МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ФІЗИКИ ТА МАТЕМАТИКИ

ПРОЄКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ «АВТОСЕРВІС»

Кваліфікаційна робота

на здобуття ступеня вищої освіти «магістр»

Виконав: студент 2 курсу, групи 12-241М
Спеціальності:121 Інженерія програмного забезпечення
Подмазко Євгеній Сергійович
Керівник: д.фіз.-матем.н., професор
Песчаненко Володимир Сергійович
Рецензент: д.пед.н., професор
Мелітопольського державного педагогічного університету
Круглик Владислав Сергійович.

Херсон – 2022

ЗМІСТ

ВСТУП

Актуальність теми. День сучасної людини починається телефону, а саме пропущені смс, повідомлення, пошта, новини і т.і. Тим не менш з цього часу вона вже не розлучається з ним аж до самої ночі.

На данний момент люди сидять в телефонах, спілкуючись з друзями, на побаченнях, на роботі і в самоті. Вони сидять в ньому коли їдуть, коли ведуть автомобіль, іноді коли приймають їжу, а близько 11% людей по статистиці приймають душ з цим пристроєм. Як тільки виникає вільна хвилина, людина одразу ж автоматично тягнеться за телефоном.

Ще не так давно, близько 10-12 років тому, ми могли вийти з дому і забути телефон. Але зараз такого не відбувається. Ми самі того не помітили як цілком і повністю погрузились у цифрове середовище, хоча по суті воно лише на етапі свого становлення.

Цифровізація – визначна тенденція розвитку людської цивілізації, яка формує більш рівноправне суспільство та кращі механізми управління, розширює доступ до охорони здоров'я, освіти та банківської справи, підвищує якість та охоплення державних послуг, розширює спосіб співпраці людей, а також дає змогу скористатися більшим розмаїттям товарів за нижчими цінами. Ми могли з вами переконатись на прикладі пандемії, яка довела цінність та необхідність цифрових технологій для добробуту населення та розвитку економік.

Під час локдауну в період пандемії, або під час воєнного стану одним із важливих та необхідних механізмів у суспільстві став транспорт. Автомобілі допомагають забезпечити стабільну та безперервну транспортну логістику. В наш час ще не винайшли автомобілі які не потребували би обслуговування, більше того, технічне обслуговування і ремонт транспортних засобів та їх складових виконують з метою підтримання їх у належному стані та

забезпечення встановлених виробником технічних характеристик під час використання, зберігання або утримання протягом періоду експлуатації. Так чи інакше потрібен виконавець технічного обслуговування. Наприклад станція технічного обслуговування, існує безліч видів спеціалізацій цих підприємств, розглянемо саме ті, що надають загальний набір послуг. Для отримання послуг вам необхідно особисто зателефонувати, відправити мейл, приїхати на підприємство, поспілкуватись з менеджером, заповнити купу бланків та паперів, за необхідності відстояти чергу, в касі оплатити і т.і. Чи не забагато зайвих дій? На початку вступу я розповідав про телефони та цифровізацію, ось до чого я все веду. Через велику чисельність побічних процесів пов'язаних з ремонтом транспортного засобу для клієнтів, та полегшення ведення бізнесу для підприємців, вирішено створити мобільний додаток. Саме для того щоб зменшити кількість проблем з обслуговуванням автомобіля на станціях технічного обслуговування, було обрану тему кваліфікаційної роботи : «Мобільний додаток Автосервіс».

Мета кваліфікаційної роботи: розробка «мобільного додатку автосервіс»

Відповідно до поставленої мети роботи необхідно вирішити наступні **завдання дослідження:**

1) Розглянути мобільні додатки у світі та Україні;
2) Визначити функціональні вимоги мобільного додатку «Автосервіс»;
3) Проєктування мобільного додатку;
4) Визначити архітектурні рішення мобільного додатку;
5) Створення мобільного додатку;

Об'єкт дослідження: Мобільні додатки.

Предмет дослідження: Мобільні додатки для автосервісу.

Методи дослідження: Під час написання кваліфікаційної роботи було використано такі методи дослідження, як: аналіз літературних джерел, спостереження, систематизація та узагальнення.

Структура роботи. Робота складається зі вступу, трьох розділів, висновків та списку використаних джерел.

РОЗДІЛ 1

ОГЛЯД СУЧАСНИХ МОБІЛЬНИХ ДОДАТКІВ АВТОСЕРВІСІВ, ЇХ ПОРІВНЯЛЬНИЙ АНАЛІЗ

1.1. Розробка мобільних додатків для IOS

“По суті, широко відома iOS - це добре усталена абревіатура, що бере свій початок від фрази «операційна система iPhone», яка спочатку була створена для системи мобільних пристроїв виробництва Apple, таких як iPhone, iPad і iPodTouch, а тепер також AppleTV.” [1].

Інтерфейси користувача, створені для iOS, створені на основі прямого маніпулювання за допомогою жестів мультитач. Елементи керування інтерфейсом мають різноманітні функції та форми, включаючи кнопки, повзунки та перемикачі. Жести, що відбуваються в процесі взаємодії з таким типом інтерфейсу, також різноманітні, наприклад, свайп, тап, щипки, зворотні щипки; більше того, усі вони мають специфічні визначення в контексті операційної системи iOS. Деякі програми iOS навіть використовують внутрішні акселерометри, які можуть реагувати на випадки тряски або обертання пристрою в трьох вимірах. Ще одна типова функція - перемикання портретного і альбомного режимів.

“З точки зору розвитку Apple віддає стійку перевагу простим і міцним речам, і ця концепція застосовується до їх продукції, програми, інструменти та фреймворки. Очікується, що коли розробник IOS створює програму для iPhone, вона працюватиме на будь-якому іншому пристрої з IOS. Цей аспект розробки IOS може заощадити багато часу програмістам, оскільки при написанні коду, скажімо, для iPhone розробник використовує одну мову програмування для всіх комп’ютерних пристроїв Apple. Іншими словами, програму, створену для iPhone, можна пізніше навіть інтегрувати в Mac.”[2]

“Як у дизайні UI/UX, так і в розробці для IOS, спеціалісти, які створюють екрани та пишуть код, застосовують свій політ творчості з точки зору чітких правил і рекомендацій, яких слід дотримуватися. Програми, які не дотримуються цих інструкцій, ризикують не представити свою програму в екосистему пристроїв Apple.”[3].

Як уже згадувалося раніше, IOS дозволяє користувачам взаємодіяти зі своїми пристроями та продуктами за допомогою таких жестів, як проведення, натискання та зведення щипків. Ці дії пальцями зазвичай виконуються на сенсорних дисплеях, які забезпечують швидку реакцію та приймають введення кількома пальцями.

Архітектуру IOS можна представити як чотири рівні абстракції, які її визначають:

Cocoa Touch. Рівень, що містить різноманітні фреймворки, які визначають зовнішній вигляд програми. Він також забезпечує базову інфраструктуру програм і підтримку основних технологій, таких як багатозадачність, сенсорне введення, push-сповіщення та багато системних служб високого рівня.

Медіа. Рівень із технологіями графіки, аудіо та відео, які розробники використовують для реалізації мультимедійних можливостей у програмах. Технології цього рівня дозволяють створювати програми, які виглядають і звучать чудово.

Основні послуги. Рівень складається з основних системних служб для програм. Основними службами є Core Foundation і Foundation frameworks, які визначають основні типи, які використовують усі програми. Він також містить окремі технології для підтримки таких функцій, як місцезнаходження, iCloud, соціальні мережі, мережа тощо.

Ядро ОС. Рівень включає такі служби, як безпека, локальна автентифікація та базові інфраструктури Bluetooth.

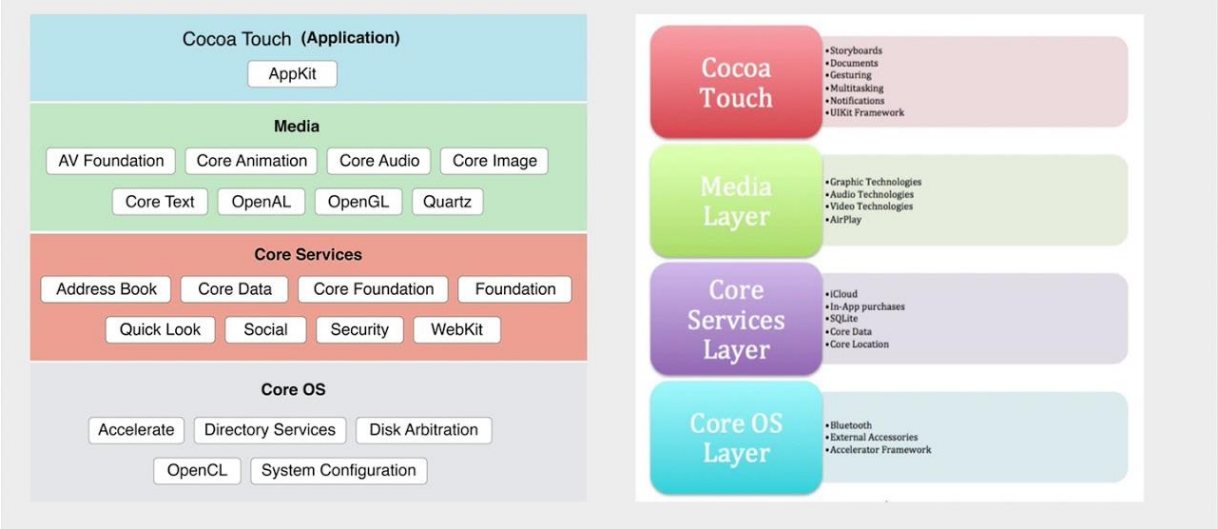


Рисунок 1.1 Архітектура IOS

Наочне представлення архітектури IOS на рисунку 1.1:

Apple рекомендує розробникам писати код на найвищому застосовному рівні, коли це можливо, і використовувати лише фреймворки нижчого рівня для функцій, які не доступні на вищих рівнях.

“У розробці програмного забезпечення шаблон проєктування архітектури є питанням структури. Це не готова архітектура, яку можна трансформувати безпосередньо в код. Патерни подібні до надійних методів, які розробник може використовувати для вирішення типових проблем під час створення програми чи системи.”[4].

Загалом, шаблон архітектури у випадку розробки можна описати як багаторазове рішення, яке можна багаторазово застосовувати до проблем, які часто виникають у певному контексті розробки та інженерних взаємодій. Цей вид шаблонів враховує різні обмеження та обмеження, включаючи проблеми продуктивності обладнання, доступності, середовища використання та навіть конкретних бізнес-цілей.

Серед основних шаблонів проєктування архітектури можна назвати:

- **MVC** (модель–перегляд–контролер)
- **MVP** (модель–перегляд–ведучий)
- **MVVM** (модель–перегляд–перегляд-модель)
- **VIPER** (Перегляд-інтерактор-ведучий-сутність-маршрутизація)

Щоб побачити, як це працює на практиці, візьмемо для прикладу шаблон MVC і опишемо його трохи докладніше. Модель-подання-контролер (MVC) — це шаблон архітектурного проєктування, який розбиває код на три частини: інтерфейс користувача (подання), дані (модель) і програмне забезпечення, яке обмінюється даними між ними (контролер). Будівельними блоками програм є об’єкти – масиви коду, організовані за шаблоном MVC. Кожен екран програми є представленням: модель даних контролює вміст, який він відображає, а контролер керує зв’язком між представленням і моделлю. Контролер model-view-controller — це лише один із шаблонів проєктування архітектури, які розробники використовують під час створення програми, однак він популярний у розробці.

“Xcode — це інтегроване середовище розробки (IDE) від Apple і основний інструмент для створення програм для iOS. Розробники iOS можуть працювати із застосуванням Objective-C або Swift, фактично використовуючи лише Xcode. Це програмне забезпечення працює лише на комп’ютерах Mac, і немає офіційних способів запустити його на ПК з Windows або Linux.”[5].

Xcode має масу функцій для розробки iOS, серед яких:

Конструктор інтерфейсу . Це дозволяє проєктувати та тестувати інтерфейс користувача без написання коду. Interface Builder допомагає створити прототип, а потім підключити інтерфейс до джерела в редакторі Xcode.

Редактор вихідного коду . Це професійний редактор із доповненням коду, згортанням коду, підсвічуванням синтаксису та повідомленнями, які відображають попередження, помилки та іншу контекстно-залежну інформацію.

Безперервна інтеграція. Це функція OS X Server, яка керує серверними ботами, які постійно створюють, аналізують, тестують і навіть архівують проєкти Xcode.

XCTest Framework. Він допомагає створювати модульні тести, які перевіряють продуктивність програм iPhone, iPad, Apple Watch, Apple TV і Mac.

“**Симулятор iOS.** Це дозволяє створювати прототипи та тестувати програми в процесі розробки. Цей інструмент тестування імітує пристрої iOS, watchOS і tvOS перед тестуванням програми на реальному пристрої.”[6]

Є дві основні мови програмування IOS це Swift та Objective-C. На даний момент Objective-C зазвичай описують як об’єктно-орієнтовану мову програмування загального призначення, яка додає до мови програмування C обмін повідомленнями у стилі Smalltalk. До появи Swift Objective-C була основною мовою програмування, яку використовували розробники для операційних систем OS X та iOS та їх API.

Swift, представлений Apple на WWDC у 2014 році, приніс власну дозу революції. У загальному описі Swift — це скомпільована мова програмування загального призначення з кількома парадигмами, яка розширює попередню сферу операційних систем, оскільки вона була розроблена для iOS, OS X, watchOS, tvOS і Linux. Swift був розроблений для забезпечення роботи з фреймворками Apple Cocoa та Cocoa Touch, а також із великою частиною коду Objective-C, уже написаного для продуктів Apple.

Спочатку Swift мав бути більш стійким і безпечним, ніж Objective-C, а також більш лаконічним і зручним для розробників. Його створено за допомогою фреймворку компілятора LLVM, включно з Xcode 6, і з використанням бібліотеки середовища виконання Objective-C, яка дозволяє виконувати код C, Objective-C, C++ і Swift в одній програмі. Таким чином, це сприяє розширенню професійних горизонтів для розробників, роблячи їхню роботу більш ефективною.

У будь-якому випадку, вибір мови програмування, фреймворку, методу та інструменту для створення додатків для iOS має ґрунтуватися на намірі забезпечити ефективний продукт як для клієнта, так і для кінцевого користувача.

1.2. Розробка мобільних додатків для Android

“**Операційна система Android** є найбільшою базою встановлених серед різноманітних мобільних платформ у всьому світі. Сотні мільйонів мобільних пристроїв працюють на **Android** у понад 190 країнах світу. До кінця 2021 року він завоював близько **71%** частки світового ринку, і ця тенденція зростає з кожним днем. У вересні 2008 року на ринку з'явився перший пристрій на базі Android. Тепер він домінує в галузі мобільних ОС через довгий список функцій, які вона надає. Він зручний для користувача, має величезну підтримку спільноти, забезпечує більший ступінь налаштування, і велика кількість компаній створюють смартфони, сумісні з Android. Як наслідок, на ринку спостерігається різке зростання попиту на розробку мобільних додатків для Android, а разом з цим компаніям потрібні розумні розробники з відповідним набором навичок.”[7].

Мови програмування, які використовуються для розробки додатків для Android:

1. **Java**
2. **Kotlin**

Google віддає перевагу розробці додатків Android за допомогою Kotlin, оскільки вона є офіційною мовою для розробки Android, яку розробляє та підтримує JetBrains. Раніше Java вважалася офіційною мовою розробки Android. Kotlin став офіційним на Google I/O 2017.

Переваги Android розробки

- Android є операційною системою з відкритим вихідним кодом, тому її підтримує велика спільнота.
- Розробка програми зроблена згідно з рекомендаціями Google, що полегшує розробникам створення інтуїтивно зрозуміліших програм.
- Фрагментація дає більше можливостей програмам Android. Це означає, що програма може виконувати дві дії на одному екрані.
- Випуск програми для Android у магазині Google Play легше порівняно з іншими платформами.

Недоліки Android розробки

- Фрагментація забезпечує дуже інтуїтивно зрозумілий підхід до взаємодії з користувачем, але вона має деякі недоліки, коли команді розробників потрібен час, щоб адаптуватися до різних розмірів екрана мобільних смартфонів, які зараз доступні на ринку, і викликати певні функції в додатку.
- Пристрої Android можуть значно відрізнятися. Тому тестування програми ускладнюється.
- Оскільки розробка та тестування займають більше часу, вартість програми може зрости залежно від складності та функцій програми.

Архітектура Android містить різну кількість компонентів для підтримки будь-яких потреб пристроїв Android. Програмне забезпечення Android містить ядро Linux з відкритим вихідним кодом, яке містить набір бібліотек C/C++, доступ до яких доступний через служби програми.

Серед усіх компонентів ядро Linux забезпечує основні функції операційної системи для смартфонів, а віртуальна машина Dalvik (DVM) забезпечує платформу для запуску програми Android.

Основні компоненти архітектури Android такі: -

- Додатки
- Фреймворк програми
- Android Runtime
- Бібліотеки платформи
- Ядро Linux

Наочне представлення архітектури android з кількома основними

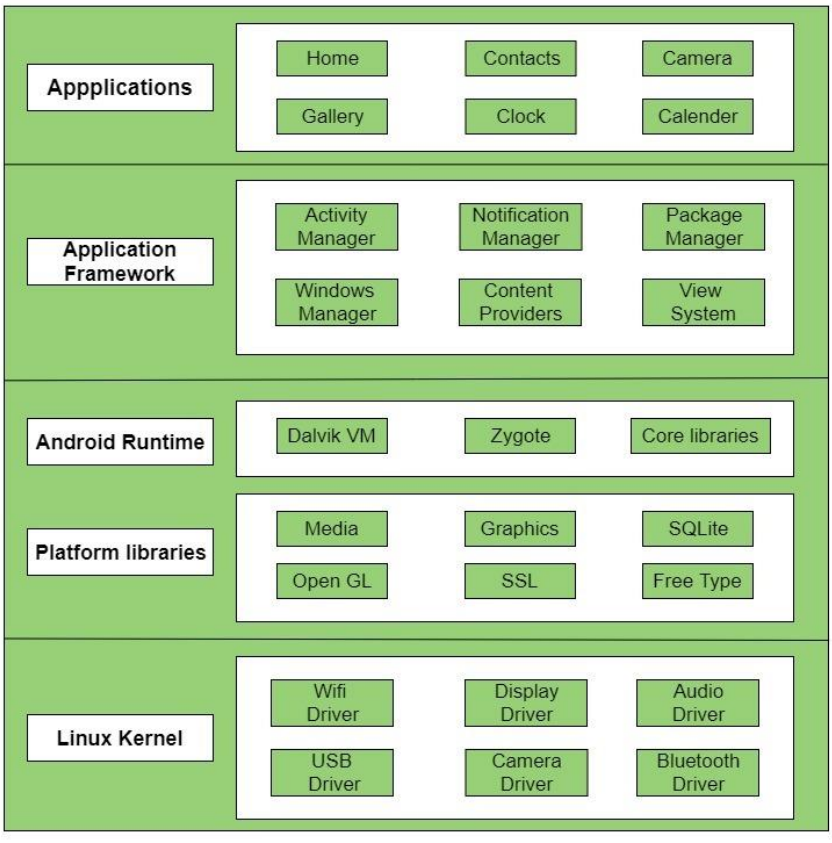


Рисунок 1.2 Архітектура Android

компонентами та їх підкомпонентами зображено на рисунку 1.2:

1.3 Огляд мобільних додатків аналогів

„ Мобільний додаток, який найчастіше називають додатком, - це тип прикладного програмного забезпечення, призначеного для роботи на мобільному пристрої, наприклад, смартфоні або планшетному комп'ютері. Мобільні додатки часто служать для надання користувачам подібних сервісів до тих, що мають доступ до ПК. Програми, як правило, невеликі, окремі програмні блоки з обмеженою функцією. Це використання програмного забезпечення спочатку було популяризоване Apple Inc. та його App Store, який пропонує тисячі програм для iPhone, iPad та iPod Touch. Мобільний додаток також може бути відомий як додаток, веб-додаток, онлайн-додаток, додаток iPhone або додаток для смартфона.” [8].

„Мобільний додаток – це артефакт інформаційних технологій програмного забезпечення, спеціально розроблений для мобільних операційних систем, встановлений на портативних пристроях, таких як смартфони чи планшетні комп’ютери. Він являється багатофункціональним та мультизадачним елементом для обслуговування персоналу.” [9].

Бізнес автосервісу є дуже конкурентним, для того щоб бути на вершині цієї ніші треба постійно вигадувати нові стратегії залучення та обслуговування клієнтів. Завдання власників автосервісу – стати більш сприятливим, доступними, зрозумілішими. Мобільний додаток стане міцним та потужним інструментом для збільшення можливостей бізнеса та полегшить роботу з клієнтами. А в випадку клієнтів, стає зручним та сучасним методом у обслуговуванні, та невід’ємним у догляді за власним авто.

Тож хотів би почати огляд функціоналу додатків для автосервісу в Україні та світу.

У кожному додатку має бути загальний, скажімо так базовий набір функцій. Можна відмовитись від них в разі якщо є на то якісь вагомі причини, але практика показує що ці функції є обов'язковими для будь-якого успішного продукту.

Віднести до **стандартного** функціоналу всякого додатку можна:

- *Інформація про компанію.* Користувачі мають знати з ким вони взаємодіють. Немає довіри невідомим. Вони мають знати хто є відповідальним за даний продукт. Також таким чином ви можете збільшити впізнаваність вашого бренду. Ну і звичайно розмітивши контакти в цьому розділі, ви зможете відповідати на запитання, що виникнуть в процесі використання вашої програми.
- *Зворотній зв'язок.* Відгуки клієнтів чудовий стимул для розвитку та вдосконалення. У будь-якому разі, відгуки мають зіграти на вашу користь(в разі якщо продукт якісний), адже перед використанням користувач спирається на декілька факторів впливу, і відгуки про додаток є одним із них.
- *Інтеграція із соціальними мережами.* Кожна людина шукає легший шлях, це дозволить швидше і простіше зареєструватись та авторизуватись в системі, дає змогу поділитись із друзями, що в свою чергу теж йде вам на користь.
- *«FAQ».* У разі якщо запитань занадто багато, або ви знаєте що ваш продукт вимагає певних уточнень, було б доцільно додати цей розділ.
- *«Push-сповіщення».* Така функція допоможе зблизитись з аудіторією, бути з ними постійно на зв'язку. Така близькість до користувачів повинна збільшити кількість продаж.

- *Синхронізація з основним сайтом.* Така можливість повинна бути, адже ви дасте можливість користувачам перейти на сайт, якщо є в цьому необхідність, а також обумовлюється тим, що в додатку є ще не всі функції. І це має полегшити працю зі своїми ресурсами, тому що не доведеться в кожному з них оновлювати інформацію окремо.
- *Галерея.* Фото та відео матеріали підвищують залученість та довіру користувачів. Така галерея, ознайомить із вашими товарами, послугами, офісами, персоналом та задоволеними клієнтами.

Щодо **індивідуального** функціоналу, то пропоную оглянути на прикладі українського мобільного додатку «**Автосервіс PRO.CTO.UA**»:

- *Система бонусів та знижок.* Бонуси та знижки нараховуються автоматично і не потребують підтверджень, якщо у вас встановлений додаток і ви залоговані в ньому.
- *Запис на сервіс.* Такий хід допоможе обрати користувачу час та день запису на СТО не виходячи з дому.
- *Історія записів на сервіс через онлайн сервісну книжку.* Дуже зручно у разі необхідності перевірити в сервісній книжці, коли і які роботи проводились з автомобілем, особливо якщо немає доступу до звичайної дилерської книжки. Таким чином клієнт може перевірити що було замінено, обслужено та скільки це коштувало.
- *Користування веб-камерами, спостерігання за ремонтом свого авто.* В разі якщо не має змоги бути присутнім на сервісі, можна перевірити в якій стадії знаходиться робота над автомобілем. Таким чином станція доводить свою прозорість та чесність, а клієнти

мають змогу впевнитись, що їх транспортний засіб в роботі та буде готовий в срок.

- *Спостерігання за новинами, знижками, акціями.* Користувачам доступні новини, акції, знижки на послуги, продукти і т.і.
- *Допомога у плануванні ремонту автомобіля у відповідності з усіма регламентами заводу-виробника.* Це стане гарним нагадуванням, коли і що саме потрібно обслуговувати, яка кількість коштів необхідна, скільки часу це заме, і які процеси мають бути виконані. З рештою клієнти можуть менше хвилюватись за автомобіль адже в будь-який час можна перевірити коли саме треба звертатись до спеціалістів.

Наприклад для порівняння «**AUTODOC**» має деякі функції як у попередньому додатку, але є і унікальні, такі як:

- *Каталог автозапчастин.* Дозволяє знайти будь-яку автозапчастину за номером артикула, та замовити серед великої кількості брендів, а також отримати інформацію щодо часу прибуття.
- *Спосіб оплати.* Є понад 15 різноманітних способів оплати послуг.
- *Доставка автозапчастин на адрес.* Дуже зручно коли не потрібно витрачати час на подорож за запчастинами, краще коли запчастини доставлять до вас до дому.
- *Спостереження за статусом доставки.* Іноді при доставці виникають непередбачувані ситуації та продукт може затримуватись, тут в нагоді буде функція відстежування, в будь-який час можна перевірити місцезнаходження посилки.

Ще одним аналогом є «**Express Auto Service & Repair**» він має такі функції:

- *Історія обслуговування.*
- *Нагадування та сповіщення про обслуговування.*
- *Винагороди за відвідування магазину.*
- *Онлайн консультація*

Тобто виходячи з функцій, можна відмітити, що цей додаток являється більше помічником у володінні автомобілем.

Наступний додаток по суті такий же як попередній, називається «**Simply Auto: Car Maintenance**» але є відмінності :

- *Відстеження пробігу автомобіля.*
- *Ведення журналу поїздок за допомогою GPS та Bluetooth.*
- *Збирання різних даних і запис у статистику.*

Цей додаток призначений для власників малих та середніх автопарків, для водіїв котрі хочуть вести облік технічного обслуговування свого авто.

Для ідеального набору функціоналу мобільного додатку немає конкретних рекомендацій та чітко визначених правил. Потрібно починати з основних функцій, стежити за відгуками і додавати по необхідності нові. Не потрібно боятись запускати функціонал ще не вдосконаленому вигляді, адже доопрацювати його завжди буде час, але також доцільно буде скасувати його розробку у разі якщо він не користуватиметься попитом.

1.4. Функціональні вимоги

„ Функціональна вимога - це формулювання того, як система повинна поводитися. Він визначає, що повинна робити система, щоб задовольнити потреби або очікування користувача. Функціональні вимоги можна розглядати як функції, які виявляє користувач. Вони відрізняються від

нефункціональних вимог, які визначають, як система повинна працювати всередині (наприклад, продуктивність, безпека тощо).

Функціональні вимоги складаються з двох частин: функції та поведінки. Функція – це те, що виконує система (наприклад, «розрахувати податок з продажу»). Поведінка залежить від того, як система це робить (наприклад, «Система розраховує податок із продажу, помноживши ціну покупки на ставку податку»).” [10]

Виходячи з вище сказаного, та проаналізувавши аналоги мобільного додатку «Автосервіс», він має містити такі функціональні вимоги:

1. Вхід в систему.
2. Підмінний автомобіль.
3. Оформлення та прийняття авто, без відвідування автосервісу.
4. Перегляд камер спостереження.
5. Замовлення автозапчастин.
6. Перегляд новин, скидок, акцій.
7. Запис на сервіс.
8. Перегляд історії обслуговування.
9. Підключення платіжної системи.

Функція «*Підмінний автомобіль*». Техніка так чи інакше час від часу виходить зі строю, навіть через найменший болтик може зупинитись весь механізм завдяки якому ми маємо змогу пересуватись від пункту «А» до пункту «Б». І поломки виникають в будь-який момент, та навіть в самий непідходящий . Якщо так вже сталось що автомобіль зламався в дорозі, виникає брак часу та попереду відповідальна зустріч, або ж якась ділова справа, тощо. Замість того, щоб викликати евакуатор, чекати на його прибуття та відправлення до сервісу, вся проблема вирішується в декілька хвилин за допомогою цієї функції. Викликається підмінний автомобіль, це

авто яке є в наявності на випадок, якщо клієнту автосервісу потрібен транспорт для пересування на відносно короткий період(близько тижня), працівник холдингу привозить справне авто з набором необхідних документів, людина від'їжджає далі по своїх справах, а представник компанії викликає евакуатор та чекає його прибуття.

Функція «*Оформлення авто без відвідування сервісу*». Враховуючи людську заклопотаність та квапливість у деяких може виникнути необхідність у обслуговуванні авто, але через брак часу, або зайнятість немає можливості. Для цього було прийнято рішення додати цю функцію, як говорить назва, створюється можливість через додаток замовити обслуговування авто не відвідуючи холдинг. Менеджер прийме авто клієнта в потрібному місці та у відведений час і таким же чином поверне вже готовий до експлуатації транспортний засіб.

Функція «*Замовлення автозапчастин*». Дуже проста та легка функція, замовлення автозапчастин за допомогою додатка, доставка замовлення виконується або на сервіс, або на вказаний адрес, вказані контактні данні консультанта(в разі необхідності). Будь-який тип запчастин, будь-то кузовні частини, або ж деталі підвіски, підбір виконується за Vin-кодом автомобіля, в разі якщо є якась помилка або він не зчитується, є можливість обрати зі списку авто по назві, марці, двигуну та іншим параметрам. Обов'язковою частиною є консультація, не кожна людина розбирається в техніці, та має розуміння і знання конструкції автомобіля, отже потребує допомоги.

Це були три основні функції котрі відрізняють більшість мобільних додатків пов'язаних з автосервісом. Там де є послуга для клієнта, буде прибуток для бізнеса.

РОЗДІЛ 2.

ПРОЄКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ «АВТОСЕРВІС»

2.1 Фреймворк React Native

„**Фреймворк** (англ. *framework* – каркас) – це програмне середовище, яке спрощує та прискорює створення програмного забезпечення. За використання фреймворків ви пишете лише код, який реалізує логіку, специфічну для вашого продукту. Вам не доводиться самостійно забезпечувати роботу з базою даних, автентифікацію, підтримку сеансів тощо” [11].

Є два типи фреймворків : *нативні* та *кросплатформні* або ще називають гібридними. Перші дозволяють створювати додатки лише під певну платформу (наприклад Android або IOS) до плюсів такої розробки можна віднести досить швидку реакцію на дії користувача, можливість мати прямий доступ до апаратної частини і розробити найбільш звичний для користувача конкретної платформи інтерфейс, другі ж пристосовані до розробки на декількох одразу. Створення та підтримка програми нативним методом має досить високу вартість, потребує тривалий час на втілення проєкту в життя, а також підходить лише для одного типу мобільних пристроїв. Стосовно переваг гібридних фреймворків, треба відзначити зручність, необтяжливість, економність як часу так і коштів.

“Зручність полягає в тому, що один раз написаний код буде працювати будь де, а значить – додаток буде комфортний для всієї цільової аудиторії незалежно від їх пристроїв. Необтяжливість заключається в роботі з кодом. Не потрібно міняти одне й те ж декілька разів, мінімальна кількість дій для того, щоб внести якісь зміни. Для бізнеса в пріоритеті менше витрат більше дохід, кросплатформні додатки забезпечують цю умову.”[12]

“**React Native** створений Facebook у 2015 році як продовження React.js для мобільних пристроїв. Спочатку цей фреймворк був тільки для IOS, через що, довгий час IOS був краще оптимізований. Основна мова програмування – Javascript. Має відкритий вихідний код, що в свою чергу стало однією з причин його популярності. Такі популярні додатки як *Instagram, Facebook, Skype, Pinterest* та *Afterpay* були запрограмовані дякуючи React Native.”[13]. Тож давайте переглянемо його переваги :

- *Велика спільнота.* Ком’юніті цього фреймворку має тенденцію зростати та розширюватись, а отже технологія постійно розвивається та підтримується. В разі необхідно не складно знайти відповіді на питання, якісь навчальні матеріали, відео, будь-що, що допоможе вирішити проблему. Звичайно це сприяє полегшенню процесу розробки.
- *Повторне використання коду*. Враховуючи той фактор, що окремі частини коду можуть бути використаними декілька разів на різних платформах, треба відзначити, що це позбавляє необхідності писати все заново, тобто економить час для розробників та кошти для замовника.
- *Стабільність*. Безумовно надійність є великим плюсом, неважливо які і скільки компонентів не було б прив’язано, продуктивність не змінюється. Що в свою чергу забезпечує надійну роботу програм.
- *Продуктивність*. Для роботи програм, які написані за допомогою цієї технології не потрібна велика кількість ресурсів пристрою, для інтерпритації коду програми не потрібні ніякі додаткові пристрої, вони взаємодіють безпосередньо. Завдяки поділу потоків API та UI, Modules та

Controls забезпечується висока продуктивність. У кінцевому результаті програми працюють швидко та ефективно.

- *Підтримка плагінів*. Потрібно відмітити, що не всі функції можуть бути втілені в проєкт за допомогою чистої мови, щоб компенсувати недостачу, встановлюють плагіни, які розширюють можливості ресурсу. React Native підтримує можливість прив'язування сторонніх плагінів, забезпечуючи для розробки ще більше можливостей та перспектив.

Кожен фреймворк є унікальним але водночас і схожим з іншими. Беручи до уваги той факт, що кожен проєкт має свої цілі та функції, необов'язково реалізовувати їх одним і тим самим інструментом, тобто не варто обирати одну технологію на всі випадки життя.

2.2 Мова програмування JavaScript

"**JavaScript** — це динамічна мова комп'ютерного програмування. Він легкий і найчастіше використовується як частина веб-сторінок, реалізації яких дозволяють сценарію на стороні клієнта взаємодіяти з користувачем і створювати динамічні сторінки. Це інтерпретована мова програмування з об'єктно-орієнтованими можливостями."[14].

"Спочатку JavaScript був відомий як LiveScript**,** але Netscape змінив його назву на JavaScript, можливо, через захоплення, викликане Java. JavaScript вперше з'явився в Netscape 2.0 у 1995 році під назвою LiveScript. Ядро мови загального призначення було вбудовано в Netscape, Internet Explorer та інші веб-браузери."[15].

JavaScript на стороні клієнта є найпоширенішою формою мови. Сценарій має бути включений до HTML-документа або посилання на нього для того, щоб код інтерпретувався браузером.

Це означає, що веб-сторінка не обов’язково має бути статичним HTML, але може містити програми, які взаємодіють з користувачем, керують браузером і динамічно створюють вміст HTML.

Клієнтський механізм JavaScript надає багато переваг перед традиційними серверними скриптами CGI. Наприклад, ви можете використовувати JavaScript, щоб перевірити, чи користувач ввів дійсну адресу електронної пошти в поле форми.

Код виконується, коли користувач надсилає форму, і лише якщо всі записи дійсні, вони надсилаються на веб-сервер.

JavaScript можна використовувати для перехоплення ініційованих користувачем подій, таких як натискання кнопок, навігація за посиланням та інші дії, які користувач ініціює явно чи неявно.

Перевагами використання цієї мови є −

- *Менше взаємодії з сервером* – є можливість перевірити введені користувачем дані перед тим, як відправити сторінку на сервер. Це економить трафік сервера, що означає менше навантаження на ваш сервер.
- *Миттєвий зворотний зв’язок з відвідувачами* − їм не потрібно чекати перезавантаження сторінки, щоб побачити, чи не забули вони щось ввести.
- *Підвищена інтерактивність* – є можливість створювати інтерфейси, які реагуватимуть, коли користувач наведе на них мишу або активує їх за допомогою клавіатури.
- *Розширені інтерфейси* – є можливість використовувати JavaScript, щоб включити такі елементи, як компоненти перетягування та скидання та повзунки, щоб надати відвідувачам вашого сайту розширений інтерфейс.

Для цієї мови програмування створено обмеження. Її не можна розглядати JavaScript як повноцінну мову програмування. У ній відсутні наступні важливі функції −

- На стороні клієнта не дозволяє читати або записувати файли. Це збережено з міркувань безпеки.
- Не можна використовувати для мережевих програм, оскільки така підтримка відсутня.
- Не має багатопоточності чи багатопроцесорності.

Знову ж таки, JavaScript — це легка, інтерпретована мова програмування, яка дозволяє вбудовувати інтерактивність у статичні сторінки HTML.

“Однією з головних переваг цієї мови є те, що вона не потребує дорогих інструментів розробки. Початок з простого текстового редактора, такого як блокнот. Оскільки це інтерпретована мова в контексті веб-браузера, навіть не потрібно купувати компілятор.”[16].

Щоб спростити наше життя, різні постачальники придумали дуже гарні інструменти редагування JavaScript. Деякі з них перераховані тут −

- *Microsoft FrontPage* – Microsoft розробила популярний редактор HTML під назвою FrontPage. FrontPage також надає веб-розробникам ряд інструментів JavaScript, які допомагають створювати інтерактивні веб-сайти.
- *Macromedia Dreamweaver MX* − дуже популярний редактор HTML і JavaScript серед професійних веб-розробників. Він надає кілька зручних попередньо зібраних компонентів JavaScript, добре інтегрується з базами даних і відповідає новим стандартам, таким як XHTML і XML.

- *Macromedia HomeSite 5* − HomeSite 5 — популярний редактор HTML і JavaScript від Macromedia, який можна використовувати для ефективного керування особистими веб-сайтами.

2.3. Середовище програмування Vscode

“Visual Studio Code — це безкоштовний, легкий, але потужний редактор вихідного коду, який працює на робочому столі та в Інтернеті та доступний для ОС Windows, macOS, Linux. Він поставляється з вбудованою підтримкою JavaScript , TypeScript і Node.js і має багату екосистему розширень для інших мов програмування (таких як C++, C#, Java, Python, PHP і Go), середовища виконання (таких як .NET і Unity), середовища (такі як Docker і Kubernetes) і хмари (такі як Amazon Web Services, Microsoft Azure і Google Cloud Platform).”[17]

Крім загальної ідеї легкості та швидкого запуску, Visual Studio Code має доповнення коду IntelliSense для змінних, методів та імпортованих модулів, графічне налагодження, лінтування, редагування кількома курсорами, підказки параметрів та інші потужні функції редагування, швидка навігація та рефакторинг коду і вбудований контроль вихідного коду, включаючи підтримку Git. Більшу частину цього було адаптовано з технології Visual Studio.

“Власне код Visual Studio створено з використанням оболонки Electron, Node.js, TypeScript і протоколу Language Server Protocol і оновлюється щомісяця. Багато розширень оновлюються так часто, як це необхідно. Різноманітність підтримки різниться в різних мовах програмування та їх розширеннях, починаючи від простого підсвічування синтаксису та підбору дужок до налагодження та рефакторингу. Ви можете додати базову підтримку вашої улюбленої мови за допомогою розфарбовувачів TextMate, якщо мовний сервер недоступний.”[18].

Архітектура Visual Studio Code

Код Visual Studio складається з багатошарового модульного ядра, яке

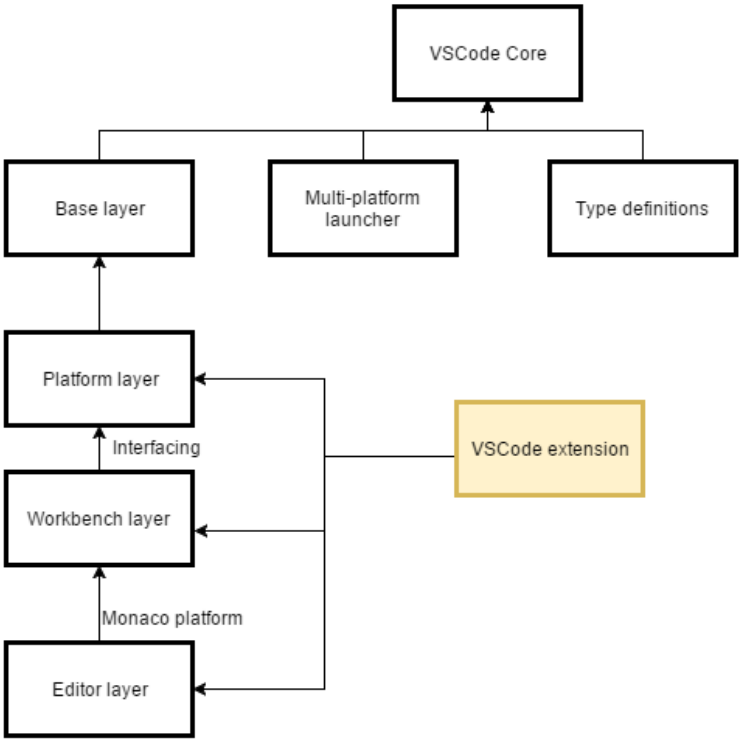


Рисунок 2.1 Архітектура Visual Studio Code

можна розширити за допомогою розширень, як показано на рисунку 2.1. Розширення запускаються в окремому процесі, який називається хостом розширення, який використовує API розширення.

Ядро Visual Studio Code розділено на такі рівні:

- *base*: надає загальні утиліти та будівельні блоки інтерфейсу користувача
- *платформа*: визначає підтримку ін’єкції служби та базові служби для Visual Studio Code

- *редактор*: Редактор «Monaco» доступний як окремий компонент для завантаження
- *workbench*: містить редактор «Monaco» та надає структуру для «віклетів», таких як Explorer, рядок стану або рядок меню, використовуючи Electron для реалізації настільної програми Visual Studio Code. Далі детально описуються шари та пояснюється функціональність кожного

“VSCode включає основний процес і процес візуалізації. У той же час, оскільки він забезпечує можливість розширення плагінів, а також заради безпеки та стабільності, на малюнку є ще один хост розширення. Фактично, цей хост розширення також є незалежним процесом для запуску коду плагіна. І, як і процес візуалізації, вони незалежні один від одного. Процес хосту розширення надає деякі API VSCode для використання розробниками плагінів.”[19].

Таким чином, Visual Studio Code — це швидкий безкоштовний редактор програмування, який підтримує більшість, якщо не весь життєвий цикл розробки програмного забезпечення. VS Code має десятки тисяч плагінів і підтримує сотні мов програмування. Це один із найкращих редакторів коду.

2.4 Контроль версій програмного продукту Git

“**Git** — це безкоштовний інструмент розподіленої системи керування версіями з відкритим вихідним кодом, розроблений для швидкої та ефективної роботи з будь-якими проєктами, від малих до дуже великих. Він був створений Лінусом Торвальдсом у 2005 році для розробки ядра Linux. Git має функціональність, продуктивність, безпеку та гнучкість, необхідні більшості команд та окремих розробників. Він також служить важливим розподіленим інструментом DevOps для контролю версій. Git в основному використовується для керування проєктом, що містить набір коду або текстових файлів, які можуть змінюватися.” [20].

Контроль версій — це керування змінами в документах, комп’ютерних програмах, великих веб-сайтах та інших зборах інформації.

Є два типи VCS:

- Централізована система контролю версій (CVCS)
- Розподілена система керування версіями (DVCS)

Централізована ВКС

Централізована система контролю версій (CVCS) використовує центральний сервер для зберігання всіх файлів і забезпечує командну співпрацю. Він працює в єдиному сховищі, до якого користувачі можуть отримати прямий доступ до центрального сервера.

На рисунку 2.2 зображена діаграма яка дає краще уявлення про

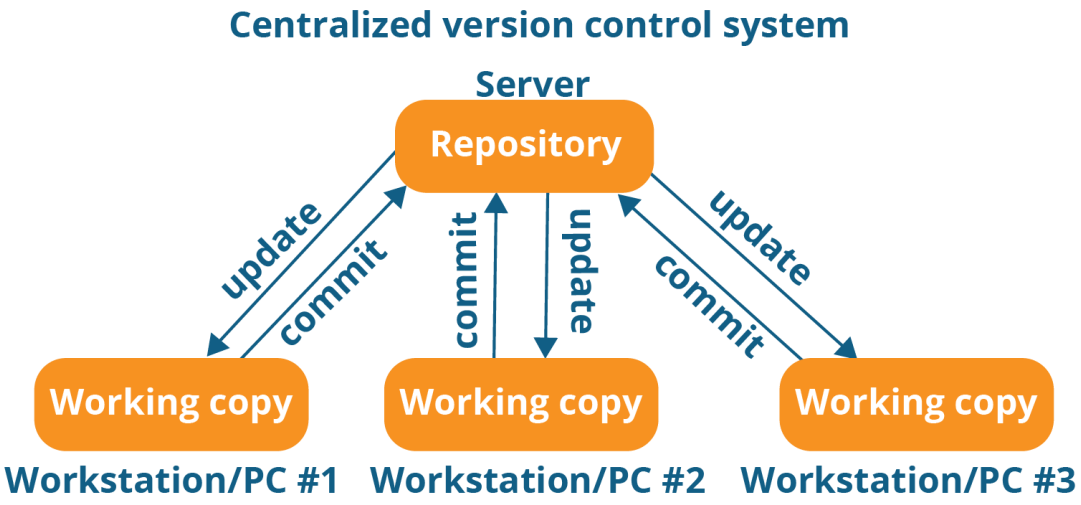


Рисунок 2.2 Діаграма централізованої системи контролю версій

CVCS:

Репозиторій на наведеній вище схемі вказує на центральний сервер, який може бути локальним або віддаленим, безпосередньо підключеним до кожної робочої станції програміста.

Кожен програміст може видобувати або оновлювати свої робочі станції за допомогою даних, наявних у сховищі, або може вносити зміни в дані чи фіксувати в сховищі. Кожна операція виконується безпосередньо в репозиторії.

Незважаючи на те, що здається досить зручним підтримувати єдине сховище, воно має деякі серйозні недоліки. Деякі з них:

- Він недоступний локально; тобто вам завжди потрібно бути підключеним до мережі, щоб виконувати будь-які дії.
- Оскільки все централізоване, у будь-якому випадку збій або пошкодження центрального сервера призведе до втрати всіх даних проєкту.

Розподілений VCS

На рисунку 2.3 зображена діаграма яка дає краще уявлення про

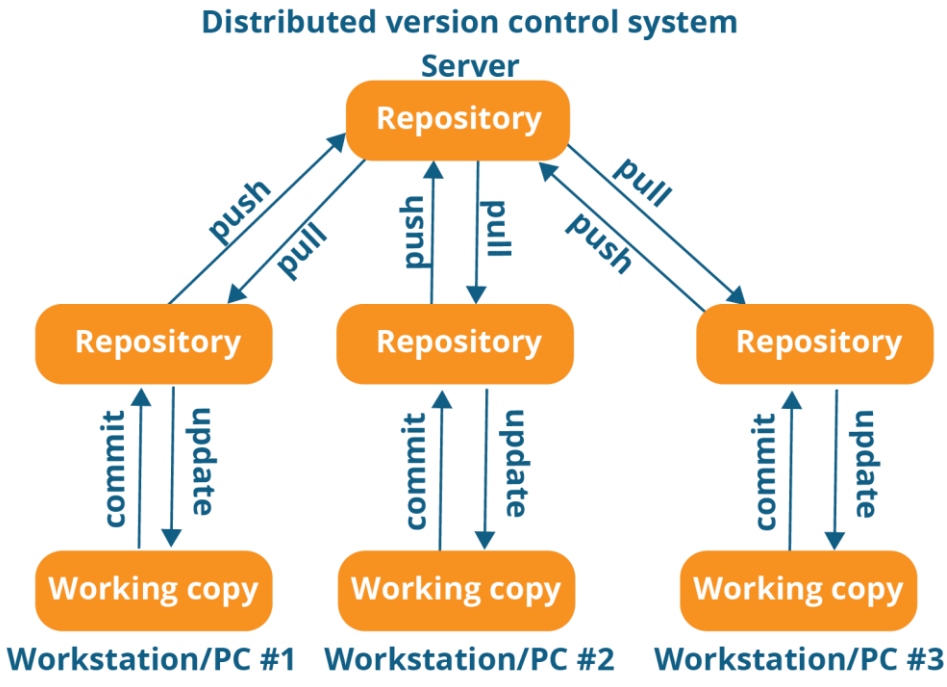


Рисунок 2.3 Діаграма розподіленої системи керування версіями

DVCS:

Ці системи не обов’язково покладаються на центральний сервер для зберігання всіх версій файлу проєкту.

“У розподіленому VCS кожен учасник має локальну копію або «клон» головного сховища, тобто кожен підтримує власний локальний репозиторій, який містить усі файли та метадані, присутні в головному сховищі.” [21].

На наведеній вище діаграмі можна побачити що, кожен програміст самостійно підтримує локальне сховище, яке насправді є копією або клоном центрального сховища на його жорсткому диску. Вони можуть фіксувати та оновлювати свій локальний репозиторій без будь-якого втручання.

Вони можуть оновлювати свої локальні сховища новими даними з центрального сервера за допомогою операції під назвою « pull » і впливати на зміни в основному сховищі за допомогою операції під назвою « push » зі свого локального сховища.

Клонування цілого сховища у робочу станцію для отримання локального сховища надає наступні переваги:

- Усі операції (крім push & pull) виконуються дуже швидко, оскільки інструменту потрібен доступ лише до жорсткого диска, а не до віддаленого сервера. Тому не завжди потрібне підключення до Інтернету.
- Закріплення нових наборів змін може здійснюватися локально без маніпулювання даними в основному сховищі. Коли буде готова група наборів змін, ви можете натиснути їх усі одночасно.

- Оскільки кожен учасник має повну копію репозиторію проєкту, вони можуть ділитися змінами один з одним, якщо хочуть отримати відгук, перш ніж вносити зміни в головне сховище.
- Якщо центральний сервер виходить з ладу в будь-який момент часу, втрачені дані можна легко відновити з будь-якого з локальних сховищ автора.

“Git випущено під ліцензією GPL (Загальна публічна ліцензія) з відкритим кодом. Не потрібно купувати Git. Це абсолютно безкоштовно. І оскільки це відкритий код, можливо змінити вихідний код відповідно до вимог.”[22].

Деякі особливості цієї системи контролю версій і чому саме його використано при розроблені додатку:

- *Швидкість.* Оскільки не потрібно підключатися до жодної мережі для виконання всіх операцій, він виконує всі завдання дуже швидко. Тести продуктивності, проведені Mozilla, показали, що вона на порядок швидша за інші системи контролю версій. Отримання історії версій із локально збереженого сховища може бути в сто разів швидшим, ніж її отримання з віддаленого сервера. Основна частина Git написана мовою C, що дозволяє уникнути накладних витрат, пов’язаних з іншими мовами високого рівня.
- *Масштабований.* Git дуже масштабований. Отже, якщо в майбутньому кількість співавторів збільшиться, він легко впорається з цією зміною. Хоча Git представляє ціле сховище, даних, що зберігаються на стороні клієнта, дуже мало, оскільки Git стискає всі величезні дані за допомогою техніки стиснення без втрат.

- *Надійність.* Оскільки кожен учасник має власне локальне сховище, у разі збою системи втрачені дані можна відновити з будь-якого локального сховища. Завжди буде резервна копія всіх ваших файлів.
- *Безпека.* Git використовує SHA1 (функцію безпечного хешування) для іменування та ідентифікації об’єктів у своєму репозиторії. Кожен файл і коміт підсумовуються та витягуються за його контрольною сумою під час перевірки. Історія Git зберігається таким чином, що ідентифікатор певної версії (коміту в термінах Git) залежить від повної історії розробки, яка веде до цього коміту. Після публікації неможливо змінити старі версії, щоб це не було помічено.
- *Економічний*. У випадку CVCS центральний сервер має бути достатньо потужним, щоб обслуговувати запити всієї команди. Для невеликих команд це не проблема, але зі збільшенням розміру команди апаратні обмеження сервера можуть бути вузьким місцем продуктивності. У випадку DVCS розробники не взаємодіють із сервером, якщо їм не потрібно надіслати або отримати зміни. Уся важка робота виконується на стороні клієнта, тому апаратне забезпечення сервера може бути дуже простим.
- *Підтримує нелінійну розробку.* Git підтримує швидке розгалуження та злиття, а також містить спеціальні інструменти для візуалізації та навігації в історії нелінійної розробки. Основне припущення полягає в тому, що зміни будуть об’єднуватися частіше, ніж вони написані, оскільки вони передаються різним рецензентам. Гілки дуже легкі. Гілка

в Git є лише посиланням на один коміт. З його батьківськими зобов’язаннями можна створити повну структуру гілки.

- *Легке розгалуження*. Керування гілками за допомогою Git дуже просте. Створення, видалення та об'єднання гілок займає всього кілька секунд. Гілки функцій забезпечують ізольоване середовище для кожної зміни кодової бази. Коли розробник хоче почати працювати над чимось, незалежно від того, великим чи малим, він створює нову гілку. Це гарантує, що головна гілка завжди містить код виробничої якості.

2.5 Протокол передачі даних HTTP

“HTTP — це протокол для отримання ресурсів, наприклад документів HTML. Він є основою будь-якого обміну даними в Інтернеті та є протоколом клієнт-сервер, піддокументів, наприклад, тексту, опису макета, зображень, відео, сценаріїв тощо. Клієнти та сервери спілкуються шляхом обміну окремими повідомленнями (на відміну від потоку даних). Повідомлення, які надсилає клієнт, як правило, веб-браузер, називаються запитами , а повідомлення, які надсилає сервер як відповідь, називаються відповідями, що означає, що запити ініціюються одержувачем, зазвичай веб-браузером.”[23]

Розроблений на початку 1990-х років HTTP є розширюваним протоколом, який з часом розвивався. Це протокол прикладного рівня, який надсилається через TCP або через TCP-з’єднання, зашифроване TLS , хоча теоретично можна використовувати будь-який надійний транспортний протокол. Завдяки своїй розширюваності він використовується не лише для отримання гіпертекстових документів, але й для зображень і відео або для публікації вмісту на серверах, як із результатами HTML-форм. HTTP також

можна використовувати для отримання частин документів для оновлення веб-сторінок на вимогу. Компоненти систем на базі HTTP

“HTTP — це клієнт-серверний протокол: запити надсилаються однією сутністю, агентом користувача (або проксі-сервером від його імені). У більшості випадків агентом користувача є веб-браузер, але це може бути що завгодно, наприклад, робот, який сканує Інтернет, щоб заповнити та підтримувати індекс пошукової системи. Кожен окремий запит надсилається на сервер, який обробляє його та надає відповідь, яка називається відповіддю . Між клієнтом і сервером є численні об’єкти, які разом називаються проксі -серверами , які виконують різні операції та діють, наприклад, як шлюзи або кеші .”[24].

Насправді між браузером і сервером, що обробляє запит, є більше комп’ютерів: є маршрутизатори, модеми тощо. Завдяки багаторівневому дизайну Інтернету вони приховані в мережевому та транспортному рівнях. HTTP знаходиться зверху, на прикладному рівні. Хоча базові рівні важливі для діагностики мережевих проблем, вони здебільшого не мають відношення до опису HTTP.

“Агент користувача - це будь-який інструмент, який діє від імені користувача. Цю роль в основному виконує веб-браузер, але її також можуть виконувати програми, які використовуються інженерами та веб-розробниками для налагодження своїх програм.”[25].

Браузер завжди ініціює запит. Це ніколи не сервер (хоча протягом багатьох років було додано деякі механізми для імітації ініційованих сервером повідомлень).

Щоб відобразити веб-сторінку, браузер надсилає оригінальний запит для отримання HTML-документа, який представляє сторінку. Потім він аналізує цей файл, роблячи додаткові запити, що відповідають сценаріям

виконання, інформації про макет (CSS) для відображення та підресурсів, що містяться на сторінці (зазвичай зображення та відео). Потім веб-браузер поєднує ці ресурси, щоб представити повний документ, веб-сторінку. Сценарії, які виконує браузер, можуть отримати більше ресурсів на наступних етапах, і браузер відповідно оновлює веб-сторінку.

“**Веб-сторінка** — це гіпертекстовий документ. Це означає, що деякі частини відображуваного вмісту є посиланнями, які можна активувати (зазвичай клацанням миші) для отримання нової веб-сторінки, що дозволяє користувачеві керувати своїм агентом користувача та переміщатися по мережі. Браузер перетворює ці вказівки в HTTP-запити, а потім інтерпретує відповіді HTTP, щоб надати користувачеві чітку відповідь.”[26].

“**Веб-сервер.** На протилежній стороні каналу зв'язку знаходиться сервер, який обслуговує документ за запитом клієнта. Сервер віртуально виглядає лише як одна машина; але насправді це може бути сукупність серверів, які розподіляють навантаження (балансування навантаження), або складне програмне забезпечення, яке опитує інші комп’ютери (наприклад, кеш-пам’ять, сервер БД або сервери електронної комерції), повністю або частково генеруючи документ на вимогу.”[27].

“**Сервер** – це не обов’язково одна машина, але на одній машині може бути розміщено кілька примірників серверного програмного забезпечення. За допомогою HTTP/1.1 і Host заголовка вони можуть навіть мати однакову IP-адресу.”[28].

2.6 REST

“**REST, або REpresentational State Transfer,** — це архітектурний стиль для забезпечення стандартів між комп’ютерними системами в Інтернеті, що полегшує між системами взаємодію. Системи, сумісні з

REST, які часто називають системами RESTful, характеризуються тим, що вони не мають стану та розділяють проблеми клієнта та сервера.”[29].

Розділення клієнта і сервера.

В архітектурному стилі REST реалізацію клієнта та реалізацію сервера можна виконувати незалежно один від одного, не знаючи про інше. Це означає, що код на стороні клієнта можна змінити в будь-який час, не впливаючи на роботу сервера, а код на стороні сервера можна змінити, не впливаючи на роботу клієнта.

Поки кожна сторона знає, який формат повідомлень надсилати іншій, вони можуть бути модульними та окремими. Відокремлюючи інтерфейс користувача від проблем зберігання даних, покращується гнучкість інтерфейсу між платформами та масштабованість за рахунок спрощення компонентів сервера. Крім того, поділ дозволяє кожному компоненту розвиватися незалежно.

Використовуючи інтерфейс REST, різні клієнти потрапляють на ті самі кінцеві точки REST, виконують однакові дії та отримують однакові відповіді.

Відсутність стану.

“Системи, які дотримуються парадигми REST, не мають стану, тобто серверу не потрібно нічого знати про те, в якому стані перебуває клієнт, і навпаки. Таким чином і сервер, і клієнт можуть зрозуміти будь-яке отримане повідомлення, навіть не переглядаючи попередні повідомлення. Це обмеження відсутності громадянства забезпечується за допомогою використання ресурсів , а не команд. Ресурси — це іменники Інтернету — вони описують будь-який об’єкт, документ або річ , які можуть знадобитися зберегти або надіслати іншим службам.”[30].

Оскільки системи REST взаємодіють через стандартні операції над ресурсами, вони не покладаються на реалізацію інтерфейсів.

Ці обмеження допомагають програмам RESTful досягти надійності, швидкої продуктивності та масштабованості як компонентів, якими можна керувати, оновлювати та повторно використовувати, не впливаючи на систему в цілому, навіть під час роботи системи.

REST вимагає, щоб клієнт зробив запит на сервер, щоб отримати або змінити дані на сервері. Запит зазвичай складається з:

- дієслово HTTP, яке визначає, яку операцію виконувати;
- заголовок , який дозволяє клієнту передавати інформацію про запит;
- шлях до ресурсу;
- додаткове тіло повідомлення, що містить дані;

2.7 Клієнт-сервер

“Модель «Клієнт-сервер» — це розподілена структура програми, яка розподіляє завдання або робоче навантаження між постачальниками ресурсів або послуг, які називаються серверами, і запитувачами послуг, які називаються клієнтами. В архітектурі клієнт-сервер, коли клієнтський комп’ютер надсилає запит на дані серверу через Інтернет, сервер приймає запитуваний процес і доставляє запитувані пакети даних назад клієнту. Клієнти не діляться своїми ресурсами. Прикладами клієнт-серверної моделі є електронна пошта, Всесвітня павутина тощо.”[31].

Коли говорять слово « Клієнт », це означає людину чи організацію, яка використовує певну послугу. Подібним чином у цифровому світі клієнт — це комп’ютер (хост), тобто здатний отримувати інформацію або використовувати певну послугу від постачальників послуг (сервери).

Подібним чином, коли ми говоримо слово « Сервери », воно означає особу або засіб, який щось обслуговує. У цьому цифровому світі сервер —

це віддалений комп'ютер, який надає інформацію (дані) або доступ до певних послуг.

Отже, в основному Клієнт щось запитує, а Сервер обслуговує його, поки він присутній у базі даних.

Щоб взаємодіяти з серверами клієнта, потрібно виконати кілька кроків.

- Користувач вводить URL (уніфікований покажчик ресурсу) веб-сайту або файлу. Потім браузер запитує сервер DNS (СИСТЕМА ДОМЕННИХ ІМЕН).
- Пошук на сервері DNS для адреси веб-сервера .
- DNS-сервер відповідає IP-адресою веб- сервера .
- Браузер надсилає запит HTTP/HTTPS на IP-адресу ВЕБ-сервера (надається DNS-сервером).
- Сервер надсилає необхідні файли сайту.
- Потім браузер відтворює файли, і веб-сайт відображається.

Ця візуалізація виконується за допомогою інтерпретатора DOM (об'єктної моделі документа), інтерпретатора CSS і механізму JS , спільно відомого як JIT або компілятор (Just in Time).

Переваги клієнт-серверної моделі:

- Централізована система з усіма даними в одному місці.
- Рентабельність вимагає менших витрат на технічне обслуговування та можливе відновлення даних.
- Ємність Клієнта та Серверів можна змінювати окремо.

Недоліки клієнт-серверної моделі:

- Клієнти схильні до вірусів, троянів і хробаків, якщо вони присутні на сервері або завантажені на сервер.

- Сервери схильні до атак типу «Відмова в обслуговуванні» (DOS).
- Пакети даних можуть бути підроблені або змінені під час передачі.
- Фішинг або захоплення облікових даних для входу чи іншої корисної інформації користувача є звичайним явищем.

РОЗДІЛ 3.

ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Архітектура додатку

„Щоб мобільний застосунок був успішним, необхідно придумати всі деталі під час його розробки. Маркетинг та просування програми – це лише мала частина. Все починається з ідеї, яка оформляється у зрозумілий вигляд. Зробити цей вид, зрозуміти, як працюватиме сервіс, побачити всі процеси до початку їхньої розробки допомагає архітектура мобільних застосунків. Її головна мета – подальша економія та обслуговування програми, створення спочатку якісного та робочого продукту.

Коли користувач вводить у пошуковому рядку браузера адресу веб-програми, він запитує сервера. З цього моменту він взаємодіє з програмою, тому що сервер відповідає і показує сторінку. Користувач бачить інтерфейс, натискає на кнопки, переходить сторінками, вводить якісь дані у поля. У цей момент на серверній частині виконуються завдання, а відповіді відображаються в інтерфейсі користувача. Усі ці компоненти формують архітектуру.” [32].

“Архітектура застосунків – це сукупність низки рішень та дій, що дозволяють організувати коректну роботу програми. До неї входять як інтерфейси, і структурні елементи, база даних, стилі та інші компоненти.

Добре побудована архітектура має відповідати наступним критеріям:

- бути ефективною та виконувати свої функції, незалежно від навантаження на сервер;
- залишатися гнучкою і мати можливість розширення, щоб за необхідності можна було додати функції;
- легко тестуватися, що дозволить збільшити надійність та зменшити час на перевірку та впровадження нових функцій;

- бути зрозумілою та структурованою, тому що над одним проєктом може працювати велика кількість людей, і новачки повинні одразу розуміти, як влаштовано програму. ” [33]

“Клієнтська та серверна частини мобільного застосунку розробляються на базі різних мов програмування та фреймворків, але це не мусить заважати їх взаємодії. Сервер надає користувачеві можливість взаємодіяти з базою даних, в якій знаходяться персональні дані клієнтів даної системи і т.і.” [34].

У проєктуванні додатку було використано трирівневу архітектуру клієнт-сервер, яка складається з рівня презентації, відомого як рівень інтерфейсу користувача, рівня додатків, який називається рівнем обслуговування, і рівня даних, що включає сервер бази даних. Трирівневу архітектуру можна розділити на три частини:

- Рівень презентації (або рівень клієнта): цей рівень піклується про інтерфейс користувача.
- Прикладний рівень (або бізнес-рівень): цей рівень забезпечує детальну обробку.
- Рівень бази даних (або рівень даних): цей рівень зберігає інформацію.

Клієнтська система контролює рівень презентації; сервер додатків піклується про рівень додатків, а система сервера контролює рівень бази даних. Така архітектура має свої переваги та недоліки.

Розглянемо спочатку переваги:

- Це централізована система, яка зберігає всі дані та елементи керування в одному місці.

- Це забезпечує високий рівень масштабованості, організації та ефективності.
- Це дозволяє IT-персоналу окремо змінювати потужності клієнта та сервера.
- Це економічно вигідно, особливо з точки зору обслуговування.
- Це дозволяє відновлювати дані.
- Це дозволяє балансувати навантаження, що оптимізує продуктивність.
- Це дозволяє різним платформам обмінюватися ресурсами.
- Користувачам не потрібно входити в термінал або інший процесор, щоб отримати доступ до корпоративної інформації або настільних інструментів, таких як презентатори PowerPoint або утиліти для роботи з електронними таблицями.
- Налаштування зменшує випадки реплікації даних

Стосовно недоліків потрібно визначити, що:

- Якщо на сервері є хробак, вірус або троян , користувачі, ймовірно, підхоплять його, оскільки мережа складається з пов’язаних клієнтів і серверів.
- Сервер вразливий до атак типу «Відмова в обслуговуванні» (DoS).
- Пакети даних можуть бути підроблені або змінені під час передачі
- Це дорого для запуску та початкового впровадження
- Якщо критично важливий сервер виходить з ладу, клієнти мертві у воді
- Налаштування схильне до фішингу та атак «Людина посередині» (MITM).

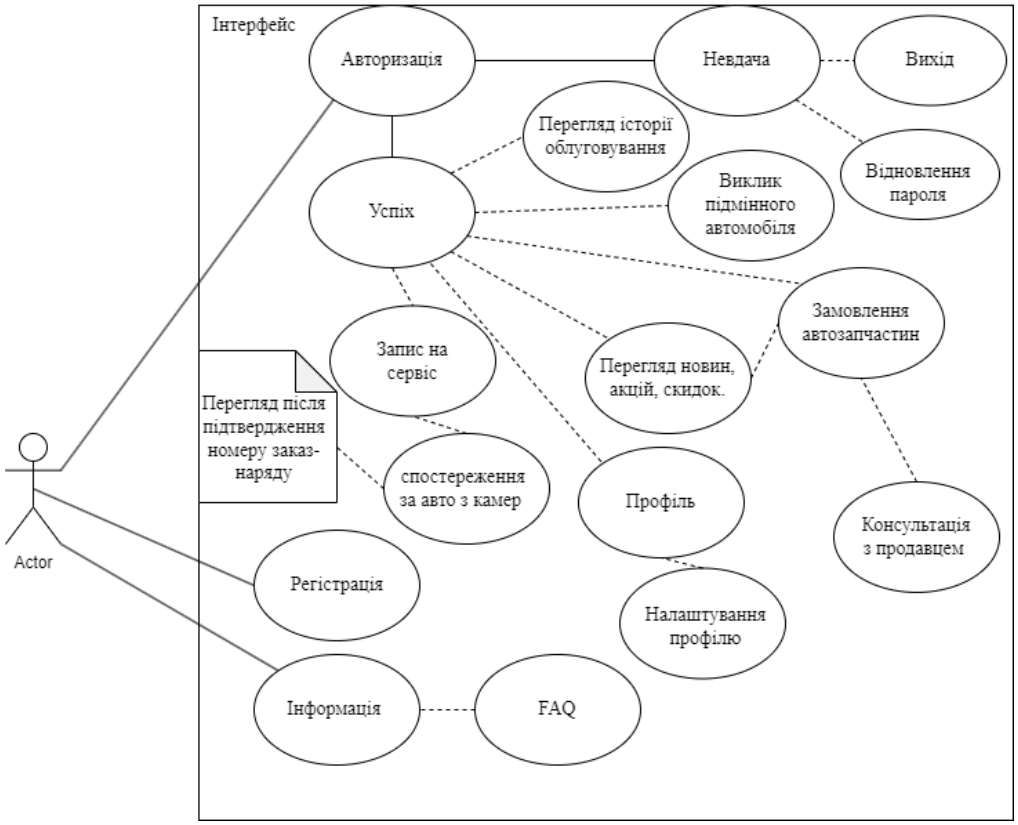
“Отже, основна ідея архітектури «клієнт-сервер» полягає в поділі мережевого додатку на кілька компонентів, кожен з яких реалізує специфічний набір сервісів. Компоненти такого додатку можуть виконуватися на різних комп’ютерах, виконуючи серверні або клієнтські функції. Це дозволяє підвищити надійність, безпеку і продуктивність мережевих додатків і мережі в цілому.” [35].

3.2 Діаграма варіантів використання

„Діаграми варіантів використання представляють собою графічне представлення взаємодії користувача і комп'ютерної системи. Кожен варіант використання охоплює деяку очевидну для користувачів функцію системи і вирішує деяку дискретну задачу користувача.” [36].

Побудуємо діаграму виділивши при цьому основні дії користувача в системі. (рис.3.1)

Рисунок 3.1 Use case діаграма користувача з інтерфейсом.



“Діаграма варіантів використання – діаграма, на якій відображені відносини, що існують між акторами і варіантами використання (прецедентами). Основна ідея Use Case діаграми – надати узагальнений вид системи, що дає можливість замовнику, кінцевому користувачеві і розробнику спільно обговорювати функціональність і поведінку системи.” [37]

Діаграма показує взаємодію користувача з інтерфейсом мобільного додатку «Автосервіс». Користувач має змогу продивитись інформацію про додаток та отримати відповіді на найпоширеніші запитання. Також є можливість зареєструватись. Ну і звичайно авторизація, якщо процес успішний, згідно діаграми користувачі можуть:

- Переглянути історію обслуговування свого автомобіля в автосервісі.(Роботи які були виконані, дата виконання, запчастини які були використання, ціна за запчастини, роботу, усього)

- Використання функції підмінний автомобіль.
- Замовлення автозапчастин.(У разі необхідності консультація з продавцем)
- Перегляд новин, акцій, скидок.
- Перегляд профілю та його редагування.
- Онлайн запис на сервіс, а після підтвердження заказ наряду, можливий перегляд камер спостереження.

При негативному результаті авторизації доступний вихід, або запуск процесу відновлення пароля.

3.3 Діаграма розгортання

„Діаграма розгортання — діаграма на якій 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; натомість, їх можна відобразити на діаграмах компонентів. Діаграма розгортання відображає робочі екземпляри компонентів, а діаграма компонентів, натомість, відображає зв'язки між типами компонентів.” [38].

Діаграма розгортання в UML моделює фізичне розгортання артефактів на вузлах.

“Вузли представляються, як прямокутні паралелепіпеди з артефактами, розташованими в них, зображеними у вигляді прямокутників. Вузли можуть мати підвузли, які представляються, як вкладені прямокутні паралелепіпеди. Один вузол діаграми розгортання може концептуально представляти безліч фізичних вузлів, таких, як кластер серверів баз даних.”[39]

Діаграма розгортання для розроблюваної системи мобільного додатку представлена на Рисунку 3.2.

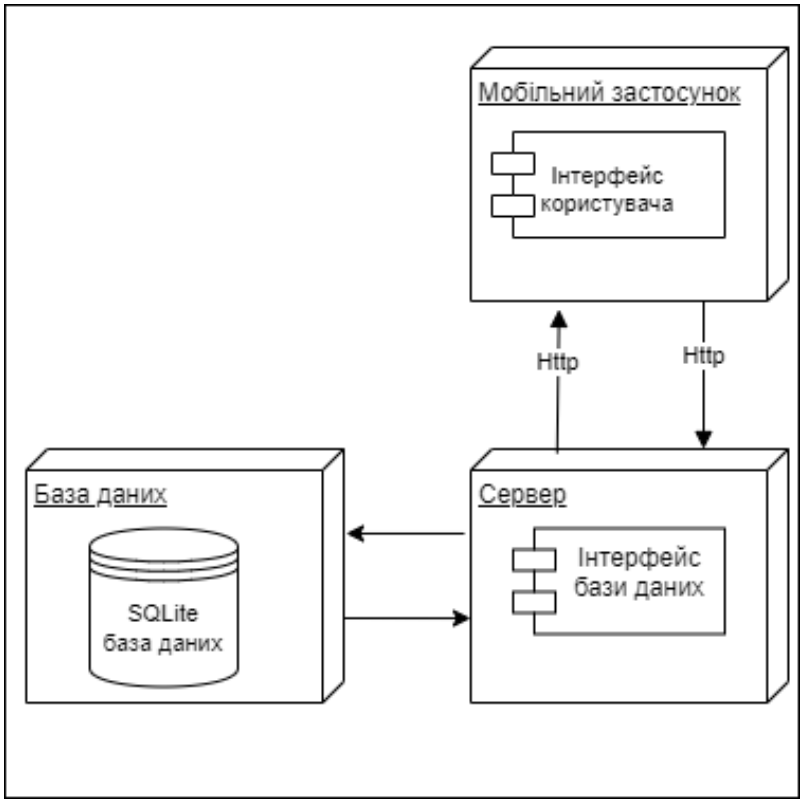


Рисунок 3.2 Діаграма розгортання мобільного застосунку

“Діаграму розгортання системи можна описати наступним чином. Користувач взаємодіє із системою шляхом використання інтерфейсу мобільного додатку. Програма відправляє запити та отримує відповіді від сервера. Сервер надає інтерфейс бази даних, для доступу до даних, їх зміни, видалення, модифікації.” [40].

ВИСНОВКИ

Виконання кваліфікаційної роботи дало можливість сформулювати наступні висновки.

1. Розглянуто мобільні додатки в Україні та світі. Яскравими прикладами аналогічних або схожих мобільних додатків є «Autodoc», «Express Auto Service & Repair», «Simply Auto: Car Maintenance», а також український додаток «Автосервіс PRO.CTO.UA». У відповідному розділі

було описано їхні функції. Проаналізувавши велику кількість додатків в play market та App Store, необхідно відмітити, що у кожного додатку різний набір функціоналу, отже знайти і підібрати який загальний функціонал не можливо. Виходячи з вище сказаного, можна розділити функціонал на дві категорії, стандартний та індивідуальний, це спростить проєктування та розробку мобільного додатку.

2. Проаналізувавши функції програм аналогів, виділивши відмінні функції було визначено функціональні вимоги серед яких : вхід в систему, підмінний автомобіль, оформлення та прийняття авто без відвідування автосервісу, перегляд камер спостереження, замовлення автозапчастин, перегляд новин (скидок, акцій), запис на сервіс, перегляд історії обслуговування, підключення платіжної системи.

3. Визначившись з функціональними вимогами, наступним етапом стало проєктування мобільного додатку. При проєктуванні для розробки додатку було вирішено скористатись середовищем програмування Visual Studio Code. Для системи було використано фреймворк React Native на мові програмування JavaScript. Впроваджено до роботи систему контролю версій Git.

4. У ході розробки, було використано трирівневу архітектуру клієнт-сервер, яка складається з рівня презентації, відомого як рівень інтерфейсу користувача, рівня додатків, який називається рівнем обслуговування, і рівня даних, що включає сервер бази даних.

5. Розпочато створення та розробку мобільного додатку «Автосервіс». В подальшому після закінчення розробки, буде проведено ряд тестувань. Також одним із варіантів подальшого розвитку програми, є розвиток архітектури у веб середовище, та розширення його функціональності, як програмної так і апаратної.

Підсумовуючи вище сказане, завдання магістерської роботи виконані. Був проведений пошук та аналіз інформації, знайдені відповідні рішення розробки мобільного додатку. Спроєктовано та продемонстровано архітектуру додатку, діаграми розгортання та компонентів. Продумано та створено зручний та простий дизайн інтерфейсу, на його основі відбулась подальша розробка програмного коду front-end частини.

Було покращено навички роботи з мовою програмування javascript. Отримано досвід роботи з фреймворком React Native.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Техопедія. Що таке мобільний додаток? [Електронний ресурс] / Техопедія – Режим доступу до ресурсу: https://uk.theastrologypage.com/mobile-application.

2. Borysenko D. ВИКОРИСТАННЯ МОБІЛЬНИХ ДОДАТКІВ ПРИ РОЗРОБЛЕНІ ДИЗАЙН-ПРОДУКТУ У НАВЧАННІ МАЙБУТНІХ ФАХІВЦІВ З ДИЗАЙНУ [Електронний ресурс] / Denys Borysenko. Українська інженерно-педагогічна академія, м. Харків, Україна. – 2018. – Режим доступу до ресурсу: https://www.researchgate.net/publication/331336443_VIKORISTANNA_MOBILNIH_DODATKIV_PRI_ROZROBLENI_DIZAJN-PRODUKTU_U_NAVCANNI_MAJBUTNIH_FAHIVCIV_Z_DIZAJNU.

3. Функціональні та нефункціональні вимоги [Електронний ресурс] – Режим доступу до ресурсу: https://visuresolutions.com/uk/blog/functional-requirements/.

4. Денисенко А. Фреймворки у веб-розробці — що це, які існують і для чого потрібні [Електронний ресурс] / Андрій Денисенко. – 2022. – Режим доступу до ресурсу: https://highload.today/uk/frejmvorki-u-veb-rozrobtsi-shho-tse-yaki-isnuyut-i-dlya-chogo-potribni/.

5. Архітектура мобільного застосунку [Електронний ресурс]. IT компанія Wezom, 2022. – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/arhitektura-mobilnogo-prilozheniya.

6. Мобільний додаток підтримки ведення нотаток путівника [Електронний ресурс] – Режим доступу до ресурсу: https://prog.bobrodobro.ru/34389.

7. Діаграма розгортання [Електронний ресурс],2022. Режим доступу до ресурсу: https://studfile.net/preview/5010027/page:6/.

8. Ян Ф. Дарвин. Android Cookbook: Problems and Solutions for Android Developers, 2nd Edition / Ян Ф. Дарвин., 2018. 768 с.

9. Герберт Шільдт. Java та посібник для початківців / Герберт Шільдт..

10. Нахавандипур В. iOS. Прийоми програмування / Нахавандипур В.., 2014.832 с.

11. Swift: розробка додатків в середовищі Xcode для iPhone і iPad з використанням iOS SDK / Девід Марк, Джек Наттінг, Кім Топл, Фредрік Олссон., 2015. 816 с.

12. Paul Hudson. Hacking with Swift / Paul Hudson., 2016. 1719 с.

13. Типи мобільних додатків. [Електронний ресурс] — Режим доступу: https://smile-ukraine.com/ua/mobile-apps/mobile-apps-types

14. Разработка под iOS и Android: рейтинг языков программирования 2020 [Електронний ресурс] — Режим доступу: https://apptractor.ru/reyting-yazyikovprogrammirovaniya-2020

15. Flutter — новый взгляд на кроссплатформенную разработку. [Електронний ресурс] — Режим доступу: https://habr.com/ru/company/google/blog/426701/

16. Developer Survey. [Електронний ресурс] — Режим доступу: https://insights.stackoverflow.com/survey/2020#most-popular-technologies

17. Elman J. Lightweight Django: Using REST, WebSockets, and Backbone [Електронний ресурс] / J. Elman, M. Lavin // O'Reilly Media, Inc., 2014. 256 с. – Режим доступу: https://doc.lagout.org/programmation/Django/Lightweight%20Django_%20Using%20RE

18. Поняття архітектури програмного забезпечення. [Електронний ресурс]. — Режим доступу: http://www.lib.mdpu.org.ua/e-book/web/Lec10.html

19. Сравнение многоуровневых и клиент-серверных систем [Електронний ресурс]. — Режим доступу: http://msdn.microsoft.com/ru-ru/library/ee895340.aspx

20. Android developers [Електронний ресурс]. — Режим доступу: http://developer.android.com/reference/android/speech/tts/TextToSpeech.html

21. Android software development. Вільна енциклопедія [Електронний ресурс]. — Режим доступу: http://en.wikipedia.org/wiki/Android_software_development#cite_note-11

22. Діаграма послідовності – Вікіпедія [Електронний ресурс]: − портал wikipedia.org − Режим доступу: https://uk.wikipedia.org/wiki/Діаграма_послідовності/

23. Мова програмування Kotlin [Електронний ресурс]: − портал https://kotlinlang.org− Режим доступу: https://kotlinlang.org/docs/reference/

24. Фримен Э. Паттерны проектирования [Текст] – Пер. с англ. / Фримен Э., Сьерра К., Бейтс Б. – СПб.: Питер, 2011. 656 с.

25. ФаулерМ. Архитектура корпоративных программных приложений [Текст]/ Фаулер М., Райс Д.: Вильямс, 2007. 544 с.

26. Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста [Текст]/ Мартин Р. – Спб.: Питер, 2010. 464с.

27. Пацей Н.В. Разработка мобильных приложений / Н.В. Пацей. – Минск: БГТУ, 2020. 266 с.

28. Макконнелл Стив. Досконалий код. Практичне керівництво по розробці програмного забезпечення / Макконнелл Стив., 2019. 896 с.

29. Glas G. Web app vs. Native app [Електронний ресурс] / Grant Glas. https://www.app-press.com/ – Режим доступу до ресурсу: https://www.apppress.com/blog/web-app-vs-native-app.

30. Web application [Електронний ресурс] – Режим доступу до ресурсу: https://www.pcmag.com/encyclopedia/term/web-application.

31. Native App vs. Mobile Web App: A Quick Comparison [Електронний ресурс] – Режим доступу до ресурсу: https://www.webfx.com/blog/webdesign/native-app-vs-mobile-web-app-comparison/.

32. Основы создания приложений [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/guide/components/fundamentals?hl=ru.

33. Bonnie Eisenman. Learning React Native: Building Native Mobile Apps with JavaScript / Bonnie Eisenman. 240 с.

34. Nader Dabit. React Native in Action Developing iOS and Android apps with JavaScript / Nader Dabit., 2019. 320 с.

35. Скотт Шакон. Pro Git — профессиональный контроль версий / Скотт Шакон., 2014. 286 с.

36. Ben Lynn. Волшебство Git / Ben Lynn., 2007. 50 с.

37. Поллард Б. HTTP/2 в действии / Поллард Б.., 2021. 424 с.

38. Brian Mulloy. Web API Design: Crafting Interfaces that Developers Love / Brian Mulloy. 38 с.

39. Сем Ньюмен. Building Microservices: Designing Fine-Grained Systems / Сем Ньюмен., 2021. 586 с.

40. Основи REST [Електронний ресурс] – Режим доступу до ресурсу: http://edu.asu.in.ua/mod/book/view.php?id=117&chapterid=256.

Додатки

КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІЗДОБУВАЧА ВИЩОЇ ОСВІТИ ХЕРСОНСЬКОГО ДЕРЖАВНОГО УНІВЕРСИТЕТУ

Я, Подмазко Євгеній Сергійович,

учасник(ця) освітнього процесу Херсонського державного університету, **УСВІДОМЛЮЮ**, що академічнадоброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

ЗАЯВЛЯЮ, що у своїй освітній і науковій діяльності **ЗОБОВ'ЯЗУЮСЯ**:

— дотримуватися:

- вимог законодавства України та внутрішніх нормативних документів університету, зокрема СтатутуУніверситету;
- принципів та правил академічної доброчесності;
- нульової толерантності до академічного плагіату;
- моральних норм та правил етичної поведінки;
- толерантного ставлення до інших;
- дотримуватися високого рівня культури спілкування;

— надавати згоду на:

- безпосередню перевірку курсових, кваліфікаційних робіт тощо на ознаки наявності академічного плагіату задопомогою спеціалізованих програмних продуктів;
- оброблення, збереження й розміщення кваліфікаційних робіт у відкритому доступі в інституційномурепозитарії;
- використання робіт для перевірки на ознаки наявності академічного плагіату в інших роботах виключно зметою виявлення можливих ознак академічного плагіату;

— самостійно виконувати навчальні завдання, завдання поточного й підсумкового контролю результатів навчання;

— надавати достовірну інформацію щодо результатів власної навчальної (наукової, творчої) діяльності,використаних методик досліджень та джерел інформації;

— не використовувати результати досліджень інших авторів без використання покликань на їхню роботу;

— своєю діяльністю сприяти збереженню та примноженню традицій університету, формуванню його позитивногоіміджу;

— не чинити правопорушень і не сприяти їхньому скоєнню іншими особами;

— підтримувати атмосферу довіри, взаємної відповідальності та співпраці в освітньому середовищі;

— поважати честь, гідність та особисту недоторканність особи, незважаючи на її стать, вік, матеріальний стан,соціальне становище, расову належність, релігійні й політичні переконання;

— не дискримінувати людей на підставі академічного статусу, а також за національною, расовою, статевою чиіншою належністю;

— відповідально ставитися до своїх обов'язків, вчасно та сумлінно виконувати необхідні навчальні танауководослідницькі завдання;

— запобігати виникненню у своїй діяльності конфлікту інтересів, зокрема не використовувати службових іродинних зв'язків з метою отримання нечесної переваги в навчальній, науковій і трудовій діяльності;

— не брати участі в будь-якій діяльності, пов'язаній із обманом, нечесністю, списуванням, фабрикацією;

— не підроблювати документи;

— не поширювати неправдиву та компрометуючу інформацію про інших здобувачів вищої освіти, викладачів іспівробітників;

— не отримувати і не пропонувати винагород за несправедливе отримання будь-яких переваг або здійсненнявпливу на зміну отриманої академічної оцінки;

— не залякувати й не проявляти агресії та насильства проти інших, сексуальні домагання;

— не завдавати шкоди матеріальним цінностям, матеріально-технічній базі університету та особистій власностіінших студентів та/або працівників;

— не використовувати без дозволу ректорату (деканату) символіки університету в заходах, не пов'язаних здіяльністю університету;

— не здійснювати і не заохочувати будь-яких спроб, спрямованих на те, щоб за допомогою нечесних і негіднихметодів досягати власних корисних цілей;

— не завдавати загрози власному здоров'ю або безпеці іншим студентам та/або працівникам.

УСВІДОМЛЮЮ, що відповідно до чинного законодавства у разі недотримання Кодексу академічної доброчесності буду нести академічну та/або інші види відповідальності й до мене можуть бути застосовані заходи дисциплінарного характеру за порушення принципів академічної доброчесності

13.12.2022 Євгеній Подмазко

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Рецензія на кваліфікаційну роботу (проєкт)

Навчальний рік: 2021-2022

Факультет: комп'ютерних наук, фізики та математики

Спеціальність: 121 Інженерія програмного забезпечення

Освітньо-професійна (наукова) програма «Інженерія програмного забезпечення»

Форма навчання: денна

Ступінь вищої освіти: другого (магістерського) рівня освіти

Тема: Проєктування та розробка мобільного додатку «Автосервіс»

Виконавець: Подмазко Євгеній Сергійович

Зміст рецензії:

Кваліфікаційна робота на здобуття освітньо-кваліфікаційного рівня магістр на тему «Проєктування та розробка мобільного додатку Автосервіс»

Робота виконана згідно визначеної теми, завдання визначені чітко та зрозуміло. Магістерська робота складається зі вступу, трьох розділів, висновку та списку використаних джерел. У роботі чітко поставленні цілі, а також мета, завдання, об'єкт і предмети дослідження. Під час написання магістерської роботи, автором було досліджено та опрацьовано значний обсяг матеріалу, завдяки великій кількості джерел, наукових статей, книг та веб-сайтів. Роботу було передано до перевірки на унікальність у системі Unicheck, де було показано рівень унікальності 100%. Робота логічно структурована, складається з теоретичної та практичної частин. Теоретична частина викладена поступово. Для даної роботи специфічні деякі огріхи.

Робота Подмазко Є.С. відповідає вимогам, що пред'являються до такого виду робіт та може бути допущена до захисту.

Рецензент д.пед.н., проф. Мелітопольського державного педагогічного університету
Круглик Владислав Сергійович

Дата

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Відгук наукового керівника

на кваліфікаційну роботу (проєкт)

Навчальний рік: 2021-2022

Факультет: комп'ютерних наук, фізики та математики Спеціальність: 121 Інженерія програмного забезпечення

Освітньо-професійна (наукова) програма: «Інженерія програмного забезпечення»Форма навчання: денна

Ступінь вищої освіти: другого (магістерського) рівня освіти

Тема: Розробка та проєктування мобільного додатку «Автосервіс»Виконавець: Подмазко Євгеній Сергійович

Зміст відгуку:

Кваліфікаційна робота студента Подмазко Є.С. присвячена розробці та проєктуванні мобільного додатку, дослідженню та аналізу проблем, полегшенню процесу обслуговування автомобілів. Актуальність полягає успрощенню відвідування автосервісу, шляхом залучення мобільного додатку.

Автором отримані наступні результати:

1) розглянуто мобільні додатки у світі та Україні;
2) визначено функціональні вимоги мобільного додатку «Автосервіс»;
3) розроблено архітектура реалізації мобільного додатку;
4) реалізовано мобільний додаток із достатньої функціональністю;
5) визначено архітектурні рішення мобільного додатку;

При написанні кваліфікаційної роботи автор проявив достатню самостійність, рішучість та наполегливість уроботі.

Робота успішно пройшла перевірку на унікальність у системі Unicheck,показавши рівень унікальності 100%.

Кваліфікаційна робота підготовлена з урахуванням вимог до робіт відповідного рівня для ЗВО III-IV рівнів вищої освіти, пройшла перевірку наакадемічну доброчесність і може бути рекомендована до захисту.

Науковий керівник  професор, д.ф.-м.н. Володимир Песчаненко

13.12.2022

UNICHECK

Ім'я користувача:
Оксана Блінова

ID перевірки:
1013034322

Дата перевірки:
27.11.2022 10:48:20 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
27.11.2022 13:03:46 EET

ID користувача:
94124

Назва документа: **121_Podmazko_FKNFM_2022_check - Freew1nd**

Кількість сторінок: **47** Кількість слів: **8263** Кількість символів: **61348** Розмір файлу: **94.83 KB** ID файлу: **1012738628**

0%
Схожість

Збіги відсутні

7.91% Цитат

Цитати 8 ..... Сторінка 50

Не знайдено жодних посилань

2.08%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за **8 слів** та **1%**)

1.31% Вилучення з Інтернету 79 ..... Сторінка 51

0.96% Вилученого тексту з Бібліотеки 89 ..... Сторінка 51