
        app.post("/user/email", context -> {
            logger.info("/user/email");
            String email = requestParam("email", context);
```

```
            logger.info("email " + email);
            boolean statusEmail =
daoConnection.checkNotExistEmail(email);
            logger.info(statusEmail);
            sendResult(context, String.valueOf(statusEmail));
        });
        app.post("/user/sendVerifyMail", context -> {
            logger.info("/user/sendVerifyMail");
            String email = requestParam("email", context);
            String subject = Property.getEmailSubject();
            String text = Property.getEmailTextRegistration();
            if (mailSecurity.checkAddContEmailAvailable(email)) {
                Integer code = mailSecurity.getNewCodeForVerify(email);
                boolean result =
mailNotification.sendNotification(email, subject, text + code);
                logger.info("email " + email + " result " + result);
                sendResult(context, String.valueOf(result));
            } else {
                sendResult(context, 500, "Exceeded message limit for
this email");
            }
        });
        app.post("/user/code", context -> {
            String email = requestParam("email", context);
            String stringCode = requestParam("code", context);
            int code = Integer.parseInt(stringCode);
            if (mailSecurity.getCodeForVerify(email) != null &&
                    mailSecurity.getCodeForVerify(email).equals(code)) {
                sendResult(context, "true");
            } else {
                sendResult(context, 500, "false");
            }
        });
        app.post("/user/create", context -> {
            logger.info("/user/create");
            String username = requestParam("username", context);
```

```
            String password = requestParam("password", context);
            String email = requestParam("email", context);
            String firstName = requestParam("firstName", context);
            String secondName = requestParam("secondName", context);
            String stringCode = requestParam("code", context);
            logger.info("username " + username + " email " + email + "
code " + stringCode
                    + "firstName " + firstName + "secondName " +
secondName);
            if (stringCode.trim().length() > 1) {
                int code = Integer.parseInt(stringCode);
                if (validateForUsername(username) &&
validateForPassword(password)
                        && validateForFistName(firstName) &&
validateForSecondName(secondName)
                        && mailSecurity.getCodeForVerify(email) != null
&&

mailSecurity.getCodeForVerify(email).equals(code)) {
                    String salt = passwordSecurity.getNewSalt();
                    password =
passwordSecurity.getEncryptedPassword(password, salt);
                    Integer userId = daoConnection.addUser(new
User(username, email, firstName,
                            secondName, password, salt));
                    if (userId != null) {
                        User user = daoConnection.getUser(userId);
                        context.sessionAttribute("userSessionActiveKey",

sessionSecurity.putToStorageUserSessionKey(user.getUserId()));
                        logger.info("user found id " +
user.getUserId());
                        logger.info("remove from storage " +
mailSecurity.removeFromStorage(email));
                        sendResult(context, String.valueOf(userId));
                    } else {
```

```
                    logger.error("/user/create error");
                    sendResult(context, 500, "false");
                }
            } else {
                logger.error("Input params error");
                sendResult(context, 500, "Input params error");
            }
        } else {
            logger.error("/user/create error");
            sendResult(context, 500, "Code error");
        }
    });
    app.get("/user/restorePassword", context -> {
        logger.info("/user/restorePassword");
        String email = requestParam("email", context);
        String password = passwordSecurity.generateNewPassword();
        if (mailSecurity.checkAddContEmailAvailable(email)) {
            if (updatePassword(email, password)) {
                String subject = Property.getEmailSubject();
                String text = Property.getEmailRestoreText();
                if (mailNotification.sendNotification(email,
subject, text + password)) {
                    sendResult(context, "true");
                } else {
                    sendResult(context, 500, "false");
                }
            } else {
                sendResult(context, 500, "Email not found");
            }
        } else {
            sendResult(context, 500, "Exceeded message limit for
this email");
        }
    });
    app.post("/user/updatePassword", context -> {
        Integer userId = getUserIdFromSession(context);
```

```
            if (userId != null) {
                logger.info("/user/restorePassword");
                String password = requestParam("password", context);
                if (validateForPassword(password)) {
                    if (updatePassword(userId, password)) {
                        sendResult(context, "true");
                    } else {
                        sendResult(context, 500, "Error. Unable to
update password");
                    }
                } else {
                    sendResult(context, 500, "Incorrect length of
password");
                }
            } else {
                sendSessionDoesNotExistResult(context);
            }
        });
    }

    private boolean updatePassword(String email, String password)
            throws NoSuchAlgorithmException, InvalidKeySpecException {
        String salt = passwordSecurity.getNewSalt();
        password = passwordSecurity.getEncryptedPassword(password,
salt);
        return daoConnection.updatePasswordForUser(email, password,
salt);
    }

    private boolean updatePassword(int userId, String password)
        throws NoSuchAlgorithmException, InvalidKeySpecException {
        String salt = passwordSecurity.getNewSalt();
        password = passwordSecurity.getEncryptedPassword(password,
salt);
        return daoConnection.updatePasswordForUser(userId, password,
salt);}}
```