Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук

Кафедра програмної інженерії

КУРСОВА РОБОТА

ПОЯСНЮВАЛЬНА ЗАПИСКА

з навчальної дисципліни «Аналіз та рефакторинг коду програмного забезпечення»

Тема роботи: «Сервіс для обміну речей “Plyushkin”»

Студент гр. ПІ-15-2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Пітюкова М.О.

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_ доц. Лещинська І.О.

Роботу захищено «\_\_»\_\_\_\_\_\_2017 р.

з оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Комісія:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ доц. Лещинський В.О.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ доц. Лещинська І.О.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ доц. Сокорчук І.П.

Харків

2017 р.

Харківський національний університет радіоелектроніки

Факультет *комп’ютерних наук* Кафедра *програмної інженерії*

Напрям підготовки *6.050103 – Програмна інженерія*

Курс 3Семестр 5

Навчальна дисципліна *Аналіз та рефакторинг коду програмного забезпечення*

ЗАВДАННЯ

НА КУРСОВУ РОБОТУ СТУДЕНТОВІ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*Пітюкової Марії Олегівни*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Тема роботи: *«Сервіс для обміну речей “Plyushkin”»*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Термін узгодження завдання курсової роботи « 10 » вересня 2017 р.

3. Термін здачі студентом закінченої роботи « 26 » грудня 2017 р.

4. Вихідні дані до проекту (роботи): *методичні вказівки до виконання курсової роботи, вимоги до інформаційної системи, постановка задачі, предметна область*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Зміст пояснювальної записки (перелік питань, що належить розробити) *вступ, аналіз предметної області і постановка задачі, моделювання програмного продукту, опис реалізації програмного продукту, опис інтерфейсу та функціоналу, висновки, перелік посилань, додатки*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) *діаграма варіантів використання, діаграма класів, діаграма станів, діаграма компонентів, діаграма розгортання, діаграма послідовностей, діаграма кооперацій, діаграма активності, архітектура проекту, схема бази даних, інтерфейс веб-додатку, інтерфейс мобільного додатку*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів курсової роботи | Термін виконання етапів роботи | Примітки |
| 1 | Функціональна специфікація програмного проекту | 10. 09. 2017 | Виконано |
| 2 | Проектування програмного проекту | 20.09.2017 | Виконано |
| 3 | Кодування програмного проекту | 20.10.2017 | Виконано |
| 4 | Оформлення пояснювальної записки | 10.12.2017 | Виконано |
| 5 | Захист курсової роботи | 30.12.2017 | Виконано |

Дата видачі завдання « 10 » вересня 2017 р.

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ доц. Лещинська І.О.

Завдання прийняв до виконання

ст.гр. ПІ-15-2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Пітюкова М.О.

РЕФЕРАТ

Пояснювальна записка до курсової роботи: 44 с., 18 рис., 2 додатки, 10 джерел.

СЕРВІС, РЕЧІ, API.NET WEB API, ANGULAR, MICROSOFT SQL SERVER, ANDROID, JAVA, BOOTSTRAP.

Об’єктом проектування є сервіс, що займається обміном різних за категоріями речами.

Метою роботи є проектування та розробка сервісу, який допомагає обмінюватися речами.

Методи розробки базуються на об’єктно-орієнтованому підході до розробки програмного забезпечення, мовах програмування C# та Java, базі даних Microsoft SQL Server, Angular та ASP.NET WEB API.

У результаті роботи здійснено програмну реалізацію сервіса, що складається з Web – клієнту, мобільного клієнту та серверу, за допомогою якого можна здійснювати обмін будь-якої речі.

ЗМІСТ

Вступ …………………………………………………………………………….…. 6

1 Аналіз та постановка задачі …………………………………………………….. 7

2 Моделювання програмного продукту ………………………………………….. 9

3 Реалізація програмного продукту ……………………...…………….………... 14

3.1 Загальні відомості про систему …………………………………...... 14

3.2 Сервер бізнес логіки ……………………………………………….... 15

3.3 Веб-сайт …………………………………………………………….... 15

3.4 Мобільний клієнт …………………………………………………… 15

4 Інтерфейс і функціонал ………………………………………………………… 16

4.1 Веб-сайт ……………………………………………………………… 16

4.2 Мобільний додаток …………………………………………………. 19

Висновки ………………………………………………………………………...... 22

Перелік посилань …………….…………………………...………………………. 23

Додаток А Vision and Scope Document ………………………………………….. 29

Додаток Б Фрагменти вихідного коду системи ..……………………………….. 33

## ВСТУП

У наш час більшість людей мають речі, які з різних причин, стали не потрібними у використанні. Виникають питання - що з ними робити. Залишити припадати пилом на полицях, але добре, якщо є вільне місце, а якщо ні? Можна продати? Виставив фотографію, почекав і продав. Хоч за третину вартості, а все одно гроші. Тут є проблема, можна назвати її психологічної або чисто людської: коли ми купуємо речі, ми чіпляємося до них, все здається занадто дорогим і не дуже хорошим. Тому оголошення про продаж можуть довго висіти на сайті, і жодна людина до вас не звернеться. Коли ж ви отримуєте щось безкоштовно, все виглядає зовсім по-іншому: якщо річ потрібна, чому б її не взяти Коли ви просто так віддаєте корисну річ незнайомцю, в цьому є якась магія добра.

Тут і приходить на допомогу сервіс обміну речей. За допомогою сервісу можна або донести ідею еко-споживання в кожну родину: чим менше купується, тим менше виробляється і використовується ресурсів нашої планети; або допомагати людям економити сімейний бюджет (багато статей затрат можна зменшити шляхом обміну); або давати друге життя речам, що не використовуються — згадайте, де зараз ваше випускне плаття чи книжка, яку прочитали місяць тому. Цей сервіс призначений для різних людей і дає колосальні можливості для обміну.

Метою даної курсової роботи є проектування, розробка та тестування програмного проекту, який допоможе вирішити проблеми з обміном речей та його автоматизацією; самостійне вивчення принципів написання чистого коду; основ рефакторингу та реінжинірингу коду; закріплення теоретичних знань та практичних навичок, отриманих в ході вивчення даної̈ навчальної̈ дисципліни; а також отримання практичних навичок проектування та розробки додатку.

1 АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ

Кожна доросла людина хоча б раз у житті стикалась з проблемою обміну як такого. Відразу пригадуються втрачений час, загублені нерви і, найчастіше, нульовий результат. Тому актуальним є створення такої системи, яка дозволила б вирішити цю проблему.

Тобто, необхідна система, яку можна використовувати для миттєвого обміну між двома користувачами. Крім цього, система повинна сортувати речі за категоріями, що дозволить користувачам більш раціонально розподіляти свій час на пошуки необхідного.

Метою є створення конкурентоспроможної системи класифікації речей, що дозволить в подальшому здійснювати обмін ними між користувачами.

Потреби користувача включають:

1. додавання речі;
2. можливість переглянути речі деяких категорій;
3. можливість здійснити обмін.

У подальшому проект може бути розширеним та модифікованим згідно зі збільшенням потреб користувачів.

Архітектура системи спроектована для досягнення легкості масштабування [1] .

Основні функції:

1. додавання речі;
2. додавання категорії;
3. проведення обміну між користувачами;
4. обробка даних користувачів;

Такий набір параметрів є достатнім для використання як зі сторони користувача, так і зі сторони адміністратора.

Для більш зручного використання даної системи слід створити додаток, що матиме сервер, веб-клієнт та мобільний клієнт. Це дозволить швидко та зручно користуватися речами. Система повинна мати швидкий доступ до бази, надавати можливість зручного доступу до інформації, та надавати можливість користуватися системою та моніторити данні по категоріям.

Програмний продукт має включати базу даних, яка зберігає данні про речі, категорії та користувачів, сервер, що обробляє запит та віддає відгук.

Сервер повинен реалізовувати наступні функції:

* 1. надавати інформацію про речі користувача;
  2. додавання речі;
  3. обробляти запит від клієнта;
  4. повертати результати до клієнта;
  5. виступати посередником між сторонніми API та клієнтським додатком.

Сервер повинен бути реалізований за допомогою наступних технологій:

1. ASP.NET Web API;
2. MSSQL.

В якості веб-сервера для віддачі статичних даних використовуватиметься веб-сервер на базі Angular.

Для того, щоб зробити систему розподіленою використовуватиметься REST протокол, основною задачею якого буде прийняття на себе необхідної логіки, дозволяючи таким чином отримати уніфікований доступ до логіки, і від веб-версії сервісу, і від мобільного клієнту.

Система повинна взаємодіяти з СУБД MSSQL, яка в свою чергу повинна зберігати та видавати за необхідністю [2]:

1. основні дані користувача (логін, ім'я, адресу електронної пошти, пароль);
2. список речей користувача;
3. список категорій;
4. список всіх речей в категоріях;

Веб-сайт повинен бути реалізований за допомогою технологій:

1. HTML;
2. CSS;
3. JavaScript, Angular;
4. Bootstrap.

На веб-сайт буде розподілятися наступне навантаження:

1. реєстрація нових користувачів;
2. авторизація користувачів;
3. додавання речей та іх відображення;
4. додавання категорій та іх відображення;
5. здійснення обміну.

Мобільний додаток повинен бути реалізований за допомогою технологій:

1. Java;
2. JSON.

Мобільний додаток повинен виконувати наступний функціонал:

1. авторизація користувачів;
2. отримання речей користувача.

2 МОДЕЛЮВАННЯ ПРОГРАМНОГО ПРОДУКТУ

Моделювання програмного продукту було проведено з використанням діаграм UML. Кожна діаграма відображує систему з конкретної сторони, і у сукупності це дає змогу отримати потрібну інформацію для розробки системи (архітектура, робота, функціонал).

Діаграма варіантів використання (Use Case) зображена на рисунку 2.1. Діаграма зображує можливі варіанти використання системи.

Акторами є адміністратор, користувач. Користувач, який додав категорію стає адміністратором. Користувач може керувати своїми речами. Адміністратор наслідує користувача, тобто йому доступні також і усі функції, які доступні для звичайного користувача.

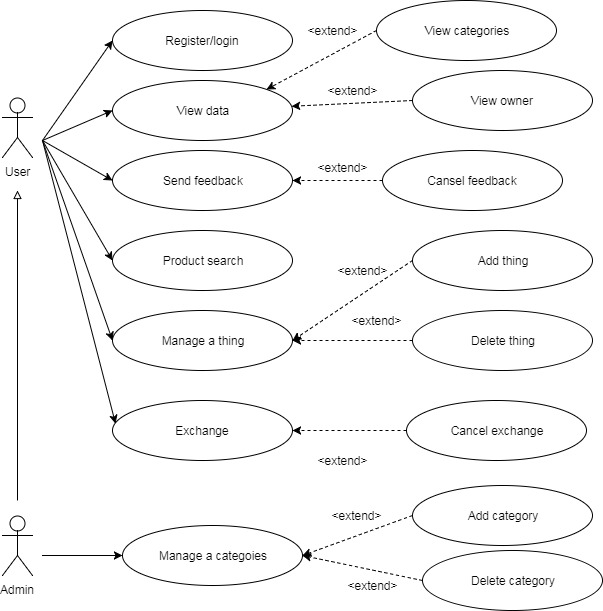


Рисунок 2.1 – Діаграма варіантів використання

Продемонструвати процеси переходів між станами системи можна за допомогою діаграми станів (див. рис. 2.2), на якій в даному випадку зображені основні переходи в веб-версії системи. Діаграма станів для мобільного додатку в даному випадку не розглядається.

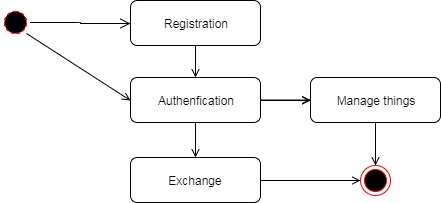


Рисунок 2.2 – Діаграма станів

На рисунку 2.3 зображена діаграма класів. Вона створена для демонстрації відношення статичних та декларативних елементів моделі. Діаграми класів можуть застосовуватися і при прямому проектуванні, тобто в процесі розробки нової системи, і при зворотному проектуванні. У діаграмі класів безпосередньо відображається вихідний код програми, таким чином, діаграма класів є кінцевим результатом проектування і відправною точкою процесу розробки.

На даній діаграмі відображені основні моделі, які були використані при розробці програмної системи.

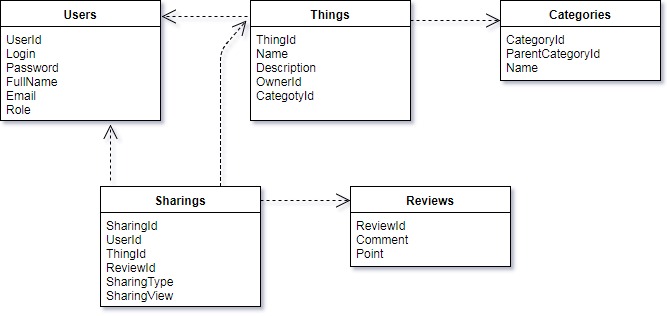


Рисунок 2.3 – Діаграма класів

На рисунку 2.4 зображена діаграма компонентів. Діаграма компонентів відображає залежності між компонентами програмного забезпечення, включаючи компоненти вихідних кодів, бінарні компоненти, та компоненти, що можуть виконуватись. Компоненти об'єднуються, разом використовуючи структурні зв'язки, щоб об'єднати інтерфейси двох компонент. Це ілюструє зв'язок типу «клієнт-сервер».

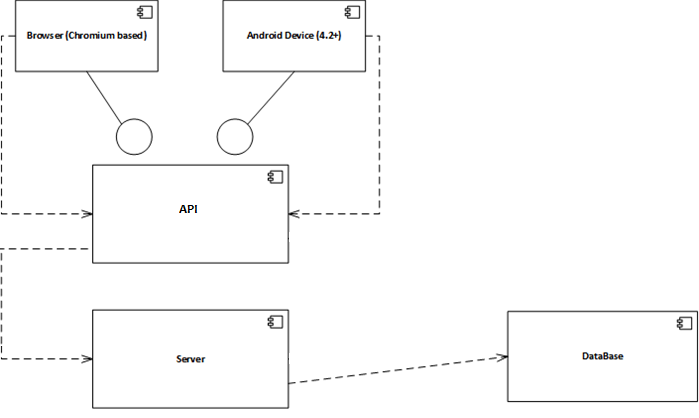


Рисунок 2.4 – Діаграма компонентів

Згідно діаграми, в системі є компонент бази даних, серверна частина додатку, API, клієнтська частина додатку та мобільний клієнт.

Треба відзначити, що на діаграмі показані усі компоненти системі та зв’язки між ними. Клієнтська частина додатку та мобільний клієнт взаємодіють з API, який обробляє дані та відправляють дані до сервера, а він в свою чергу взаємодіє з базою даних.

Відповідно до діаграми компонентів побудована діаграма розгортання (див. рис. 2.5). На діаграмі розгортання відображаються обчислювальні вузли під час роботи програми, компоненти, та об'єкти, що виконуються на цих вузлах. Компоненти відповідають представленню робочих екземплярів одиниць коду. Компоненти, що не мають представлення під час роботи програми на таких діаграмах не відображаються. Діаграма розгортання відображає робочі екземпляри компонент.

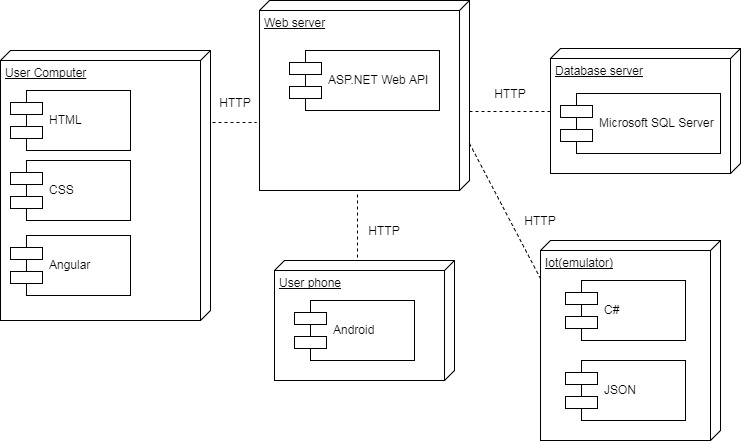


Рисунок 2.5 – Діаграма розгортання

На діаграмі розгортання зображено місцезнаходження окремих компонентів системи: на сервері додатку, сервері бази даних та на мобільному додатку.

Діаграма послідовностей представляє собою систему в цілому та те, як відбувається взаємодія між об’єктами на різних рівнях. Відображає взаємодії об'єктів впорядкованих за часом. Зокрема, такі діаграми відображають задіяні об'єкти та послідовність відправлених повідомлень На даній діаграмі послідовності (див. рис. 2.6) зображено послідовність дій при реєстрації нового користувача.

Користувач вводить дані для реєстрації чи входу, клієнт їх обробляє та передає на сервер, який у свою чергу після обробки передає дані до бази даних, яка їх зберігає.

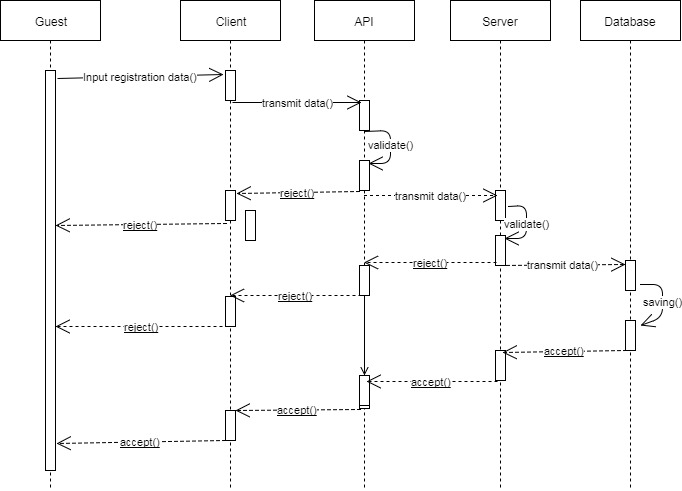


Рисунок 2.6 – Діаграма послідовностей

Особливості взаємодії елементів модельованої системи можуть бути представлені на діаграмі кооперації. Діаграма кооперації призначена для специфікації структурних аспектів взаємодії.

Діаграма кооперації зображена на рисунку 2.7 показує взаємодію користувача з веб-клієнтом. Головна особливість діаграми кооперації полягає в можливості графічно представити не тільки послідовність взаємодії, але і всі структурні відносини між об'єктами, які беруть участь у цій взаємодії.

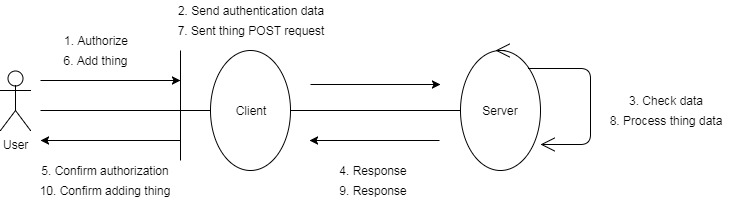


Рисунок 2.7 – Діаграма кооперації

Діаграма активності – це, в основному: блок-схема для подання потоку форми одного виду діяльності на інший вид діяльності. Потік управління малюється від однієї операції до іншої. Цей потік може бути послідовним, розгалуженим або одночасно.

Розглянемо діаграму активності, зображену на рисунку 2.8, яка наглядно демонструє процес логіну користувача для веб-користувача.

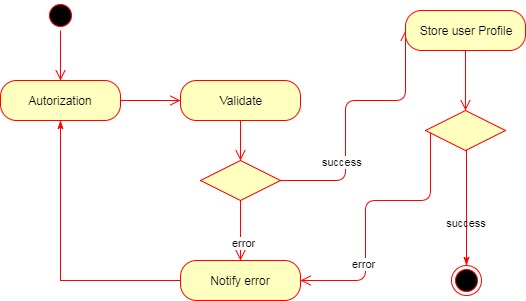


Рисунок 2.8 – Діаграма активності

В залежності від даних, які валідуються на сервері, клієнт зареєструється в систему, чи отримає помилку авторизації. Якщо валідація даних пройде не успішно, то клієнт повернеться до кроку авторизації.

На даному етапі моделювання програмної системи можна вважати завершеним та переходити безпосередньо до її реалізації.

# 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

Під час планування та моделювання програмного продукту було визначено, що проект має складатися з чотирьох частин:

1. веб-додаток, який уявляє собою основний робочий простір з основними функціями системи;
2. сервер з базою даних, на якому будуть оброблятися запити ззовні і зберігатися вся інформація про систему, її користувачів;
3. API-сервіси, що будуть звертатися до серверу. Клієнти відсилають запити до API через веб сайт;

г) мобільний клієнт, розроблений під Android, для надання механізму управління системою.

У наступних версіях планується створення IoT пристрію для розпізнавання місцезнаходження речі.

Інформація про кожну з частин даної версії системи буде приведена далі в відповідному розділі.

3.1 Загальні відомості про систему

Під час моделювання і планування програмного продукту було встановлено, що проект повинен складатися з таких частин:

1. веб-сайт;
2. сервер та база даних;
3. API-сервіси;
4. мобільний додаток.

Графічне зображення архітектури проекту наведено на рисунку 3.1.

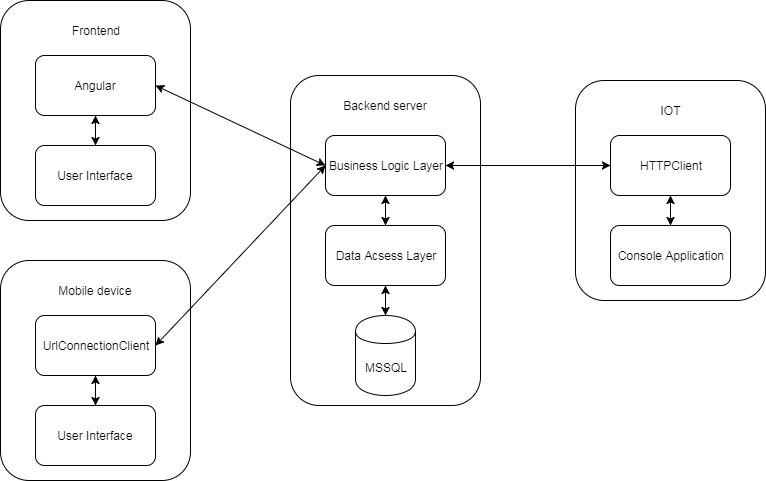


Рисунок 3.1 – Архітектура проекту

Таким чином архітектура проекту являє собою сервер з базою даних, доступом до даних і бізнес логікою, веб-сайт і мобільний додаток з користувацьким інтерфейсом.

Для розробки веб-сайту використовувався фреймворк Bootstrap, HTML, CSS, Angular. Для взаємодії клієнта з сервером використовується HTTP з’єднання. Для роботи з базою даних було використано Microsoft SQL Server. У відповідності з технологіями, використовується мови програмування C#, JavaScript. Для розробки сервера використовуються ASP.NET WEB API.

Під час розробки клієнту для Android використовується Android SDK, та мова програмування Java. Дані з мобільного додатку передаються на сервер та оновлюються у основній базі даних. Додаток має інсталюватися на мобільних девайсах та планшетах локально за допомогою apk-файлу.

Якщо описувати модель взаємодії усіх компонентів системи, то можна зрозуміти, що головною ланкою між усіма клієнтськими додатками та базою даних є сервер, API. Але за рахунок того, що всі головні елементи системи розділені, то це дозволяє гарно масштабувати систему.

3.2 Сервер бізнес логіки

Серверна частина - головний компонент сервісу. Система не буде повністю працездатною, якщо в ній буде відсутній хоча б один з перерахованих вище основних компонентів, однак без серверної частини система перестане бути працездатною зовсім. Як тільки буде виключений з системи сервер, сама система перестане існувати: замість неї буде кілька окремо взятих компонентів, ніяк один з одним не взаємодіючі додаток для управління, веб-сайт. Тобто для повноцінної роботи системи потрібен постійний доступ до серверу бізнес логіки.

Сервер реалізований за допомогою мови програмування C# та ASP.NET WEB API. Всі данні зберігаються з використанням СУБД Microsoft SQL Server, сама БД створена за допомогою підходу Сode First.

Сервер реалізує всю бізнес логіку та зберігає дані. Він написаний згідно шаблону API.

На рівні моделі знаходяться усі класи, що описують бізнес модель, та сервісний клас, у якому представлена уся бізнес-логіка. Зазначимо, що в усіх чотирьох компонентах системи присутні одні й ті ж класи-моделі для зручності передачі даних. На рис. 3.2 можна побачити моделі сервера.

Для виклику методів на сервері використовується WebAPI, що дозволяє отримувати та відправляти різноформатні дані по протоколу https.

В цілому, якщо розглядати взаємодію сервера бізнес логіки та веб-сервера, можна зробити висновок, що сервер є багаторівневим, має кілька окремих рівнів, пов'язаних один з одним: рівень загальнодоступного API, бізнес-рівень, рівень доступу до бази даних.

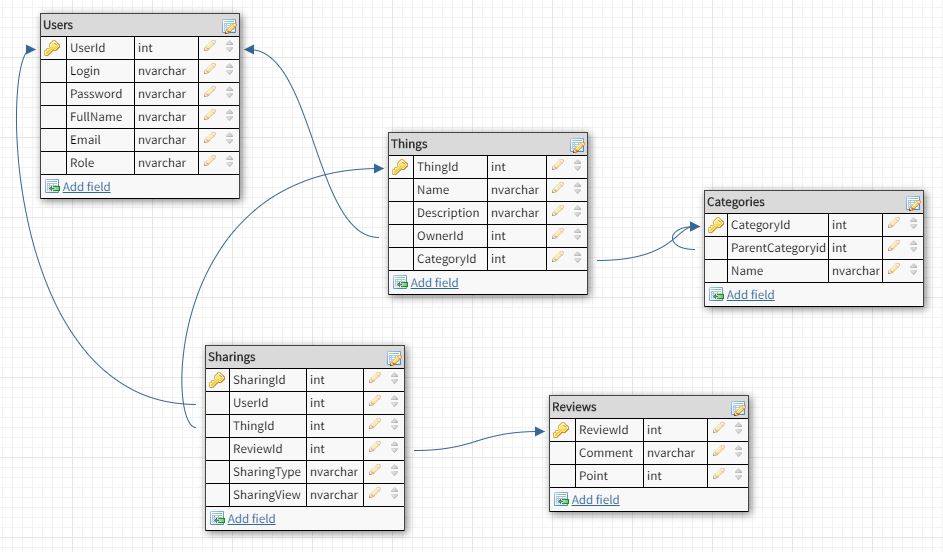


Рисунок 3.2 – Схема бази данних сервера

3.3 Веб-сайт

Сайт – це основний робочий простір, в якому буде відбуватися безпосередня робота в системі, тому можна стверджувати, що це основна частина проекту. Веб-клієнт написаний за допомогою Angular, HTML, CSS та фреймфорку адаптивної верстки Bootstrap. Всі статичні дані і програмні ресурси, які необхідні веб-клієнту, зберігаються на веб-сервері.

Кожен користувач-гість може зареєструватися в системі через веб-сайт, а не мобільний додаток, також користувач матиме можливість авторизуватися під уже існуючим ім'ям користувача і паролем, тоді його роль зміниться на активного користувача.

3.4 Мобільний клієнт

Останній з перерахованих вище компонентів системи і один з найбільш важливих її складових – це мобільний клієнт, що призначений для здійснення обміну. Мобільний додаток не повинен бути перенавантаженим на функціонал. Таке обмеження функцій має значення для раціоналізації використання системи. Оскільки дублювання функціоналу може заплутати користувачів та ускладнити використання системи було прийняте рішення про розподіл функціоналу.

Всі дані від користувача приходять на сервер саме від мобільного додатку, тому що саме з ним безпосередньо може взаємодіяти користувач. Це є основною причиною для того, щоб поставити юзабіліті мобільного клієнта на друге місце за важливістю критеріїв якості.

Мобільний додаток було реалізовано з використанням Android SDK під ОС Android. Він отримує дані про речі за допомогою використання https протоколу, та передає ці дані API-сервісу, який обробляє їх та перенапправлює на сервер.

# ІНТЕРФЕЙС І ФУНКЦІОНАЛ

* 1. Веб-сайт

Веб-сайт є додатком для кінцевих користувачів і має такий функціонал:

1. реєстрація нових користувачів;
2. авторизація користувачів;
3. додавання речей та іх відображення;
4. додавання категорій та іх відображення;
5. здійснення обміну.

Стартовий екран (див. рис. 4.1) має форму авторизації. Якщо користувач ще не зареєстрован у системі, він може перейти до форми реєстрації (див. рис. 4.2), натиснувши на надпис «Register». З форми реєстрації можна повернутися на форму авторизації, натиснувши на «Login».

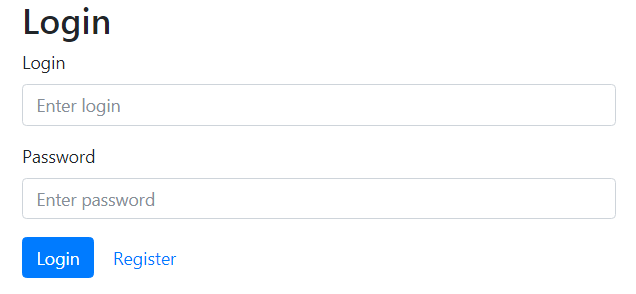


Рисунок 4.1 – Форма авторизації

Після авторизації користувач переходить на наступну сторінку (див. рис. 4.3), де відображається список всіх речей та форма для їх додавання.

Для додавання речі треба заповнити такі поля, як «Name», «Description», «Category». Після додавання справа в списку буде відображуватися додана річ.

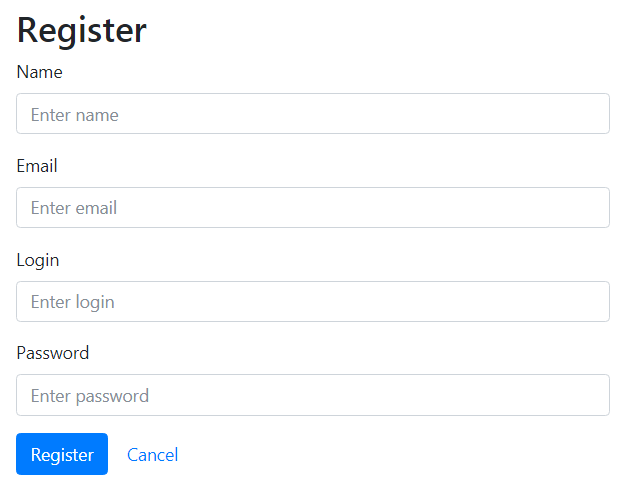


Рисунок 4.2 – Форма реєстрації

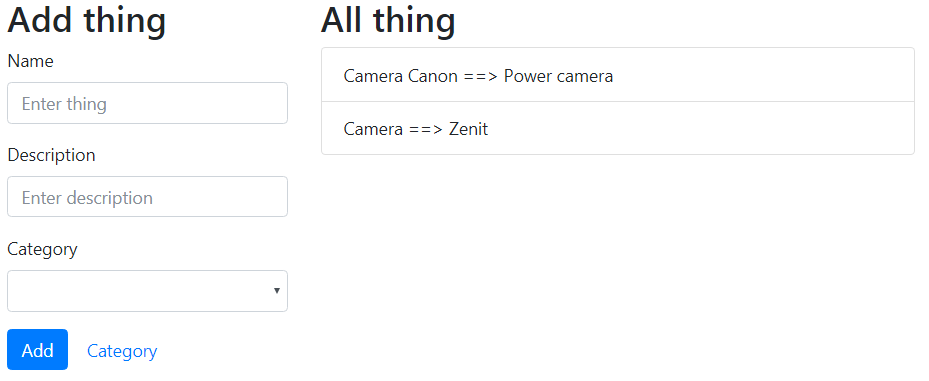


Рисунок 4.3 – Сторінка речей

Користувач з правами адміністратора може додати категорію, натиснувши на поле «Category» на сторінці речей, що дає можливість перейти на сторінку категорій (див. рис. 4.4).

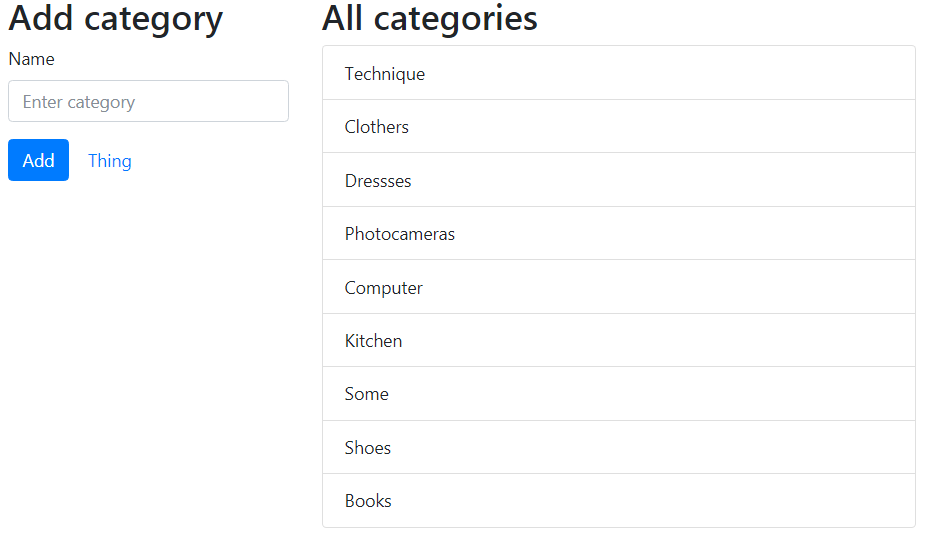


Рисунок 4.4 – Сторінка категорій

Для додавання категорій треба заповнити таке поле, як «Name». Після додавання справа в списку буде відображуватися додана категорія.

При виборі категорії є можливість подивитися на усі речі, додані користувачем в даній категорії (див. рис. 4.5).

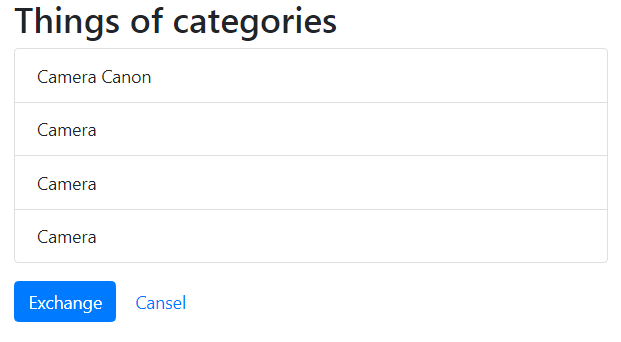


Рисунок 4.5 – Сторінка речей категорії

При виборі речі з категорії та активацією кнопки «», можна зробити обмін на обрану річ з іншим користувачем.

Якщо натиснути на «About», з’являється вікно, де користувач може отримати інформацію про даний сервіс (див. рис. 4.6).



Рисунок 4.6 – Навігація веб-сайту

Для того, щоб завершити сеанс роботи з системою, користувач повинен натиснути значок “Log out” в правому верхньому куті меню (див. рис. 4.6).

* 1. Мобільний додаток

Мобільний додаток дозволяє користувачу зареєструватися і потім авторизуватись. Форма авторизації зображена на рисунку 4.7.



Рисунок 4.7 – Форма авторизації мобільного додатку

Після авторизації користувач потрапляє на сторінку, де відображується список всіх речей користувача (див. рис. 4.8)

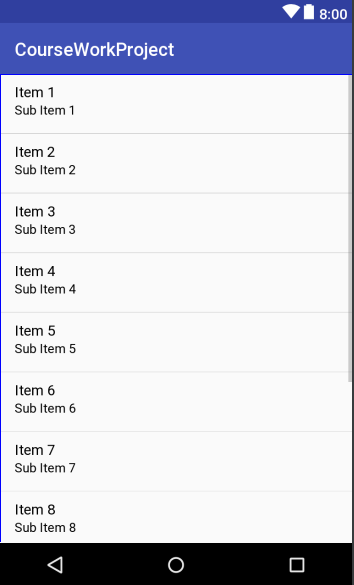


Рисунок 4.8 – Список речей користувача

ВИСНОВКИ

В результаті виконання курсової роботи було створено систему «Plyushkin» – сервіс для обмінну речей, за допомогою технології WebAPI, розробка клієнтського додатка з елементами адміністрування за допомогою мови програмування Angular, JavaScript.

Результатом курсового проекту є сервіс, що складається з клієнту та серверу, яка дозволяє користувачам переглядати основну інформацію про речі та також обмінюватися їми.

Використовуючи мову програмування C# було розроблене серверне програмне забезпечення на базі WebAPI, а також мобільний додаток для операційної системи AndroidOS версії 4 або новіше. За допомогою Angular, JavaScript, HTML, CSS та Bootstrap був створений веб-клієнт для даної системи.

Для реалізації всього необхідного функціоналу була проаналізована предметна область та виявлені взаємовідносини між основними об'єктами системи, які стали основою базової архітектури системи. Для збереження інформації, яка необхідна для повноцінного функціонування програмної системи, була спроектована та розгорнута реляційна база даних MSSQL, яка відображає усі зв’язки між сутностями предметної області. При проектуванні бази даних був проведений процес нормалізації.

Дана інформаційна система є багатофункціональною у використанні, має доступний інтерфейс, що дозволяє працювати з програмою користувачам з різним рівнем комп'ютерної грамотності, має середні вимоги до апаратного та програмного забезпечення, забезпечує досить високу швидкість роботи, безпечна у використанні, що є явними плюсами цього додатка.

Серед закордонних аналогів система виділяється можливістю обміну речей на невизначений термін та впевненістю у тому, що речі не будуть пошкодженні, що забезпечує вирішити проблеми сайтів обміну речей.

Отримана система задовольняє потребам користувача, а саме:

1. додавання речі;
2. можливість переглянути речі за категоріями;
3. можливість здійснити обмін між користувачами.

Під час проектування та розробки системи були отримані та поглиблені теоретичні та практичні знання з написання чистого коду і рефакторингу коду програмного забезпечення [10], проектування UML, роботи з Web API, архітектурними патернами та багато іншого.

ПЕРЕЛІК ПОСИЛАНЬ

1. Архитектура ПО [Електронний ресурс] – Режим доступу: http://brtrg.by/blog/post/66 – 23.05.2016 р. – Архитектура ПО

2. Дейт, К.Дж. Введення в системи баз даних [Текст] / К.Дж. Дейт. - 8-е вид. – М.: Вільямс, 2005.- 1328 с.

3. Грехем, І. Object-Oriented: Principles & Practice [Текст] / І. Грехем – 2-е вид. – М.: Вільямс, 2004.- 880 с.

4. Керівництво по MS SQL Server 2017 [Електронний ресурс] – Режим доступу: https://metanit.com/sql/sqlserver/ – 23.05.2016 р. – MS SQL Server

5. Керівництво по ASP.NET Web API [Електронний ресурс] – Режим доступу: https://metanit.com/sharp/aspnet\_webapi/ – 23.05.2016 р. – ASP.NET Web API

6. Керівництво по C# і .NET [Електронний ресурс] – Режим доступу: https://metanit.com/sharp/tutorial/ – 23.05.2016 р. – Керівництво по C# і .NET

7. Керівництво по Angular [Електронний ресурс] – Режим доступу: https://metanit.com/web/angular2 – 23.05.2016 р. – Керівництво по Angular

8. Грабер, М. SQL [Текст] / М.Грабер. – 2-е вид. – К: ЛОРІ, 2003. – 644 с.

9. Кренке, Д. Теорія і практика побудови баз даних [Текст] / Д.Кренке - 8-е вид. – СПб.: Пітер, 2003. – 800 с.

10. Рефакторинг [Електронний ресурс] – Режим доступу: https://refactoring.guru/ru/refactoring – 23.05.2016 р. – Рефакторинг

ДОДАТОК А

Vision and Scope

**Vision and Scope Document**

**for**

**Plyushkin**

**Version 1.0 approved**

**Prepared by Pityukova Maria**

**ПИ-15-2**

**KNURE**

**04.10.2017**

1. **Business Requirements**
2. **Background**

*Не існує такого сервісу на даний момент на ринку, де ви зможете віддати свою річ у володіння іншій людині на який-небудь проміжок часу без гарантії того, що цю річ вам повернуть в цілості й збереженні, або назовсім. Ми пропонуємо вам сервіс, за допомогою якого ви зможете перевіряти статус збереження і не боятися за свої речі.*

1. **Business Opportunity**

*Конкуренція в даній області практично немає, якщо не брати до уваги сайти обміну речей, але там мета полягає в іншому. Сайти обміну речами дають можливість просто обміну, без повернення тієї чи іншої речі. А наш сервіс дозволить власникам не тільки позичати речі на будь-який термін, але так само і заробляти бонуси, щоб можна було просто взяти в прокат річ, не маючи нічого.*

1. **Business Objectives and Success Criteria**

*Готовий продукт повинен задовольняти потреби користувачів в обміні, а також в впевненості того, що річ не буде мати ушкоджень по поверненню власнику, якщо таке буде. Мета – втілення в реальність успішного сервісу з можливостями соціалізації. Головні критерії успіху - досить велика аудиторія користувачів, звернення уваги зацікавлених людей на даний сервіс, універсальність і простота, швидке навчання роботи з сервісом, зручний інтерфейс користувача.*

1. **Customer or Market Needs**

*Зі сторони ринку, попит на даний продукт буде достатньо великим через невелику кількість аналогів у даній сфері.*

*Сервіс повинен бути простим для розуміння звичайним користувачем: зрозумілість, відсутність спеціалізованих позначень, простий інтерфейс, гнучка налагодженість параметрів для універсальності використання.*

*IoT-пристрій повинен бути інтуїтивно зрозумілим для використання звичайним користувачем; інтегровані мобільні і web-сервіси повинні мати простий і зрозумілий інтерфейс. Користувач буде купувати лише сам пристрій. Використання додатків і сайту буде безкоштовним.*

1. **Business Risks**

*Ризик пов’язаний з часом розробки програмного продукту і можливим виникненням помилок у процесі розробки, які можуть завадити реалізації тих чи інших особливостей програмного продукта та самого пристрою для спостерігання.*

1. **Vision of the Solution**
   1. **Vision Statement**

*ExchaneService**– це сервіс, за допомогою якого ви зможете віддавати речі іншому користувачеві назовсім чи на деякий проміжок часу, не боятися за неї, маючи змогу перевірити цілісність того, що ви позичили. На сайті користувач може додавати речі, обирати речі для обміну. Контроль здійснюється за допомогою мобільного додатку, дані про стан і місцезнаходження збираються автоматизовано, відповідними датчиками.*

* 1. **Major Features**

*F1: авторизація та реєстрація в програмі.*

*F2: додавання речі користувачем.*

*F3: віддалений збір інформації про стан та місцезнаходження речі.*

* 1. **Assumptions and Dependencies**

*Успішність даного продукту можна спрогнозувати виходячи з того, що він є унікальним і не має прямих конкурентів на програмному ринку. Від степеня реалізації усіх заявлених функцій напряму залежить оцінка та прийняття продукту користувачем. Для повноцінної роботи продукту необхідний постійний доступ до інтернету, а для використання мобільного додатку потрібна операційна система Android версії 4 або вище.*

1. **Scope and Limitations**
   1. **Scope of Initial Release.**

*У першій версії будуть реалізовані наступні функції:*

* *авторизація та реєстрація за допомогою сайта, відправлення нових даних на сервер и збереження у базі даних;*
* *додавання речей користувачем.*
  1. **Scope of Subsequent Releases**

*У випадку подальшої розробки продукта може бути реалізовано наступні функції:*

* *вибір речей за категоріями;*
* *здійснення обміну між користувачами;*
* *розробка пристрою для знаходження місцезнаходження речі і обробка отриманих даних на сервері.*

* 1. **Limitations and Exclusions**

*L1: Користувач може отримати доступ до сервісу тільки після авторизації.*

*L2: Обмін здійснюється тільки між зареєстрованими користувачами.*

ДОДАТОК Б

Фрагменти вихідного коду системи

***UsersController.cs***

[EnableCors(origins: "http://localhost:8080", headers: "\*", methods: "\*")]

public class UsersController : ApiController

{

private readonly UserService \_unitOfWork;

public UsersController()

{

\_unitOfWork = new UserService();

}

// GET: api/Users

public IHttpActionResult GetUsers()

{

var users = \_unitOfWork.GetUsers();

return Json(users);

}

// GET: api/Users/5

[ResponseType(typeof(User))]

public IHttpActionResult GetUser(int id)

{

var user = \_unitOfWork.GetUser(id);

return Json(user);

}

// PUT: api/Users/5

[ResponseType(typeof(void))]

public IHttpActionResult PutUser(User user)

{

try

{

\_unitOfWork.PutUser(user);

return Ok();

}

catch

{

return BadRequest("Cannot put user");

}

}

// POST: api/Users

[HttpPost]

[ResponseType(typeof(void))]

public IHttpActionResult PostUser([FromBody]User user)

{

try

{

\_unitOfWork.PostUser(user);

return Ok();

}

catch

{

return BadRequest("Cannot post user");

}

}

// DELETE: api/Users/5

[ResponseType(typeof(User))]

public IHttpActionResult DeleteUser(int id)

{

try

{

\_unitOfWork.DeleteUser(id);

return Ok();

}

catch

{

return BadRequest("Cannot put user");

}

}

protected override void Dispose(bool disposing)

{

base.Dispose(disposing);

}

private bool UserExists(int id)

{

return \_unitOfWork.ExistUser(id);

}

}

***register.controller.js***

(function () {

'use strict';

angular

.module('app')

.controller('RegisterController', RegisterController);

RegisterController.$inject = ['UserService', '$location', '$rootScope', 'FlashService'];

function RegisterController(UserService, $location, $rootScope, FlashService) {

var vm = this;

vm.user = {

UserId: 0,

Login: "",

Password: "",

FullName: "",

Email: "",

Role: "user"

}

vm.register = register;

function register() {

vm.dataLoading = true;

UserService.Create(vm.user)

.then(function (response) {

FlashService.Success('Registration successful', true);

$location.path('/login');

});

}

}

})();

***login.controller.js***

(function () {

'use strict';

angular

.module('app')

.controller('LoginController', LoginController);

LoginController.$inject = ['UserService', '$location', 'AuthenticationService', 'FlashService'];

function LoginController(UserService, $location, AuthenticationService, FlashService) {

var vm = this;

vm.user = { login: "", password: "" };

vm.auth = auth;

(function initController() {

})();

function auth() {

vm.dataLoading = true;

UserService.Auth(vm.user).then(function (response) {

localStorage.setItem("currentUser", JSON.stringify(response));

FlashService.Success('Logged in', true);

$location.path('/home');

});

}

}

})();

***home.controller.js***

(function () {

'use strict';

angular

.module('app')

.controller('HomeController', HomeController);

HomeController.$inject = ['UserService', '$location', '$rootScope', 'FlashService'];

function HomeController(UserService, $rootScope) {

var vm = this;

vm.currentUser = JSON.parse(localStorage.getItem("currentUser"));

vm.userThings = [];

vm.categories = [];

(function initController() { UserService.GetUserThings(vm.currentUser.UserId).then(function (response) {

vm.userThings = response;

});

UserService.GetCategories().then(function (response) {

vm.categories = response;

});

})();

vm.addThing = addThing;

vm.thing = {

ThingId: 0,

Name: "",

Description: "",

OwnerId: vm.currentUser.UserId,

CategoryId: 1

}

function addThing() {

vm.dataLoading = true;

UserService.CreateThing(vm.thing)

.then(function () {

location.reload();

});

}

}

})();

***category.controller.js***

(function () {

'use strict';

angular

.module('app')

.controller('CategoryController', CategoryController);

CategoryController.$inject = ['UserService', '$location', 'FlashService'];

function CategoryController(UserService, $location, FlashService) {

var vm = this;

vm.categories = [];

vm.addCategory = addCategory;

vm.openCategory = openCategory;

(function initController() {

UserService.GetCategories().then(function (response) {

vm.categories = response;

});

})();

vm.category = {

CategoryId: 0,

ParentCategoryId: 4,

Name: ""

}

function addCategory() {

vm.dataLoading = true;

UserService.CreateCategory(vm.category)

.then(function (response) {

location.reload();

});

}

function openCategory(CategoryId){

$location.path("/category/" + CategoryId);

}

}

})();

***index.html***

<!DOCTYPE html>

<html ng-app="app">

<head>

<meta charset="utf-8" />

<title>Shop</title>

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<link href="vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">

<link href="css/shop-homepage.css" rel="stylesheet">

<script src="vendor/jquery/jquery.min.js"></script>

<script src="vendor/bootstrap/js/bootstrap.bundle.min.js"></script>

<script src="//code.angularjs.org/1.6.0/angular.min.js"></script>

<script src="//code.angularjs.org/1.6.0/angular-route.min.js"></script>

<script src="//code.angularjs.org/1.6.0/angular-cookies.min.js"></script>

<script src="app.js"></script>

<script src="app-services/authentication.service.js"></script>

<script src="app-services/flash.service.js"></script>

<script src="app-services/user.service.js"></script>

<script src="home/home.controller.js"></script>

<script src="login/login.controller.js"></script>

<script src="register/register.controller.js"></script>

<script src="category/category.controller.js"></script>

<script src="exchange/exchange.controller.js"></script>

<script src="categoryItems/categoryItems.controller.js"></script>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js"></script>

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark fixed-top">

<div class="container">

<a class="navbar-brand" href="#">Shop</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarResponsive" aria-controls="navbarResponsive" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarResponsive">

<ul class="navbar-nav ml-auto">

<li class="nav-item active">

<a class="nav-link" href="#!/home">Home

<span class="sr-only">(current)</span>

</a>

</li>

<!-- <li class="nav-item">

<a class="nav-link" href="#!/category" >Category</a>

</li> -->

<li class="nav-item">

<a class="nav-link" href="" data-toggle="modal" data-target="#myModal">About</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#!/login"><img class="d-block img-fluid" src="https://png.icons8.com/male-user/win8/24/ffffff"></a>

</li>

<li class="nav-item">

<a class="nav-link" href="#!/exchange"><img class="d-block img-fluid" src="https://png.icons8.com/shopping-bag/win10/24/ffffff"></a>

</li>

<li class="nav-item">

<a class="nav-link" href="#!/login"><img class="d-block img-fluid" src="4.png"></a>

</li>

</ul>

</div>

</div>

</nav>

<div class="modal fade" id="myModal" role="dialog">

<div class="modal-dialog modal-lg">

<div class="modal-content">

<div class="modal-header">

<button type="button" class="close" data-dismiss="modal">&times;</button>

</div>

<div class="modal-body">

<p>Shop — the first and so far one of a kind internet service.

Thanks to him you can share your thing with another.

And not only. You will get many bonuses. Come in :)

</p>

<img class="d-block img-fluid" src="5.jpg">

</div>

<div class="modal-footer">

<button type="button" class="btn btn-default" data-dismiss="modal">Close</button>

</div>

</div>

</div>

</div>

<div ng-view></div>

</body>

</html>