ВСТУП

Зі зростанням чисельності ресторанів стрімко посилюється і конкуренція, що неминуче призводить до необхідності ефективно і раціонально використовувати наявні ресурси. У цих умовах для успішного ведення бізнесу необхідно інвестувати в засоби та інструменти його підтримки і розвитку. Один з основних інструментів розвитку ресторанного бізнесу – це сучасна система керування замовленнями у ресторанних мережах.

Сучасна система керування замовленнями у ресторанних мережах – це професійна система управління, яка є багатофункціональною і легко модернізується. Метою автоматизації є підвищення ефективності управління рестораном, прискорення обслуговування та покращення якості надання послуг. Значна частка успіху складається з відмінного сервісу та оперативної роботи персоналу. Саме можливості автоматизації даної сфери дозволяють оптимально поєднувати швидкість і якість.

Одними із переваг даного сервісу перед іншими подібними – це висока якість сервісу і швидкість обслуговування клієнтів, відсутність помилок при оформленні замовлення, обробка та передача замовлення в автоматичному режимі, абсолютний контроль всіх процесів від моменту прийому замовлення до його виконання, можливість безперервно відстежувати фінансові результати роботи закладу.

Ще однією з переваг є оперативність обробки замовлення, оскільки як тільки офіціант прийняв замовлення у відвідувачів і підтвердив його, воно миттєво за допомогою хмарного сховища даних Firebase потрапляє до додатку, який повинен бути встановлений на планшеті, що знаходиться в кухні. Таким чином забезпечується швидка взаємодія між оформленням замовлення та передаванням його на кухню. На кухні повар має змогу передивитись список замовлень, а також список страв для кожного замовлення і позначити статус їх готовності, що дасть змогу повідомити офіціанта у додатку, коли страви будуть готові.

РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

1.1 Аналіз вимог до програмної системи

1.1.1Аналіз предметної області.

Нині ресторанне господарство в Україні досить швидко розвивається, з’являється велика кількість закладів харчування з різними національними кухнями та розвагами. Зважаючи на їхню розмаїтість, замовнику не важко підібрати місце відпочинку до смаку та у відповідності з власним бюджетом. З іншого боку, підприємцям, аби утриматися в цій сфері, необхідно постійно вдосконалювати роботу, забезпечувати високу якість обслуговування, впроваджувати нові технології, підвищувати конкурентоспроможність закладу.

Автоматизація підприємств громадського харчування – це вже давно не питання: всі розуміють її необхідність і окупність. Головним питанням залишається тільки вибір програмного забезпечення для відповідного типу закладу.

І якщо для великих ресторанів і кафе на ринку пропонується ряд великих систем автоматизації (вартість таких рішень починається від 20000 грн. і більше, приблизно 1000 дол. США, доходячи до 50000 - 80000 грн., 2000 - 4000 дол. США), то власники невеликих кафе, бістро, барів і фаст-фудів не завжди можуть дозволити собі такі витрати. А якщо додати до вартості програмних продуктів вартість необхідного обладнання, то загальні витрати на автоматизацію можуть становити 40000 - 150000 грн. (2000 - 5500 дол.США).

Потреба в простій і доступній як за ціною, так і по впровадженню і використанню програмі була давно. Для задоволення цієї потреби програмний продукт «Stuffer» готовий запропонувати власникам малого та середнього бізнесу рішення для підприємств громадського харчування.

Програмний продукт «Stuffer» – ідеальне рішення для автоматизації кафе, барів, бістро, піцерій, фаст-фудів, їдалень і невеликих ресторанів, а також

клубів і розважальних закладів. Особливість програми в тому, що вона дозволяє власнику або керівнику закладу.

Основними складовими ефективності роботи закладу ресторанного господарства є цінова категорія, кухня, якість обслуговування, спектр наданих послуг, атмосфера. Запорукою успіху ефективного розвитку закладу харчування є тісна взаємодія цих складових. Особливістю послуг є збіг у часі та просторі процесів виробництва, реалізації і споживання її споживчої вартості. Виробництво послуг може бути, а може і не бути пов'язане з товаром в його матеріальному вигляді. У нормативних документах по веденню ресторанного бізнесу наведено перелік послуг ресторанного господарства та основні вимоги щодо їх надання.

Послуги, що надаються споживачам на підприємствах ресторанного господарства, визначаються як:

- послуги харчування;
- послуги з виготовлення кулінарної продукції і кондитерських виробів;
- послуги з організації споживання і обслуговування;
- послуги з реалізації кулінарної продукції;
- послуги з організації дозвілля;
- інформаційно-консультативні послуги та ін.

Відносини між споживачами і виконавцем у сфері надання послуг ресторанного господарства затверджені Наказом Міністерства економіки з питань європейської інтеграції України від 24.07.2002 р. № 219 «Правила роботи закладів (підприємств) ресторанного господарства», розроблені відповідно до Законів України «Про захист прав споживачів», «Про безпечність та якість харчових продуктів».

Послуги ресторанного господарства визначаються виконавцем (підприємством ресторанного господарства) відповідно до його типу (а для ресторанів і барів – їх класом), і якщо це передбачено нормативними актами України – підтверджуються органом сертифікації відповідно до державного

стандарту. Підприємства ресторанного господарства, які реалізують алкогольні, тютюнові вироби, зобов’язані мати ліцензію на цей вид діяльності.

Заклади харчування є візитною карткою гостинності будь якого міста, місцем проведення дозвілля і спілкування. Щоб втриматись на ринку праці, підвищувати конкурентоспроможність закладу, власники бізнесу намагаються не лише здійснювати обслуговування згідно стандартів, але й по-різному заохочують клієнтів: практикують дисконтні картки, подарунки від закладу до дня народження відвідувача, безкоштовне надання страви-сюрпризу, за умови замовлення ресторанних послуг на певну суму, дегустації, приготування страв шеф-поваром за рецептом клієнта, радіо-вікторини, де переможці правильних відповідей, нагороджуються безкоштовним ланчем.

У наш час заклади харчування є одним з основних способів проведення дозвілля і спілкування, тому керівники закладів значну увагу приділяють атмосфері, комфортності клієнта. Кожен заклад харчування відрізняється від інших власним стилем: національний, епохи лицарських турнірів і замків, сільський, мисливський, елітний, царський тощо. Серед атрибутів інтер'єру можна побачити картини, фонтани, каміни, акваріуми, композиції з декоративних квітів тощо. Але основне, що складає престиж закладу харчування, - це смачні страви та широкий асортимент, кулінарна майстерність, професійний рівень обслуговування клієнтів, інфраструктура сервісу, гостинний прийом.

Вдосконалення рівня обслуговування потребує високих фінансових витрат власників бізнесу. Зокрема, керівники часто проводять семінари, курси для офіціантів, вкладаючи в це великі обсяги коштів, але при цьому не маючи гарантії якості роботи майбутнього працівника. А якість їх роботи часто розчаровує. Виконуючи замовлення, офіціант повинен дотримуватись загальних правил техніки обслуговування і строго дотримуватись прийнятої послідовності подачі холодних і гарячих закусок, різноманітних страв і напоїв.

Недосвідченість персоналу в закладах харчування є однією з основних

недоліків сучасного обслуговування. Недостатня кількість персоналу з обслуговування призводить до виникнення черг. Збільшення тривалості обслуговування впливає на індивідуальний розклад часу кожного клієнта закладу харчування. Впровадження інформаційних технологій в області обслуговування клієнтів у закладах харчування – важливий крок у розвитку галузі, підвищення рівня обслуговування клієнтів, спосіб вирішення основних проблем в області обслуговування клієнтів.

Інформаційні технології сьогодні можна класифікувати за такими ознаками: способом реалізації в інформаційній системі, ступенем охоплення завдань управління, класами реалізованих технологічних операцій, типом користувацького інтерфейсу, варіантами використання мережі, що обслуговує предметну область.

Управління – найважливіша функція, без якої неможлива цілеспрямована діяльність будь-якої соціально-економічної, організаційно-виробничої системи (підприємства, організації, території). Систему, що реалізує функції управління, називають системою управління. Найважливішими функціями, реалізованими цією системою, є прогнозування, планування, облік, аналіз, контроль і регулювання.

Управління пов'язане з обміном інформацією між компонентами системи, а також системи з довкіллям.

У процесі управління одержують відомості про стан системи в кожен момент часу, про досягнення (або недосягнення) заданої мети з тим, щоб впливати на систему і забезпечити виконання управлінських рішень.

Автоматизована інформаційна система – це сукупність інформації, економіко-математичних методів і моделей, технічних, програмних, технологічних засобів і фахівців, призначена для опрацювання інформації та прийняття управлінських рішень.

Отже, інформаційну систему можна визначити з технічного погляду як набір взаємопов'язаних компонентів, які збирають, опрацьовують, накопичують і розподіляють інформацію, щоб підтримати прийняття рішень і управління в

організації. На додаток до підтримки прийняття рішень, координації й управління інформаційні системи можуть також допомагати менеджерам аналізувати проблеми, роблять видимими комплексні об'єкти й створюють нові вироби.

Інформаційні системи можуть також бути функціонально диференційовані. Головні організаційні функції типу продажу і маркетингу, виробництва, фінансів, бухгалтерського обліку та людських ресурсів обслуговуються власними інформаційними системами. У великих організаціях підфункції кожної з цих головних функцій також мають власні інформаційні системи. Наприклад, функція виробництва могла б мати системи для управління запасами, управління процесом, обслуговування організації, автоматизованого розроблення і матеріального планування вимог.

1.1.2Аналітичний огляд існуючих рішень.

Poster – система автоматизації роботи кафе чи ресторану, яка включає у себе безліч можливостей. За допомогою даного сервісу можливо покрити усі необхідні варіанти та принципи надання хороших ресторанних послуг високого класу.

Програмний комплекс Poster містить такі основні переваги, які допомагають даному продукту :

- доступ з будь-якої точки світу;
- робота без інтернету;
- просте встановлення та навчання;
- доступна ціна;
- зручне робоче місце офіціанта.

Доступ з будь-якої точки світу. Poster – хмарна система. Це означає, що статистика, склад і фінанси вашого бізнесу доступні вам з будь-якого місця, де є інтернет.

Робота без інтернету. Навіть якщо в закладі тимчасово немає інтернету, то робота не зупиниться: є можливість приймати замовлення та

друкувати чеки. Дані автоматично потраплять до хмари, коли з'єднання відновиться.

На рисунку 1.1 зображено вигляд сайту програмного продукту Poster.

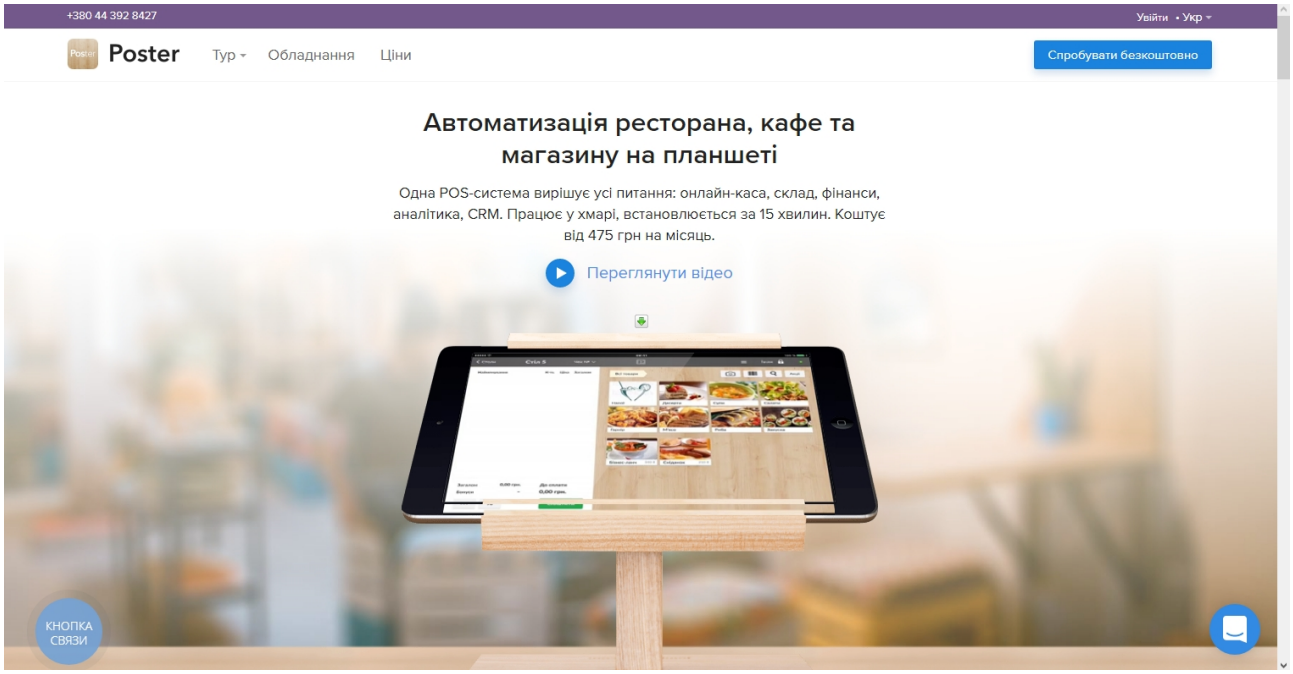


Рисунок 1.1 – вигляд сайту програмного продукту Poster

Просте програмне встановлення та навчання. Розробники Poster є фанатами простих й зручних продуктів. Тому даний програмний комплекс не вимагає додаткового навчання – все й так зрозуміло. Ваші працівники почнуть продавати вже через 5 хвилин.

Доступна ціна. Poster працює за моделлю передплати, вам не потрібно заморожувати велику суму в автоматизації. Вартість від 475 грн на місяць.

Зручне робоче місце офіціанта. Ваші офіціанти, бармени або бариста повинні спілкуватися з гостями, а не з програмним забезпеченням. Саме тому дана система зроблена максимально простою, швидкою та зрозумілою.

Даний програмний комплекс містить такі складові для роботи:

- термінал офіціанта;
- грошові скриньки;
- фіскальні реєстратори;

– сканери штрих-кодів;
– принтери чеків.

На рисунку 1.2 та 1.3 зображено варіанти обладнання із фіскалізацією, оскільки даний сервіс може запропонувати як обладнання із фіскалізацією, так і без. Підтримуються всі чекові принтери, які працюють за протоколом ESC/POS та підключаються через Ethernet-інтерфейс.

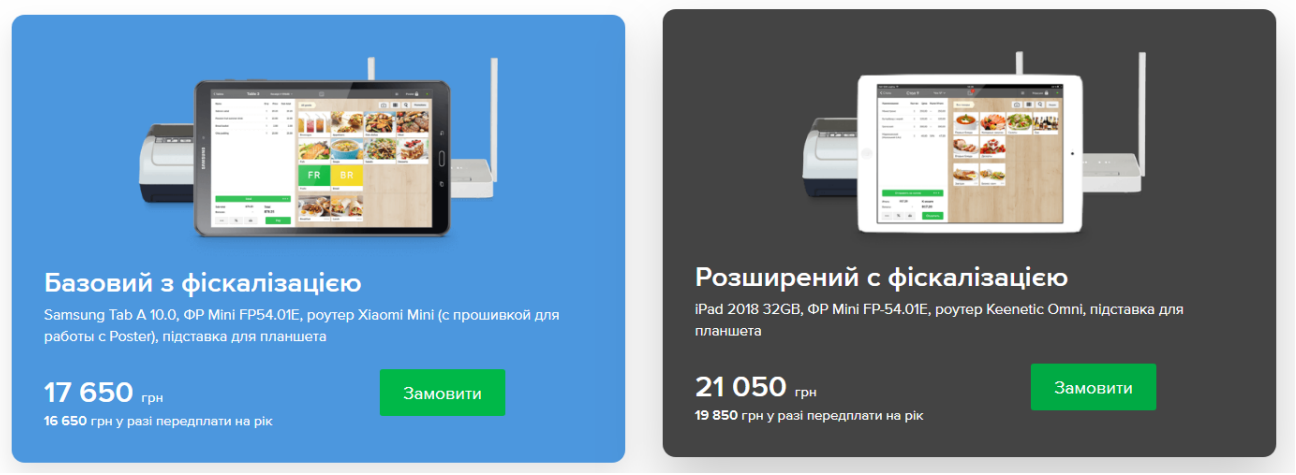


Рисунок 1.2 – варіанти обладнання із фіскалізацією

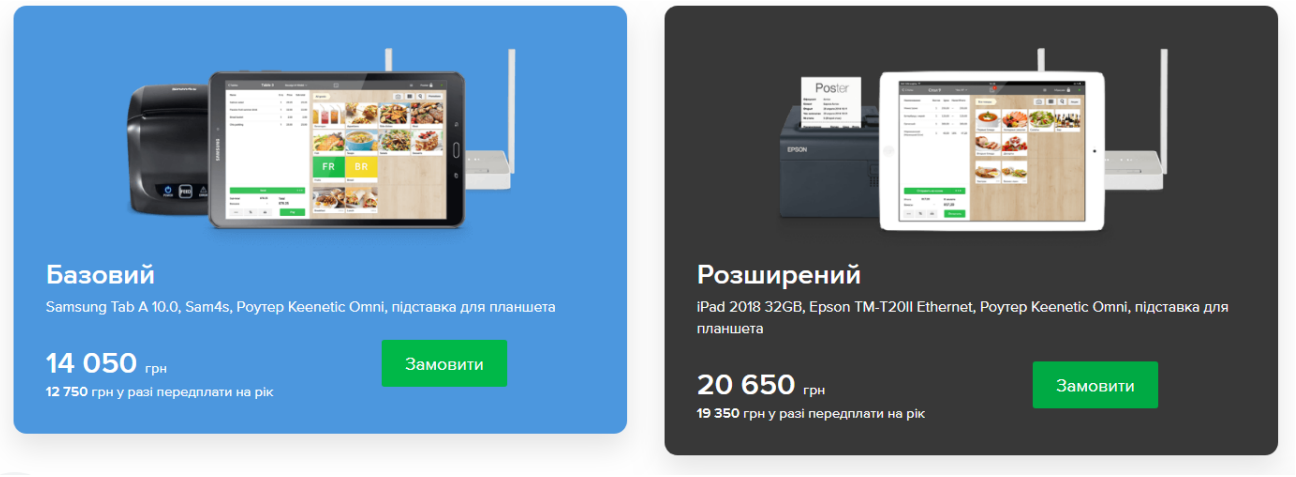


Рисунок 1.3 – варіанти обладнання без фіскалізації

Термінал офіціанта. Робоче місце офіціанта запускається на будь-якій платформі: iOS, Android, Windows, macOS.

Грошові скриньки. Підключіть грошову скриньку до принтеру та він буде відкриватися кожного разу, коли чек буде оплачено готівкою.

Фіскальні реєстратори. Poster повністю сумісний з ФЗ-54. Ми працюємо з Атолами, Штрихами та Вікі-принтами.

Сканери штрих-кодів. Так, з ними ми теж працюємо. Bluetooth для iOS та Android, USB для Windows.

Принтери чеків. Poster підтримує близько 20 моделей термальних принтерів. Epson, Star, XPrinter, Sam4s, Mprint, Mini та інші.

Yelp – одна з найбільших і найяскравіших програмних комплексів у сфері ресторанного бізнесу. Незважаючи на те, що у нього багато галасу та шуму, його зростання, в основному, пояснюється тим, що це настільки чудовий додаток. На рисунку 1.4 зображено інтерфейс додатку Yelp.

Пропонуючи понад 50 мільйонів відгуків для підприємств у всьому світі, можна легко шукати ресторани поблизу, читати безліч оглядів і навіть переглядати чудові місцеві пропозиції та фотографії цього місця. Фільтри пошуку гарантують, що ви можете звузити речі відповідно до відстані, ціни та рейтингу, щоб точно знати, до чого потрапите.

Рисунок 1.4 – інтерфейс додатку Yelp

Yelp був заснований у 2004 році колишніми співробітниками PayPal Расселом Сіммонсом та Джеремі Стоппелманом. Компанія Yelp зросла у використанні та отримала кілька раундів фінансування у наступні роки. До 2010 року він мав 30 мільйонів доларів доходу, а веб-сайт опублікував близько 4,5 мільйонів оглядів натовпу. З 2009 по 2012 рік Yelp розширився по всій Європі та Азії. У 2009 році він розпочав кілька переговорів з Google щодо потенційного придбання. Yelp стала публічною компанією в березні 2012 року і вперше стала прибутковою через два роки.

Також, Yelp – це служба ділових довідників та оглядовий форум з натовпом, а також однойменна публічна компанія, штаб-квартира якої знаходиться у Сан-Франциско, Каліфорнія. Компанія розробляє, розміщує та продає веб-сайт Yelp.com та мобільний додаток Yelp, в яких публікуються відгуки про бізнес, ресторани, тощо. Він також працює як онлайн-сервіс бронювання під назвою Yelp Reservations.

Станом на другий квартал 2019 року Yelp повідомляє, що середній місяць складає 61,8 мільйона унікальних відвідувачів через настільний комп'ютер та 76,7 мільйона унікальних відвідувачів через свій мобільний веб-сайт. Станом на 30 червня 2019 року Yelp на своїй сторінці зв’язків з інвесторами заявив, що на своєму сайті було 192 мільйони відгуків.

Компанію звинувачують у використанні недобросовісної практики для отримання доходів від підприємств, які переглядаються на її веб-сайті - наприклад, шляхом подання більше негативної інформації про огляди для компаній, які не купують її рекламні послуги, або шляхом чіткого розміщення реклами конкурентів таких не -оплачують компанії. Також були скарги на агресивну та оманливу тактику з боку деяких її торгових представників. На надійність системи оглядів компанії вплинуло також подання підроблених оглядів зовнішніх користувачів, таких як помилкові позитивні відгуки, подані компанією для просування власного бізнесу, або помилкові негативні відгуки щодо конкуруючих підприємств - практика, іноді відома як "астротурфінг" , з якою компанія намагалася боротися різними способами.

На рисунку 1.5 зображено екран пошуку ресторанів.

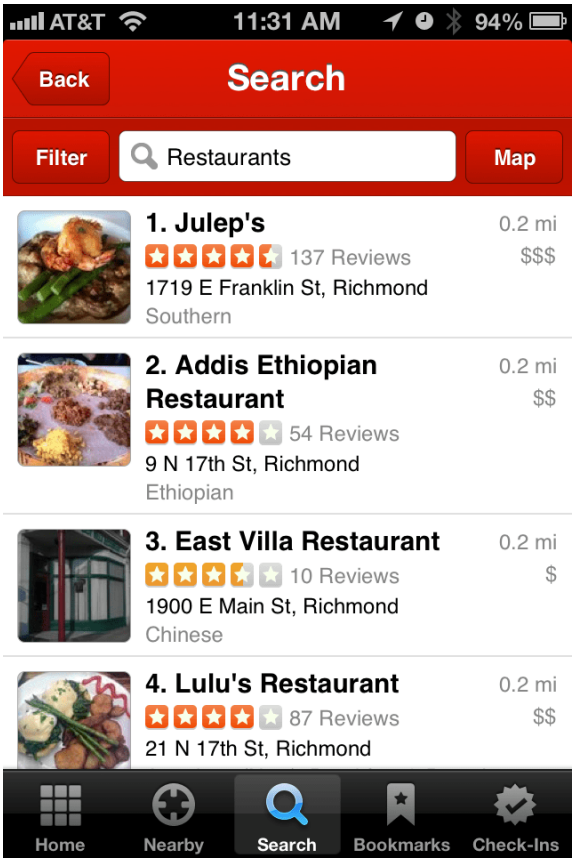


Рисунок 1.5 – екран пошуку ресторанів

Ще одним із відомих додатків для ресторанного бізнесу є додаток Urbanspoon. Даний додаток дозволяє порівнювати вибір ресторану за рейтингом, відстанню, кухнями та рівнями популярності. Urbanspoon – це чудова допомога, особливо коли ви перебуваєте в новому місті. Ви можете переглядати меню та фотографії пропонованої їжі, а також дивитися путівники критиків в Інтернеті як критиків, так і інших любителів їжі. Навіть можна зарезервувати за допомогою програми, переконавшись, що це їдальський швейцарський ніж.

Urbanspoon – служба інформації та рекомендацій у ресторані, заснована у 2006 році колишніми співробітниками Джобстера, які пропонували свої послуги в Північній Америці та частинах англомовної Австралії та Європи. У січні 2015 року Zomato, служба пошуку та виявлення ресторанів, що базується в Індії, придбала Urbanspoon.

На рисунку 1.6 зображено інтерфейс додатку Urbanspoon.

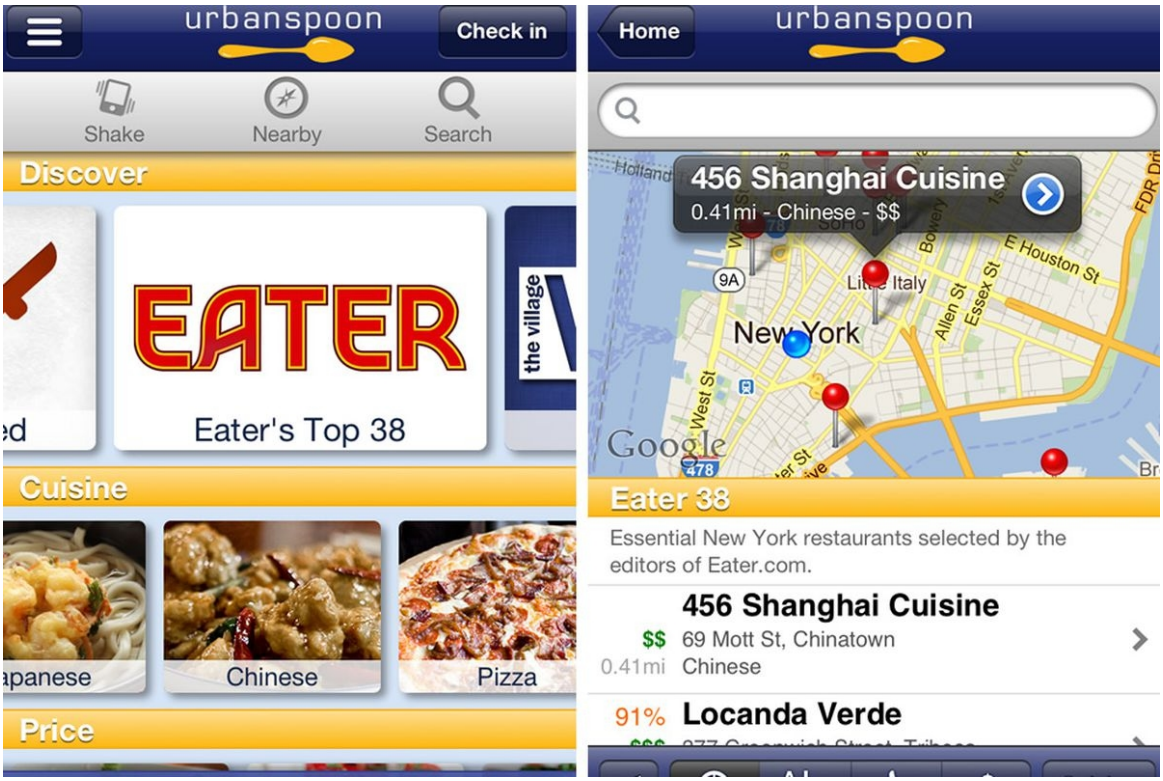


Рисунок 1.6 – інтерфейс додатку Urbanspoon

Веб-сайт Urbanspoon та бізнес-операція в Сіетлі спочатку розвивались із додаванням колективів вмісту для збору даних від ресторанів від дверей до дверей, оперативного дизайну, який добре служив Zomato в Індії. Однак Zomato припинив роботу веб-сайту Urbanspoon 1 червня 2015 року, перенаправляючи трафік Urbanspoon на сервери Zomato; водночас логотип Urbanspoon був включений у новий логотип Zomato.

Zomato припинив раніше розробку додатків Urbanspoon, що призвело до від'їзду всіх інженерів в Сіетлі до серпня 2015 року. В середині жовтня 2015 року було закрито кілька офісів США Zomato (раніше Urbanspoon), включаючи колишню корпоративну штаб-квартиру в Сіетлі. Бізнес-звіти станом на жовтень 2015 року вказують або на те, що Zomato-Urbanspoon припинив усі операції в США, або що він збереже пару сайтів, включаючи Даллас, новий сайт адміністративної функції Zomato в США.

Також, слід згадати саме сервіс Zomato. Zomato – це індійський агрегатор ресторанів, і доставки продуктів харчування, заснований у 2008 році. Його

розпочали Deepinder Goyal та Pankaj Chaddah. Він надає інформацію, меню та відгуки користувачів про ресторани, а також має варіанти доставки їжі з ресторанів-партнерів у вибрані міста. Станом на 2016 рік послуга доступна у 24 країнах. На рисунку 1.7 зображено постер сервісу Zomato.

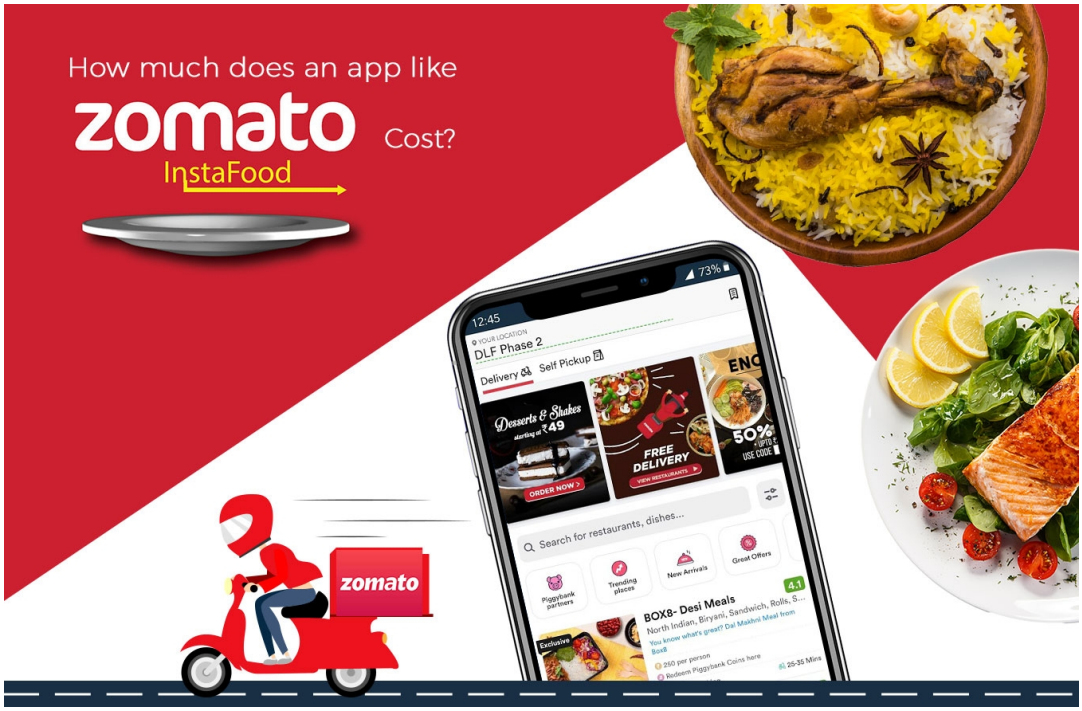


Рисунок 1.7 – постер сервісу Zomato

Завдяки своїй хмарній кухні компанія мала на меті допомогти ресторанам розширити свої послуги, не несучи жодних постійних витрат. Пізніше, у вересні 2017 року, Zomato стверджував, що компанія "стала вигідною" у 24 країнах, що діють, і оголосила, що "ресторанів з нульовою комісією" буде запроваджено для ресторанів-партнерів. Наприкінці 2017 року Zomato перестав приймати оновлення від своїх активних користувачів, не використовуючи модераторів для перевірки та оновлення. Інформація про ресторан не оновлювалася. Користувачі програми повідомили про проблеми з новими можливостями для оплати замовлень.

Наступний досить відомий сервіс для надання ресторанних послуг є такий сервіс, як Zagat. Сервіс Zagat було створено Тімом та Ніною Загатом у 1979 році як спосіб збирати та співвідносити рейтинги ресторанів під час обідів, інтерфейс сервісу зображено на рисунку 1.8.

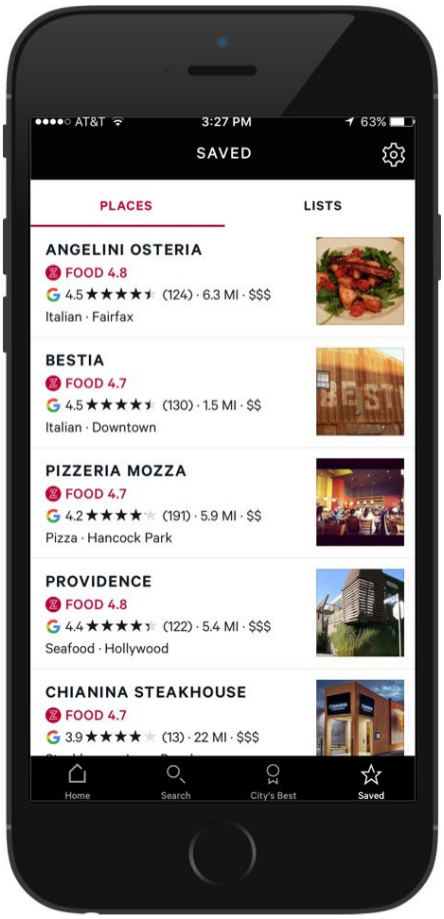


Рисунок 1.8 – інтерфейс додатку Zagat

Загати опитали своїх спершу друзів. Пізніше, у 2005 році в опитування Zagat було включено 70 міст, огляди на основі даних 250 000 людей з гідів, що повідомляють про рейтинги ресторанів, готелів, нічного життя, шопінгу, зоопарків, музики, кінотеатрів, театрів, полів для гольфу та авіакомпаній. Посібники продаються в книжковій формі, і раніше вони були доступні лише у вигляді платної підписки на веб-сайті Zagat.

У рамках придбання Google у розмірі понад 150 мільйонів доларів у вересні 2011 року, пропозиція відгуків та рейтингів Zagat стала частиною групи «Гео та комерція» Google, з часом була тісно інтегрована в сервіси Google. 29 липня 2013 року Google відновив веб-сайт Zagat із покращеним інтерфейсом, але скоротив його з 30 міст до дев'яти. Вони випустили базу даних оглядів з інших 21 міст у наступні дні, коли вони працювали над розширенням для включення нових міст на новий сайт. У грудні 2012 року Google оголосив, що звільнить

більшість колишніх штатних працівників Zagat, які були підрядниками на момент придбання, що призвело до пророчих ділових звітів, що описують майбутнє виробництва книг Zagat як похмурі, а також наступні ділові новини звіти, що фіксують скорочення своїх поліграфічних підприємств. Незважаючи на те, що Google придбання та інтеграція Zagat забезпечили його сильним брендом у рекомендаціях місцевого ресторану та достатньому вмісті для пошуку на основі локації.

Ще однією досить відомою платформою для ресторанного бізнесу є сервіс LocalEats. LocalEats – інформаційно-рекомендаційна служба ресторану, що підтримується підприємством, яке працює в США та 50 міжнародних містах.

Даний сервіс розробив популярну програму iPhone і має веб-сайт, подібний до власних Google Zagat, Yelp та IAC / InterActiveCorp, що належить Urbanspoon. На початку 2011 року LocalEats запустив додаток iPad, який додав 50 міжнародних міст до свого посібника зі США. Пізніше в 2011 році LocalEats співпрацював з агрегатором угод для ресторанів BiteHunter, щоб принести інформацію про угоди на свій веб-сайт та додаток для угод у ресторанах LocalEats. Фірма також співпрацює з підтримкою VC SinglePlatform, щоб запропонувати меню додатків для більш ніж 12 000 з 14 000 обраних ресторанів. На рисунку 1.9 зображено інтерфейс додатку LocalEats.

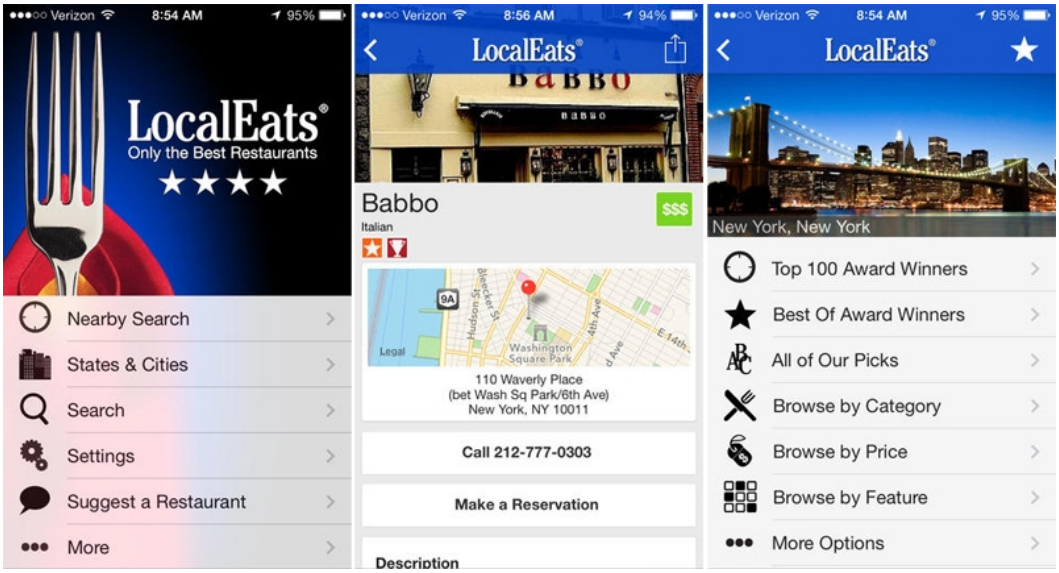


Рисунок 1.9 – інтерфейс додатку LocalEats

Інтерфейс для LocalEats простий: він пропонує вибрати, чи шукати у своєму місті (якщо він є у списку міст у додатку), поблизу (в межах декількох миль від того, де вас знайде GPS), або з числа ваших улюблених. Якщо вибрано місто, то будете використовуватись інтерфейс з вкладками, щоб шукати серед топ-100 у цьому місті, переглядати "Найкращі" в певних категоріях або шукати ресторани за сусідніми або типовими кухнями.

Після вибору ресторану інтерфейс з вкладками надає розташування, категорію, діапазон цін, посилання на веб-сайт, детальний опис та коментарі. Ви можете переглянути фотографії закладу або скористатися посиланнями в інтерфейсі, щоб зателефонувати до ресторану перед поїздкою. Ви навіть можете використовувати LocalEats, щоб забронювати таксі, щоб доставити вас додому, коли їжа закінчена. LocalEats ідеально підходить, якщо ви захоплені підтримкою місцевого бізнесу. У США додаток безкоштовний для Android, 99 центів для iPhone та 2,99 доларів для користувачів BlackBerry.

1.1.3Постановка задачі.

Для вирішення, поставленого завдання необхідно провести формалізацію задачі, як необхідний етап розробки завдання, який полягає в побудові структури довідників, схеми взаємозв'язків документів з довідниками і опису алгоритмів обробки інформаційних потоків.

Для підприємств, що працюють в сфері громадського харчування дуже важливо оперативно і своєчасно вести облік продуктів і замовлень. Тому задача автоматизації обробки даних ресторану є однією з пріоритетних завдань розвитку підприємства. Основною метою завдання є своєчасний оперативний облік замовлень клієнтів, а також вирішуються завдання складання звітів та інших необхідних документів.

Автоматизація дозволить знизити трудовитрати і число помилок працівників підприємства при обробці даних. Також вона призведе до економії часу співробітників і підвищить оперативність прийнятих рішень. Централізоване зберігання даних підвищує точність і достовірність наданої

інформації. Вся інформація необхідна для вирішення даного завдання зберігається в хмарному сховищі даних Firebse Firestore. Автоматизація дозволить отримувати звіти по заданим умовам, з підрахунком підсумкових значень, і організувати зберігання даних для подальшого аналізу.

Успішний ресторан – це злагоджений механізм, який пропонує своїм споживачам високий рівень сервісу і оперативну роботу персоналу.

Дослідженню проблем ресторанного бізнесу присвячені праці таких науковців, як Архіпова В.В., Мостової Л.М., Новікової О.В., Мальської М.П., Пандяка І.Г. та ін.шими, що в умовах конкуренції підприємства ресторанного бізнесу вимагають сучасних методів управління, які базуються, в першу чергу, на впровадженні та використанні автоматизованих систем управління.

В даний час на ринку комп'ютерних систем є універсальні аналітичні програми і спеціальні, що використовуються в окремих галузях економіки. Більшість користувачів віддають перевагу універсальним комп'ютерним засобам унаслідок їх достатньо легкої адаптації до особливостей управлінських функцій в різних закладах ресторанного бізнесу.

Сучасні підприємства вимагають найновіших методів управління. Ухвалення управлінських рішень в умовах постійно змінного, динамічного середовища вимагає не тільки аналізу, оцінки і прогнозування внутрішнього розвитку підприємства, але і забезпечення відповідності між зовнішнім мікросередовищем, зовнішнім макросередовищем і результатами фінансово-господарської діяльності підприємства, що, у свою чергу, висуває високі вимоги до інформаційного та інших видів забезпечення управління підприємством. Інтеграція цифрової обробки інформації прискорює в рази робочий процес, що створює приємну атмосферу турботи про відвідувачів закладу.

Отже, слід розглянути обов’язки офіціанта закладу громадського харчування, таких, як кафе, бари, ресторани, тощо, тому що даний програмний продукт розробляється з метою оптимізації та покращення роботи офіціанта, допомагаючи йому працювати більш продуктивно, використовуючи додаток на смартфоні з операційною системою iOS.

Отже, офіціант – це працівник ресторану, кафе, бару, їдальні, закусочної, підприємства харчування в туристичних комплексах, готелях і на транспорті, що подає страви відвідувачам.

Офіціант повинен знати правила обслуговування відвідувачів і сервірування столу, асортимент і кулінарну характеристику страв і напоїв, ціни на них, правила роботи на контрольно-касових апаратах. У випадку роботи з іноземними відвідувачами необхідне знання іноземних мов у межах професійного розмовного мінімуму.

Для успішного виконання роботи офіціант повинний бути товариським, бездоганно ввічливим, легко вступати в контакти і розуміти особливості поведінки людей. Необхідна стійка увага, розвинута оперативна і довгострокова пам'ять, розвинуті розрахункові здібності, гарний розвиток органів почуттів (зору, нюху, тактильної чутливості). Потрібна фізична витривалість, швидкодія, ручна вправність з одночасним артистизмом.

В обов'язки офіціанта входить:

- сервірування столу;
- зустріч відвідувачів;
- ознайомлення відвідувачів із меню;
- у разі необхідності – поради відвідувачам у виборі страв;
- одержання замовлення;
- передача замовлень у гарячий і холодний цехи кухні;
- пробивання чеків на касовому апараті;
- доставка замовлення відвідувачу;
- розрахунок відвідувача;
- прибирання використаного посуду.

У задачі автоматизується процес замовлення страв в ресторані, що необхідно для швидкого пошуку, обробки інформації про страви і продуктах, що використовуються в них. Автоматизувавши процес, ресторан зможе виявити саму куповану і популярну продукцію, що дозволить, згодом, стежити за зміною попиту і відповідно складати більш вигідний план закупівель сировини.

Дотримання смакам споживачів підвищить популярність організації серед споживачів і збільшить виручку.

До функцій, що реалізовуються в даній задачі, відносяться:

– вибірка страв по найменуванню та ціні;

– реєстрація нових рецептів для використання в кулінарії та внесення їх в меню(у нотатках);

– видача звітів по заданих параметрах;

– сортування страв за типом (закуска, гаряче, салати);

– додаток для кухні із можливістю перегляду замовлення та страв, які потрібно приготувати;

– можливість роздрукувати на чек замовлення, яке потрібно приготувати.

Звітність, що реалізується при автоматизації процесу:

– формування меню ресторану(додавання даних у консолі Firebase);

– складання бланка замовлених страв, переданого на кухню.

З усього вищевказаного випливає кілька обмежень предметної області:

– кожен запис повинен бути унікальною і не мати своїх

– дублікатів;

– обмеженість продуктів на складі;

– ціна продукції вимірюється в національній валюті.

Також, слід зауважити, що даний програмний продукт повинен бути максимально простим із достатньою кількістю функціоналу для того, що офіціант міг швидко прийняти замовлення, а у разі потреби його редагувати чи видалити. Якщо замовлення оформлене і звірене із відвідувачами, тоді воно може бути відправлене на кухню для подальшого його етапу виконання.

1.1.4 Специфікація, вимоги до сервісу та забезпечення.

Основною вимогою до системи є відмовостійкість, оскільки це один із найважливіших показників будь-якого сервісу. Отже, навантаження на сервер

постійно буде зростати, тому потрібно забезпечити коректну обробку інформації, а також побудувати правильну архітектуру взаємодії компонентів.

База даних на сервері повинна забезпечувати швидку вибірку елементів та їх обробку. Сервер повинен бути з високою відмовостікістю, щоб уникнути непрацездатності та збоїв.

Додаток для кухні та додаток для офіціанта працюють через сервер, тобто всі дані зберігаються та обробляються на сервері.

Оскільки увесь сервіс взаємодіятиме за допомогою сервера, слід поставити вимоги до сервера. Отже, сервер повинен забезпечувати:

- реєстрацію офіціанта у базу даних на сервері;
- авторизацію офіціанта у додатку (запит на сервер і перевірка на наявність такого у базі даних);
- зберігання у базі даних та надсилання офіціанту інформації про статус столиків;
- зберігання у базі даних інформації про категорії страв;
- зберігання інформації про страви (опис, ціни, можливі модифікації страви, можливі додатки до страви);
- зберігання у базі нотаток, які записують офіціанти, стосовно меню, щоб можна було підвести статистику, проаналізувати і покращити ті чи інші страви;
- зберігати зроблене відвідувачами замовлення із списком страв, гостей і т.п.;
- зберегти отримане замовлення, та відправити його на додаток для кухні, щоб повари могли подивитись та прийняти у роботу замовлення і приступити до приготування.

Також, нефункціональні вимоги потрібно поставити у відповідності до потреб офіціантів, та персоналу кухні.

Нефункціональні вимоги до сервісу:

- інтерфейс повинен бути інтуїтивно зрозумілим для офіціанта;

- програмний продукт повинен постачатись із презентацією для кращого розуміння системи загалом, а також для швидкої інтеграції його у громадський заклад харчування;

- додаток для офіціанта повинен містити дві мови: англійську(основну) та українську;

- сервер повинен бути відмовостійким та мати змогу працювати під високим навантаженням;

- додаток для офіціанта повинен містити сторінку з іменами його розробників та авторськими правами;

- додаток повинен постачатись лише для комерційного використання;

- система повинна бути призначена для iOS.

Специфікація iOS додатків для офіціанта та кухні:

- мова розробки додатку: Swift;

- веб-сервіси: REST;

- обов’язкова сумісність із всіма версіями iOS починаючи від iOS 9;

- забезпечення потокобезпеки та асинхронності роботи додатку: RxSwift, ReactiveCocoa;

- робота із зображеннями: Photokit;

- розмітка сторінки: SwiftUI;

- Binding представлень: RxViewBinding;

- робота з сервером, парсинг даних: Standart iOS development kit;

- інтеграційне тестування: XCTest, OCMock, Appium;

- тестування UI: EarlGray, monkeyTest;

- середовще розробки: XCode;

- система контролю версій: GIT;

- репозиторій для завантаження проекту: GitHub;

- програми для роботи з GIT: SourceTree, Git Kraken, консоль Xcode або її графічний інтерфейс.

Специфікація серверної частини:

- мова розробки серверної частини: JavaScript(консоль Firebase);

- вигляд програми: консоль Firebase;
- веб-сервіси: REST;
- серіалізація: FirebaseSerializer;
- фреймворк: FirebaseFirestore;
- логування: Timber;
- робота з мережевими протоколами: OkHttp;
- робота з http ресурсами: HttpClient;
- тестування: JUnit;
- база даних: Firestore realtime database;
- середовище розробки серверної частини: консоль Firebase або будь-який редактор для JavaScript коду.

Визначившись із стеком необхідних технології, можна звернути увагу на розробку інтерфейсу для взаємодії з додатками.

Найбільша увага повинна приділятися вимогам до зовнішніх інтерфейсів. Інтерфейс користувача – засіб зручної взаємодії користувача з інформаційною системою. Сукупність засобів для обробки та відображення інформації, максимально пристосованих для зручності користувача; у графічних системах інтерфейс користувача реалізовується багатовіконним режимом, змінами кольору, розміру, видимості (прозорість, напівпрозорість, невидимість) вікон, їхнім розташуванням, сортуванням елементів вікон, гнучкими налаштовуваннями як самих вікон, так і окремих їхніх елементів таких як файли, папки, ярлики, шрифти, доступністю багатокористувацьких налаштувань.

Важливим атрибутом є доступність – властивість інформаційного ресурсу, яка полягає в тому, що користувач та/або процес, який володіє відповідними повноваженнями, може використовувати цей ресурс відповідно до правил, встановлених політикою безпеки не очікуючи довше заданого (прийнятного) інтервалу часу. Суть властивості полягає в тому, що потрібний інформаційний ресурс знаходиться у вигляді, необхідному користувачеві, в місці, необхідному користувачеві, і в той час, коли він йому необхідний, з

цілодобовим доступом, а також із цілодобовою підтримкою.

Інформаційна безпека є важливим атрибутом майбутньої створюваної системи. Це захищеності систем обробки і зберігання даних, при якому забезпечено конфіденційність, доступність і цілісність інформації, або комплекс заходів, спрямованих на забезпечення захищеності інформації від несанкціонованого доступу, використання, оприлюднення, руйнування, внесення змін, ознайомлення, перевірки, запису чи знищення (у цьому значенні частіше використовують термін «захист інформації»).

Найбільша увага повинна приділятися вимогам до зовнішніх інтерфейсів. Інтерфейс користувача – засіб зручної взаємодії користувача з інформаційною системою. Сукупність засобів для обробки та відображення інформації, максимально пристосованих для зручності користувача; у графічних системах інтерфейс користувача реалізовується багатовіконним режимом, змінами кольору, розміру, видимості (прозорість, напівпрозорість, невидимість) вікон, їхнім розташуванням, сортуванням елементів вікон, гнучкими налаштовуваннями як самих вікон, так і окремих їхніх елементів таких як файли, папки, ярлики, шрифти, доступністю багатокористувацьких налаштувань.

Оскільки тема дипломної включає в себе захист даних шляхом шифрування їх за допомогою стандарту AES256, який дозволяє забезпечити конфіденційність даних.

Отже, AES (Advanced Encryption Standard), також відомий під назвою Rijndael — симетричний алгоритм блочного шифрування (розмір блока 128 біт, ключ 128/192/256 біт), фіналіст конкурсу AES і прийнятий як американський стандарт шифрування урядом США. Вибір припав на AES з розрахуванням на широке використання і активний аналіз алгоритму, як це було із його попередником, DES. Державний інститут стандартів і технологій (англ. National Institute of Standards and Technology, NIST) США опублікував попередню специфікацію AES 26 жовтня 2001 року, після п'ятилітньої підготовки. 26

травня 2002 року AES оголошено стандартом шифрування. Станом на 2009 рік AES є одним із найпоширеніших алгоритмів симетричного шифрування.

В принципі, алгоритм, запропонований Рейменом і Дейцменом, і AES не одне і те ж. Алгоритм Рейндол підтримує широкий діапазон розміру блоку та ключа. AES має фіксовану довжину у 128 біт, а розмір ключа може приймати значення 128, 192 або 256 біт. В той час як Рейндол підтримує розмірність блоку та ключа із кроком 32 біт у діапазоні від 128 до 256. Через фіксований розмір блоку AES оперує із масивом 4×4 байт, що називається станом (версії алгоритму із більшим розміром блоку мають додаткові колонки).

Інформаційна безпека є важливим атрибутом майбутньої створюваної системи. Це захищеності систем обробки і зберігання даних, при якому забезпечено конфіденційність, доступність і цілісність інформації, або комплекс заходів, спрямованих на забезпечення захищеності інформації від несанкціонованого доступу, використання, оприлюднення, руйнування, внесення змін, ознайомлення, перевірки, запису чи знищення (у цьому значенні частіше використовують термін «захист інформації»).

Вимоги до пристрою користувача:

- операційна система: iOS 9 та вище;
- підключення до мережі інтернет;
- надання дозволів додатку для зберігання даних на пристрої.

Вимоги до серверу:

- хостинг повинен мати хорошу швидкість передачі даних;
- хостинг повинен мати захист від DDOS атак;
- хостинг повинен забезпечувати стабільну та безперебійну роботу даного сервісу;
- хостинг повинен мати хорошу пропускну здатність.

Важливим атрибутом є доступність – властивість інформаційного ресурсу, яка полягає в тому, що користувач та/або процес, який володіє відповідними повноваженнями, може використовувати цей ресурс відповідно

до правил, встановлених політикою безпеки не очікуючи довше заданого (прийнятного) інтервалу часу. Суть властивості полягає в тому, що потрібний інформаційний ресурс знаходиться у вигляді, необхідному користувачеві, в місці, необхідному користувачеві, і в той час, коли він йому необхідний, з цілодобовим доступом, а також із цілодобовою підтримкою.

1.1.5 Етапи життєвого циклу сервісу.

Життєвий цикл проекту – це період часу від задуму проекту до його закінчення, який може характеризуватися моментом здійснення перших витрат за проектом (поява проекту) й отриманням останньої вигоди (ліквідація проекту).

Життєвий цикл проекту є концепцією, що розглядає проект як послідовність фаз, подій та етапів, кожна з яких має свою назву та часові межі.

Життєвий цикл проекту є концепцією, що розглядає проект як послідовність фаз, подій та етапів, кожна з яких має свою назву та часові межі. Він є базовим елементом концепції проектного аналізу та відображає розвиток проекту, а саме роботи, які відбуваються на різних стадіях підготовки, реалізації та експлуатації проекту. Таким чином відображається схема або алгоритм певної послідовності дій під час розробки та впровадження проекту.

Ступінь деталізації і термінологія опису відповідних процедур залежать від характеру проекту, предметної культури, поставлених завдань, наявних ресурсів і, можливо, уподобань та смаків проектного аналітика.

Як правило проектний цикл поділяють на три фази: передінвестиційну, інвестиційну та експлуатаційну, які, у свою чергу, розгалужуються на стадії.

Передінвестиційна фаза включає такі стадії:

- преідентифікація;
- ідентифікація;
- підготовка;
- розробка та експертиза;
- детальне проектування.

Інвестиційна фаза охоплює роботи, які можна об'єднати у такі стадії:

- підготовка і проведення тендерів;
- інженерно-технічне проектування;
- будівництво;
- виробничий маркетинг;
- пропозиції щодо підвищення продуктивності;
- прогнози на роботу сервісу;
- навчання персоналу.

Основними стадіями експлуатаційної фази є:

- здавання в експлуатацію (е граничною між інвестиційною та експлуатаційною фазами, тому може перебувати і в одній, і в іншій);
- виробнича експлуатація;
- заміна та оновлення;
- розширення та інновації;
- вдосконалення уже існуючого сервісу;
- внесення змін в уже існуючий функціонал;
- заключна оцінка проекту.

По осі часу життєвий цикл ділиться на чотири основні фази:

- початок (Inception) – формування концепції проекту, розуміння того, що створюється, уявлення про продукт (vision), розробка бізнес-плану (business case), підготовка прототипу програми або часткового вирішення. Це фаза збору інформації та аналізу вимог, визначення способу проекту в цілому. Мета – отримати підтримку і фінансування. У кінцевій ітерації результат цього етапу - технічне завдання;
- проектування, розробка (Elaboration) – уточнення плану, розуміння того, як ми це створюємо, проектування, планування необхідних дій і ресурсів, деталізація особливостей . У кінцевій ітерації – технічний проект;
- реалізація, створення системи (Construction) – етап розширення функціональності системи, закладеної в архітектурі. У кінцевій ітерації це - робочий проект;

Схему життєвого циклу проекту зображено на рисунку 1.10.

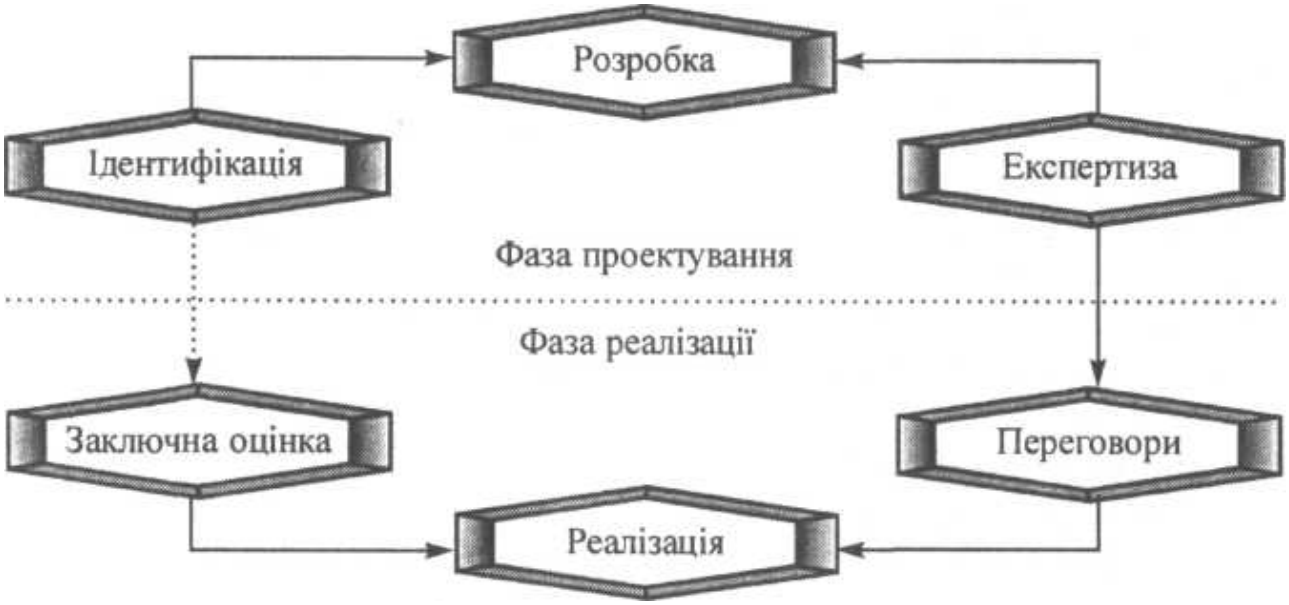


Рисунок 1.10 – Схема життєвого циклу проекту

- впровадження, розгортання (Transition). Створення кінцевої версії продукту. Фаза впровадження продукту, постачання товару конкретного користувача (тиражування, доставка і навчання).

Відповідно до кожного етапу, розробка даного сервісу поділяється на такі покрокові ітерації:

- розробка дизайну, мокап екранів додатків;
- написання серверної частини сервісу;
- ручне тестування(Postman/HTTP запити) написаного функціоналу;
- тестування відмовостійкості сервера;
- написання iOS додатку для офіціанта;
- написання iOS додатку для кухні;
- написання Unit тестів для серверної частини;
- написання Unit тестів для iOS додатків;
- написання тестів інтерфейсу користувача для Android додатків.

На етапі дизайну створюється дизай одного стилю і з використанням одних і тих же компонентів для додатку офіціанта та додатку для кухні. Продумується послідовність переходів між екранами додатків задля зручності користування та легкістю розширення у майбутньому в разі потреби.

На етапі написання серверної частини відбувається дослідження необхідних для використання технологій, а також вибір необхідної бази даних, оскільки сервер виступає у ролі агрегатора даних.

На етапах розробки додатку для користувача та додатку для лікаря відбувається інтеграція дизайну, яка була розроблена на початку життєвого циклу, підбір потрібних бібліотек для роботи із зображеннями, інтеграція бази даних додатку, задля зменшення навантаження на сервер.

Модульні тести реалізовувати швидше, ніж GUI тести (час на розробку менше). Модульні тести відпрацьовують швидше, ніж GUI.

Модульні тести дешевше, ніж GUI тести (проте, прагнучи до вершини піраміди, ми отримуємо велику впевненість в тому, що все працює як очікувалося).

Тестування має бути різнобічним, але співвідношення тестів повинна бути такою, що модульних кількісно більше, ніж GUI. Системні тести займають золоту середину.

На рисунку 1.11 зображено піраміду порядку тестування програмного забезпечення.

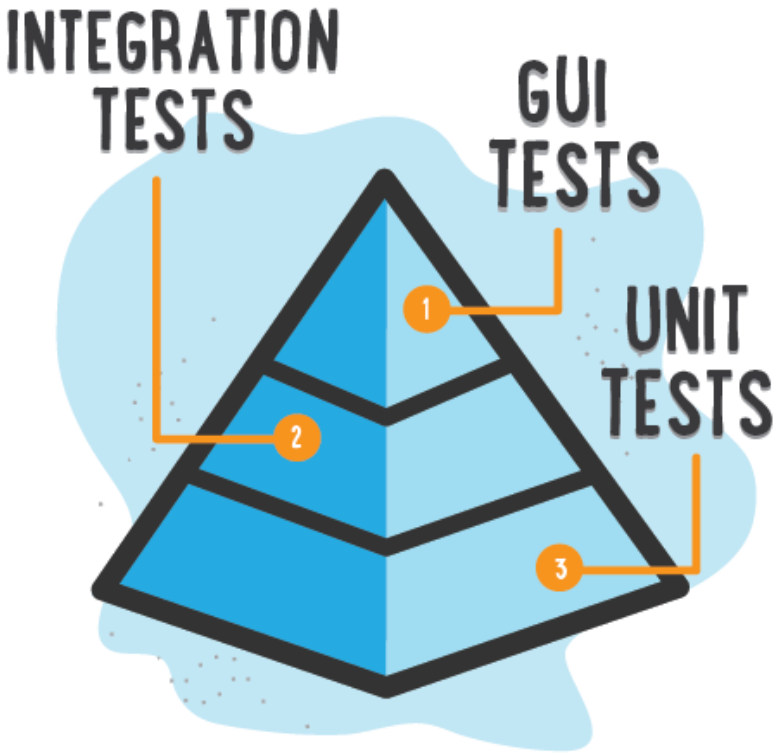


Рисунок 1.11 – Піраміда порядку тестування програмного забезпечення

На етапі Unit тестування сервера, відбувається написання тестів для бази даних, а також для контролерів.

На етапі Unit тестування додатків відбувається написання тестів для бази даних, шляхом запису у базу імітованих даних.

На етапі написання тестів стосовно інтерфейсу користувача відбувається написання тестів, які перевіряють події, які відбуваються при кліках офіціанта та персоналу кухні в своїх додатках.

1.1.6 Пошук актантів та варіантів використання.

У даному сервісі є 2 актори: офіціант та персонал кухні. Для кухні розробляється один додаток на весь персонал, оскільки планшет потрібен буде лише для перегляду замовлення та друку чека. Тому слід описати їх функціонал для розуміння взаємодії додатків та серверної частини із акторами.

Кожен з них має можливість використовувати певний функціонал сервісу, такий як: перегляд інформації про категорії страв, перегляд інформації про страви, їх ціни, склад, друк на принтер замовлення, тощо.

Основною метою розробки даного сервісу є економія часу при отриманні замовлення, а також подорожі офіціанта у потрібний цех кухні. Для офіціанта повинні бути забезпечені такі функції:

- реєстрація у додатку;
- авторизація у додатку;
- заповнення та редагування власного профілю;
- перегляд списку категорій страв;
- перегляд списку страв конкретної категорії;
- перегляд списку усіх страв;
- пошук по усіх стравах;
- пошук по стравах у категорії;
- сортування страв у ціновому діапазоні;
- сортування страв по їх інгрідієнтах;
- перегляд інформації про страву, її інгредієнти, час приготування, додатки до неї, тощо;

- додавання користувачів до певного столику;
- додавання страви до певного користувача;
- перегляд суми замовлення для кожного користувача;
- редагування замовлення;
- видалення замовлення;
- зміна статусу столика (вільно/зайнято/не визначено);
- відправка замовлення на кухню.

За допомогою додатку для кухні, персонал може бачити детальну інформацію про замовлення а саме скільки замовлень чекають на приготування, а також переглянути список страв у замовленні та роздрукувати чек для початку приготування замовлення. Під час розробки додатку для персоналу кухні повинні бути забезпечені такі функції:

- перегляд списку замовлень;
- перегляд інформації про замовлення та список страв у замовленні;
- відправка статусу готовності замовлення чи страви для конкретного замовлення;
- друкування чеку для початку роботи над замовленням.

Автоматизація ресторанного бізнесу надає реальну допомогу у виконанні багатьох функцій, які виконуються різними співробітниками підприємства — завідувачами виробництва, технологами, кухарями, комірниками, бухгалтерами, відповідальними за різні ділянки обліку.

Автоматизація кафе і ресторану дозволяє оптимізувати документообіг, забезпечити порядок на складі. У більшості співробітників ресторанів змінний графік роботи, і вони не завжди встигають обмінюватися інформацією — автоматизація ресторану вирішує цю проблему. Всі дані зберігаються в єдиній системі, користуватися якою просто та зручно. У даний час, особливо у великих містах, автоматизація ресторану є абсолютно необхідним рішенням для того, щоб обігнати конкурентів і зробити свій заклад кращим.

Для наглядності функціональних можливостей наведено приклад діаграми варіантів використання на рисунку 1.12.

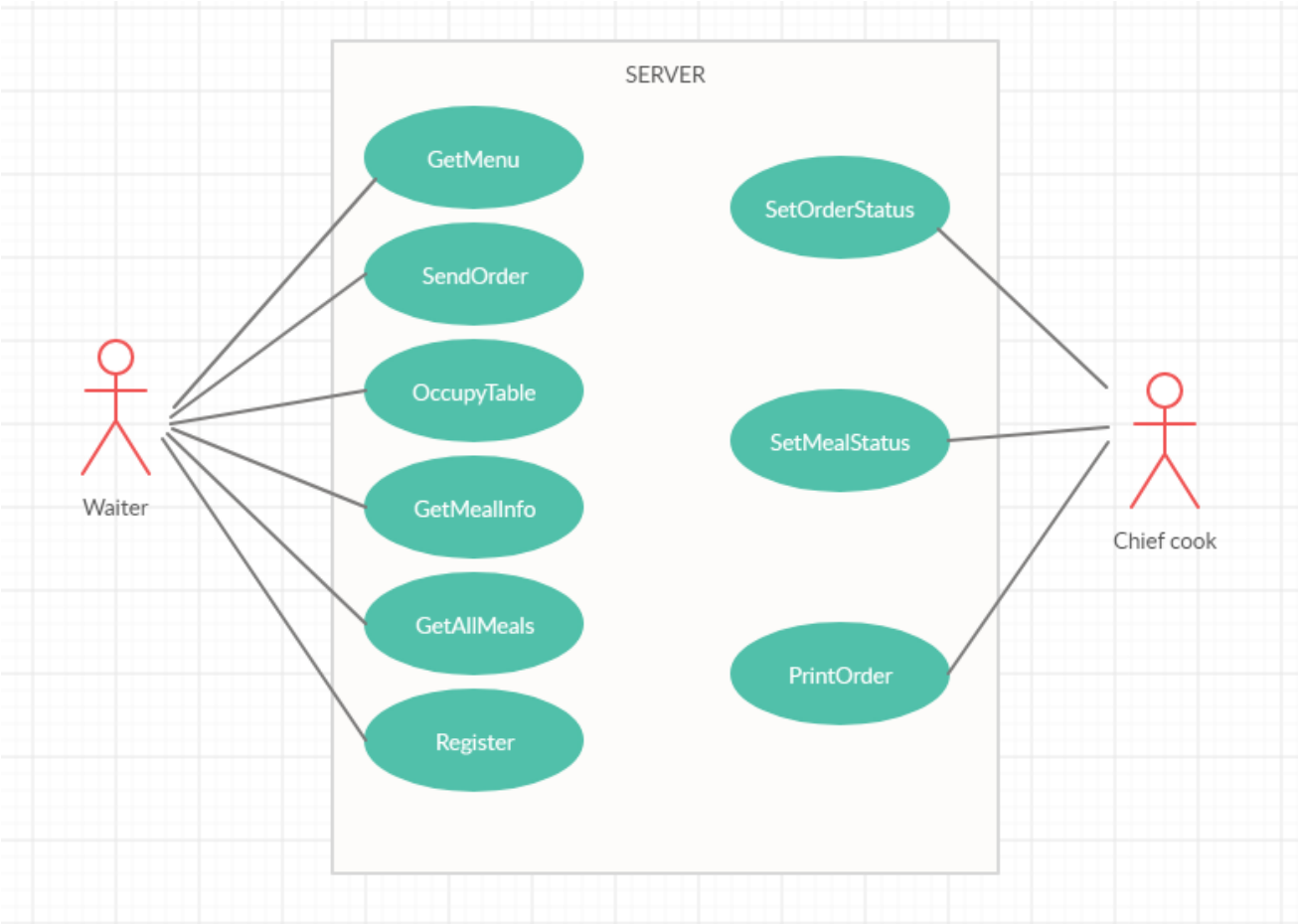


Рисунок 1.12 – Діаграми варіантів використання

Також, звертаючи увагу на те, що додатки взаємодіятимуть за допомогою сервера, слід поставити вимоги до сервера. Отже, сервер повинен забезпечувати:

– реєстрацію офіціантів;

– авторизацію офіціантів;

– зберігання даних про категорії страв;

– зберігання даних про страви, їх ціни, інгредієнти, тощо;

– зберігання інформації про статус виконання того чи іншого замовлення;

– зберігання інформації про статус готовності страви;

– зберігання у базі інформації про офіціантів(профіль офіціата);

– підтримка протоколу шифрованих даних у форматі AES-256.

Відповідно до наведеної діаграми можна помітити, що функціонал офіціанта є явно більшим, ніж функціонал для персоналу кухні. Це пояснюється тим, що для кухні обрано найпотрібніші функції, які будуть прискорювати та оптимізовувати роботу як кухарів, так і офіціантів. Саме тому функціонал для кухні не є дуже багатим.

Проте функціонал для офіціанта є більш обширним. Після реєстрації у додатку, офіціант потрапляє на головну сторінку додатку, де бачить список столів а також їх статус. Статус столику може бути такий:

- вільний;
- зайнятий;
- не визначено.

Кожен із статусів використовується у повсякденній роботі постійно, тому, якщо офіціант бачить статус «вільний», тоді можна сміливо запрошувати відвідувачів за цей столик та приймати замовлення.

Якщо статус «зайняти», тоді це значить, що за даним столиком уже є гості, які зробили замовлення.

А також, статус «не визначено». Цей статус відповідає за той момент, коли відвідувачі уже сіли за стіл, але ще не знають чи будуть вони у даному закладі, а лише визначаються. Також, даний статус буде увімкнено, як тільки гості посаджені за стіл, але не зробили ще замовлення чи знаходяться в процесі оформлення замовлення.

Як тільки гості зроблять замовлення та воно буде оформлене та звірене офіціантом, тільки тоді статус із «не визначено» зміниться на «зайнято» і гості будуть очікувати на приготування та подачу їхніх страв.

Кухарі після підтвердження замовлення отримують сповіщення на планшет, який знаходиться на кухні. Вони мають можливість перевірити замовлення та всі страви, а також змінити статус приготування страви чи замовлення. Таким чином, офіціант завжди буде знати і зможе надати достовірну інформацію стосовно замовлення, яке зробили гості.

На рисунку 1.13 зображено діаграму варіантів використання для офіціанта.

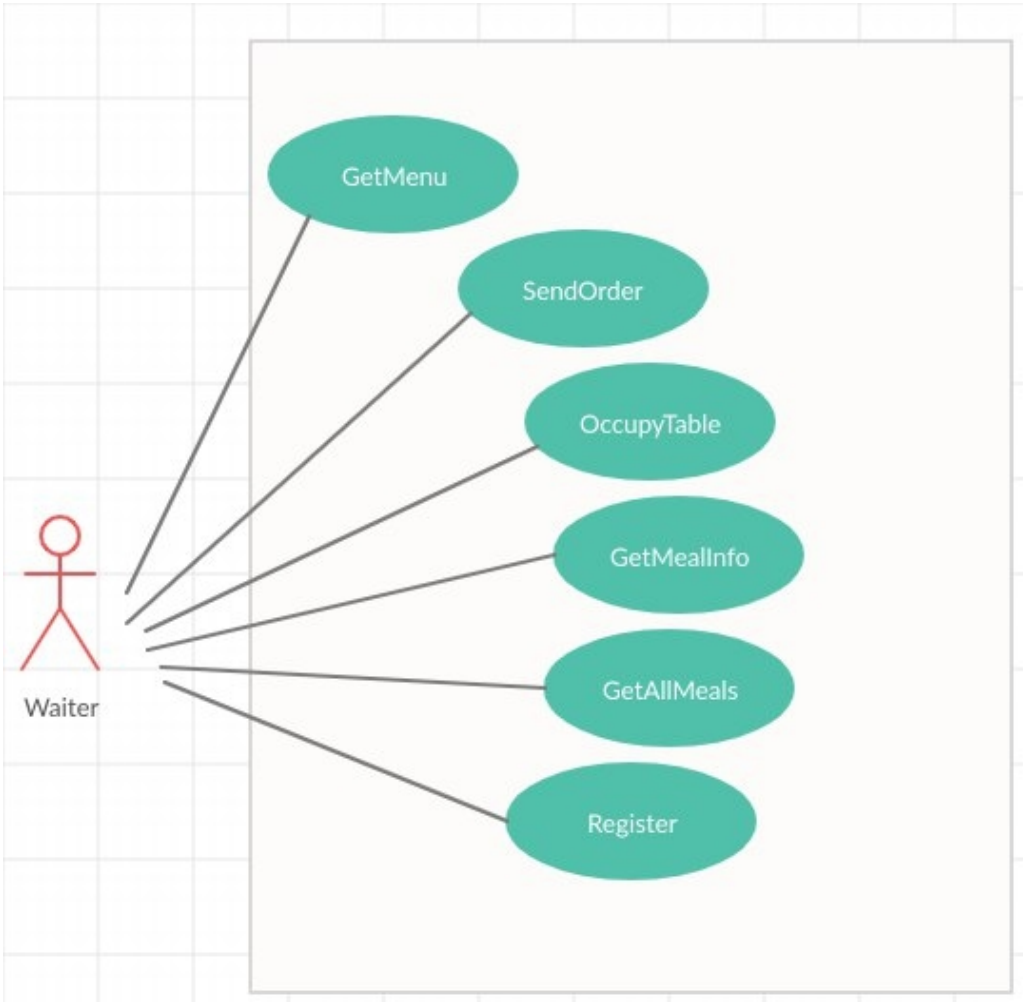


Рисунок 1.13 – Діаграма варіантів використання для офіціанта

Офіціант має змогу здійснити такі дії:

– отримати меню ресторану;

– відправити сформоване замовлення;

– змінити статус столика;

– отримати інформацію про категорії страв;

– отримати детальну інформацію про страву, таку як вага, склад, додатки до неї та інші;

– зареєструватись у додатку;

– авторизуватись у додатку.

На рисунку 1.14 зображено діаграму варіантів використання для працівників кухні.

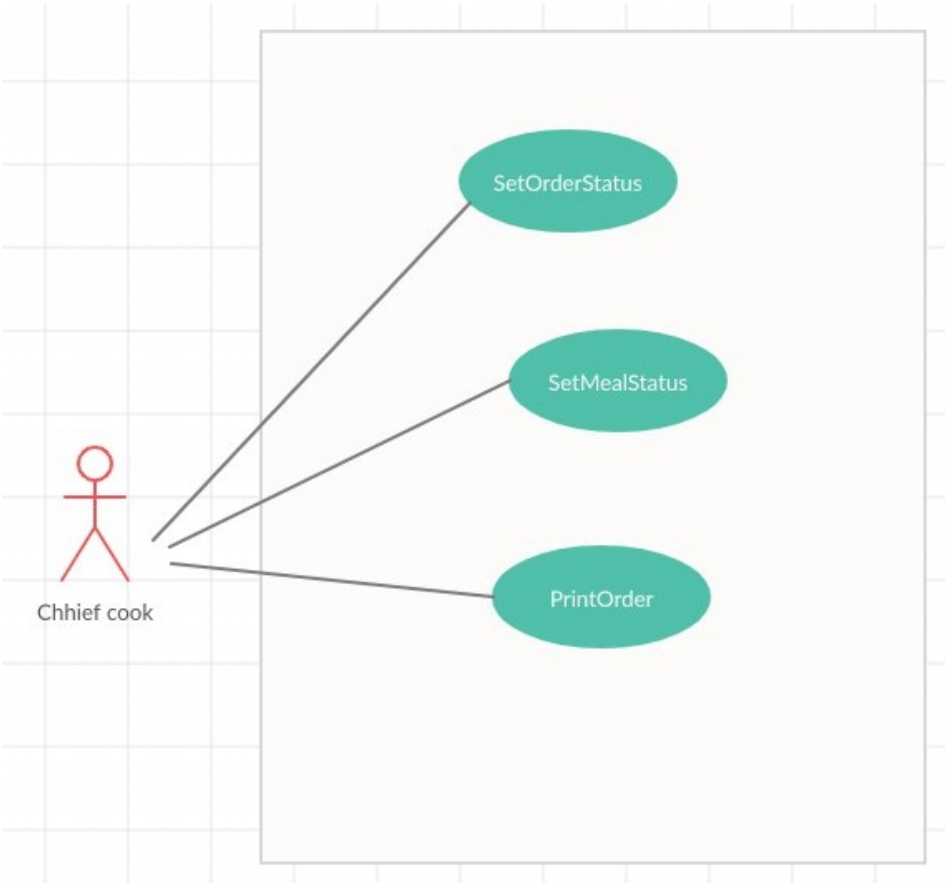


Рисунок 1.14 – Діаграма варіантів використання для працівників кухні

Додаток для кухарів буде встановлено на один планшет під керівництвом операційної системи iOS. Достатньо буде лише одного планшета на кухню, оскільки кухарі постійно зайняті роботою, а планшет лише потрібен для зміни статусу страви чи замовлення, а також для друкування замовлення на чек.

Зазвичай, на кухні видруковуються чеки із замовленням та кухарі ставлять їх десь поблизу, щоб завжди бачити список того, що потрібно приготувати. Тому дана функція буде дуже корисною, адже кухарі зможуть роз приділяти замовлення між собою або це робитиме шеф-повар та це допоможе прискорити час приготування страви. В якості принтера підійде будь-який принтер, що друкує чеки, хоча може і використовуватись повно габаритний принтер з друком формату А4.

1.2 Проектування програмної системи

1.2.1 Вибір процесу розробки.

Під час розробки програмного продукту використовувалась методологія управління IT-проектами Agile, яка забезпечила ітеративний підхід до виконання поставлених задач, а також її застосування допомогло продуктивно спланувати план робіт для виконання проекту.

Гнучка розробка програмного забезпечення – це клас методологій розробки програмного забезпечення, що базується на ітеративній розробці, в якій вимоги та розв'язки еволюціонують через співпрацю між багатофункціональними командами здатними до самоорганізації. Гнучка розробка – це засіб для підвищення продуктивності розробників програмного забезпечення.

Більшість гнучких методологій націлені на мінімізацію ризиків, шляхом зведення розробки до серії коротких циклів, що мають назву ітерацій, які зазвичай тривають один-два тижні. Кожна ітерація сама по собі виглядає як програмний проект в мініатюрі, і включає всі завдання, необхідні для видачі мінімального приросту за функціональністю: планування, аналіз вимог, проектування, кодування, тестування і документування. Хоча окрема ітерація, як правило, недостатня для випуску нової версії продукту, мається на увазі те, що гнучкий програмний проект готовий до випуску наприкінці кожної ітерації. Після закінчення кожної ітерації, команда виконує переоцінку пріоритетів розробки.

Agile акцентує увагу на безпосередньому спілкуванні «віч-на-віч». Більшість agile команд розташовані в одному офісі, його іноді називають bullpen. Як мінімум вона включає і «замовників» (замовники, які визначають продукт, також це можуть бути менеджери продукту, бізнес аналітики або клієнти). Офіс може також включати тестувальників, дизайнерів інтерфейсу, технічних авторів і менеджерів.

Основною метрикою agile методів є робочий продукт. Віддаючи перевагу безпосередньому спілкуванню, agile-методи зменшують обсяг письмової документації в порівнянні з іншими методами. Це привело до критики цих методів як недисциплінованих.

Основні ідеї методології Agile:

- особистості та їхні взаємодії важливіші, ніж процеси та інструменти;
- робоче програмне забезпечення важливіше, ніж повна документація;
- співпраця із замовником важливіша, ніж контрактні зобов'язання;
- реакція на зміни важливіша, ніж дотримання плану.

Agile розуміють як використання на проекті гнучких підходів, таких як Scrum, Kanban, Extreme Programming та інших, в яких вимоги і рішення розвиваються на основі співпраці між собою учасників самоорганізованої, крос-функціональної команди. Це сприяє адаптивному плануванню, еволюційному розвитку, ранньому наданню продукту, постійному вдосконаленню, дозволяє швидко і гнучко реагувати на зміни.

Plan the Next Iteration. Планування наступної ітерації — планування робіт для наступної ітерації.

Product Backlog. Список завдань по продукту — створення повного списку всіх загальних завдань, при реалізації яких ми отримаємо кінцевий продукт.

Sprint Planning. Планування спринта — планування завдань на Спринт, в якій береться кілька найважливіших загальних завдань з Product Backlog, які реалістично виконати протягом періоду стринта (1-4 тижні), ці завдання в свою чергу розбиваються на дрібніші і детальніші робочі завдання. Як результат, створюється Sprint Backlog.

Sprint Backlog. Список завдань на спринт — це список завдань, який визначено і погоджено на найближчий звітний період. Завдання в спринт-беклог беруться з Product Backlog.

Sprint. Спринт – це період, коли завдання виконуються: пишеться код, тестується, виправляються помилки, перетестовуються і т.д. Що не зроблене в поточному спринті потрапляє або в Product Backlog, або в наступний спринт, залежно від важливості. Спринти повторюються ітеративно (циклічно).

Potentially Shippable Software. Програмне забезпечення потенційно готове до відправки клієнту — в кінці спринту отримується готовий до показу клієнту продукт, демонструється клієнту, отримуються зауваження, що формує роботу на наступний спринт. Якщо клієнт повністю задоволений, чи поспішає, тоді він може забрати вже цей продукт.

1.2.2 Побудова схеми бази даних.

Побудова схеми бази даних є важливим етапом розвитку програмного продукту, оскільки потрібно визначити основні сутності та реалізувати найкращу схему зв’язків між ними.

Для розробки бази даних та серверної частини вцілому було обрано хмарний сервіс Cloud Firestore. Оскільки даний сервіс підтримується та покращується корпорацією Google, його використання є доцільним.

В той час, коли Firebase Realtime Database по-суті представляє собою JSON-файл, Cloud Firestore є більш структурованою альтернативою. Cloud Firestore – це база типу документа-моделі. Це означає, що всі дані зберігаються в об'єктах, які називаються документами, у вигляді ключ-значення. Де значення можу бути яким завгодно, від стрічкової змінної до бінарної. Такі документи групуються в колекції.

Cloud Firestore, буде мати один або декілька колекцій з документами, які будуть посилатися на інші "підколекції" зі своїми документами.

Нова структура даних дає кілька переваг при побудові запитів до бази.

По-перше: всі запити є неглибокими і можна просто витягти документ без необхідності затягнути всі дані, що містяться в будь-якому з пов'язаних субколекцій. Це означає, що можна зберігати свої дані ієрархічно в такий спосіб, що не доведеться турбуватися про завантаження тонн непотрібних даних.

По-друге: Cloud Firestore має набагато більше можливостей побудувати запити до бази даних Realtime Database. У реальному часі бази даних для фільтрації даних по декільком полям доводиться дуже сильно замислюватись та продумувати структуру даних або ж завантажувати інші дані і фільтрувати їх на клієнтській стороні.

1.2.3 ПобудоваUML діаграм класів.

Діаграма класів – статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Вона допомагає більш наглядно зобразити взаємозв’язок та структуру проекту.

На рисунку 1.15 зображено діаграму класів системи.

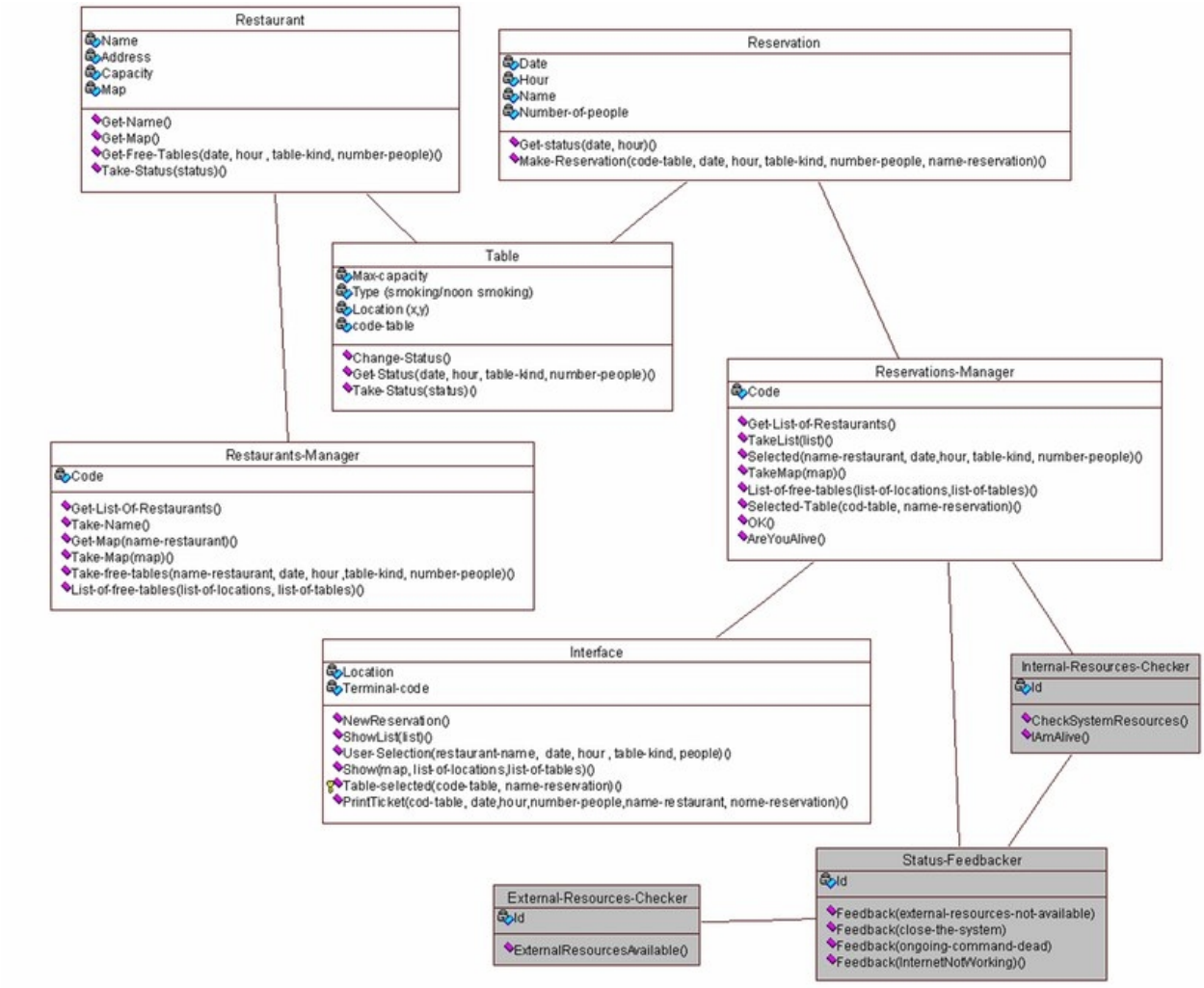
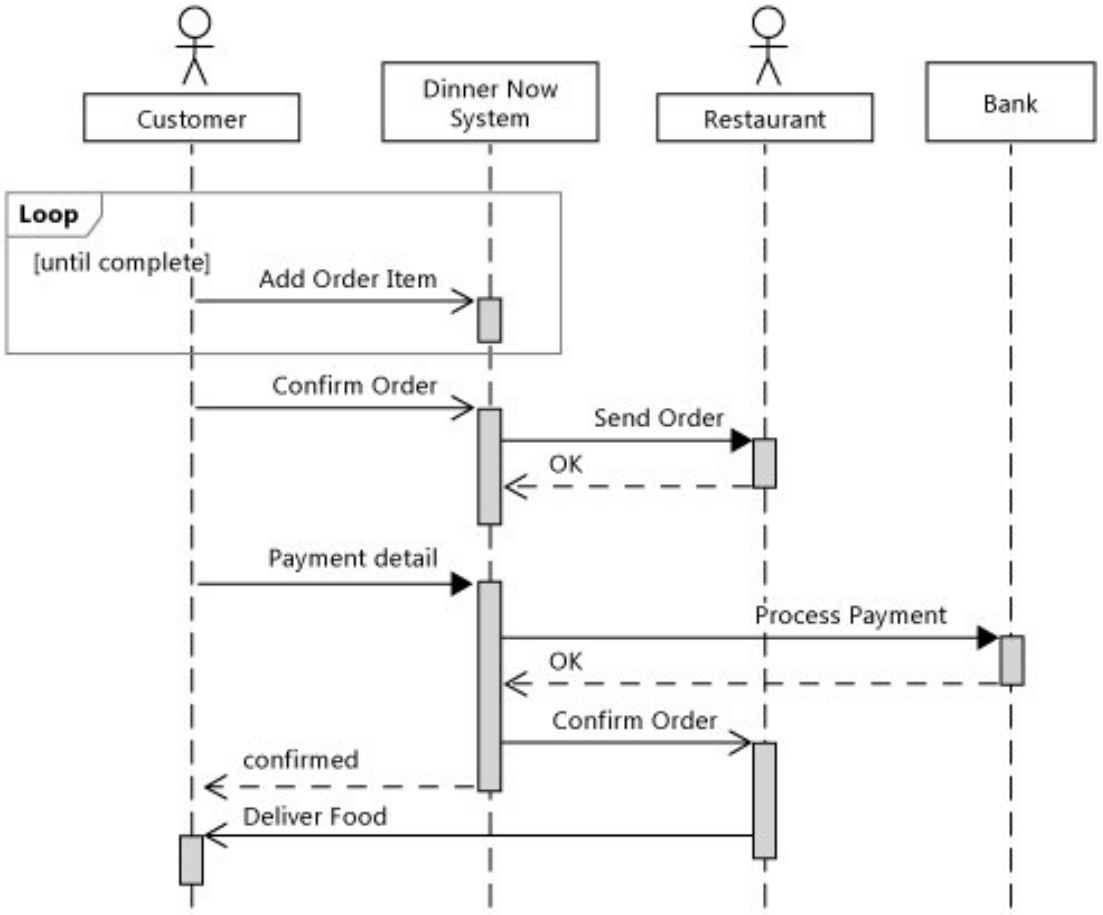


Рисунок 1.15 – Діаграма класів системи

Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм. Діаграма класів служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

На рисунку 1.16 зображено діаграму послідовності взаємодії офіціанта з додатком.



Рисуно 1.16 – Діаграма послідовності взаємодії офіціанта з додатком

Діаграма взаємодії – це одна з моделей опису поведінки взаємодіючих груп об'єктів в UML. Як правило, кожна окрема діаграма взаємодії описує поведінку тільки в межах одного варіанта використання. На такій діаграмі

прийнято відображати екземпляри об'єктів та повідомлення, якими ці об'єкти обмінюються один з одним в рамках даного варіанта використання.

Діаграма послідовності – різновид діаграми в UML. Діаграма послідовності відображає взаємодії об'єктів впорядкованих за часом. Зокрема, такі діаграми відображають задіяні об'єкти та послідовність відправлених повідомлень.

На рисунку 1.17 зображено діаграму послідовності отримання замовлення на кухні.

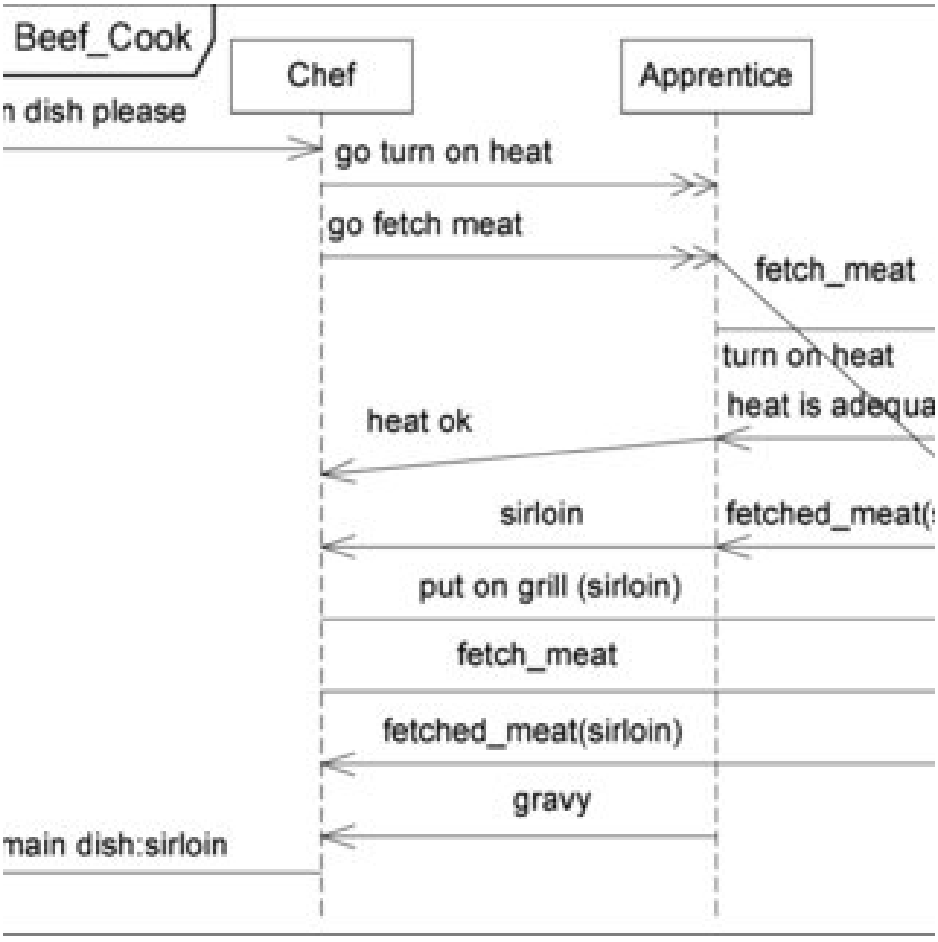


Рисунок 1.17 – Діаграма послідовності отримання замовлення на кухні

На діаграмі послідовностей показано у вигляді вертикальних ліній різні процеси або об'єкти, що існують водночас. Надіслані страви та замовлення зображуються у вигляді горизонтальних ліній, в порядку відправлення. Також, на діаграмі зображено рух потоку даних.

1.2.4 Моделювання архітектури системи.

Архітектура програмного забезпечення – це спосіб структурування програмної або обчислювальної системи, абстракція елементів системи на певній фазі її роботи. Система може складатись з кількох рівнів абстракції і мати багато фаз роботи, кожна з яких може мати окрему архітектуру.

Дослідження архітектури програмного забезпечення намагається визначити як найкраще розбити систему на частини, як ці частини визначають та взаємодіють одна з одною, як між ними передається інформація, як ці частини розвиваються поодинці і як все вищеописане найкраще записати використовуючи формальну чи неформальну нотацію.

Архітектура повинна будуватись щоб найкраще відповідати вимогам до системи що створюється, згідно принципу "форма відповідає функції".

Згідно Perry та Wolf архітектурою є набір елементів що мають певну форму (властивості і обмеження що накладаються на елементи), і їх обґрунтування. Обґрунтування фіксує мотиви вибору певного архітектурного стилю, елементів і обмежень. Рой Філдінг вважає що обґрунтування необхідне на етапі створення архітектури, і корисне надалі але не є невід'ємним її елементом. Тому що архітектура має набір властивостей що дозволяють їй задовольняти вимоги, і незнання цих вимог може призвести до змін що порушують архітектуру, але до архітектури входять властивості а не вимоги. Наприклад, можна не знати що в "архітектуру" стола закладену вимогу стійкості, і тому він повинен мати більше двох ніжок. Не знаючи про цю вимогу, ми можемо відпиляти забагато ніжок і стіл впаде. Але це тому що порушується архітектурне обмеження "мати три чи більше ніжок".

З огляду на мету зменшення трудовитрат на розробку складного програмного забезпечення, припустимо, що необхідно використовувати готові уніфіковані рішення. Адже шаблонність дій полегшує комунікацію між розробниками, дозволяє посилатися на відомі конструкції, знижує кількість помилок. Згідно «Вікіпедії», патерн – це повторна архітектурна конструкція,

що представляє собою рішення проблеми проектування в рамках деякого часто виникаючого контексту.

Варто почати з головного – Model-View-Controller. MVC – це фундаментальний патерн, який знайшов застосування в багатьох технологіях, дав розвиток нових технологій і кожен день полегшує життя розробникам.

Вперше патерн MVC з'явився в мові SmallTalk. Розробники повинні були придумати архітектурне рішення, яке дозволяло б відокремити графічний інтерфейс від бізнес логіки, а бізнес логіку від даних. Таким чином, в класичному варіанті, MVC складається з трьох частин, які і дали йому назву. Складові MVC:

- модель;
- представлення;
- контролер.

Під Моделлю, зазвичай розуміється частина, яка містить в собі функціональну бізнес-логіку програми. Модель повинна бути повністю незалежна від інших частин продукту. Модельний шар нічого не повинен знати про елементи дизайну, і яким чином він буде відображатися. Досягається результат, що дозволяє змінювати подання даних, то як вони відображаються, не чіпаючи саму модель.

Модель володіє наступними ознаками:

- модель – це бізнес-логіка програми;
- модель володіє знаннями про себе саму і не знає про контролерах і уявленнях;
- для деяких проектів модель – це просто шар даних (DAO, база даних, XML-файл);
- для інших проектів модель – це менеджер бази даних, набір об'єктів або просто логіка додатка.

Далі можна перейти до частини представлення. В обов'язки представлення входить відображення даних отриманих від моделі. Однак,

представлення не може безпосередньо впливати на модель. Можна говорити, що представлення отримує доступ «тільки на читання» до даних.

Представлення володіє наступними ознаками:

- у представленні реалізується відображення даних, які виходять від моделі будь-яким способом;
- в деяких випадках, представлення може мати код, який реалізує деяку бізнес-логіку.

Для розробки під операційну систему Android, зазвичай, використовується архітектурний паттерн MVC – Model-View-Controller.

Даний підхід дозволяє створювати абстракцію уявлення. Для цього необхідно виділити інтерфейс уявлення з певним набором властивостей і методів. Контролер, в свою чергу, отримує посилання на реалізацію інтерфейсу, підписується на події вистави і за запитом змінює модель. Схему взаємодії між компонентами архітектури наведено на рисунку 1.18.

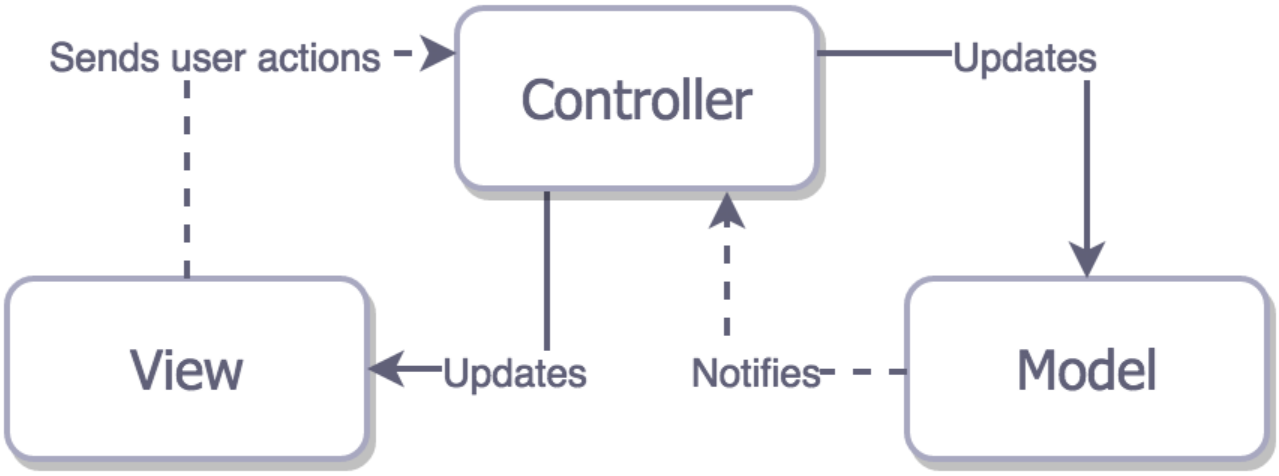


Рисунок 1.18 – Схема взаємодії між компонентами архітектури

Отже, ознаками контролера є:

- двостороння комунікація з представленням;
- представлення взаємодіє безпосередньо з контролера, шляхом виклику відповідних функцій або подій екземпляру контролера;

- контролер взаємодіє з View шляхом використання спеціального інтерфейсу, реалізованого представленням;
- один екземпляр контролера пов'язаний з одним представленням.

Кожне представлення має реалізовувати відповідний інтерфейс. Інтерфейс представлення визначає набір функцій і подій, необхідних для взаємодії з користувачем (наприклад, IView.ShowErrorMessage (string msg)). Контролер повинен мати посилання на реалізацію відповідного інтерфейсу, яку зазвичай передають в конструкторі.

Логіка представлення повинна мати посилання на екземпляр контролер. Всі події вистави передаються для обробки в контролер і практично ніколи не обробляються логікою представлення (в т.ч. створення інших представлень).

Model-View-ViewModel – це шаблон проектування, що застосовується під час проектування архітектури застосунків (додатків). Публічно вперше був представлений Джоном Госсманом (John Gossman) у 2005 році як модифікація шаблону Presentation Model. MVVM орієнтований на такі сучасні платформи розробки, як Windows Presentation Foundation та Silverlight від компанії Microsoft.

MVVM полегшує відокремлення розробки графічного інтерфейсу від розробки бізнес логіки (бек-енд логіки), відомої як модель (можна також сказати, що це відокремлення представлення від моделі). Модель представлення є частиною, яка відповідає за перетворення даних для їх подальшої підтримки і використання. З цієї точки зору, модель представлення більше схожа на модель, ніж на представлення і оброблює більшість, якщо не всю, логіку відображення даних. Модель представлення може також реалізовувати патерн медіатор, організовуючи доступ до бек-енд логіки навколо множини правил використання, які підтримуються представленням.

Функції архітектури MVVM:

- здійснювати зв'язок між моделлю та вікном;
- відслідковувати зміни в даних, що зроблені користувачем;
- відпрацьовувати логіку роботи View (механізм команд).

Взаємодію компонентів архітектури MVVM зображено на рисунку 1.19

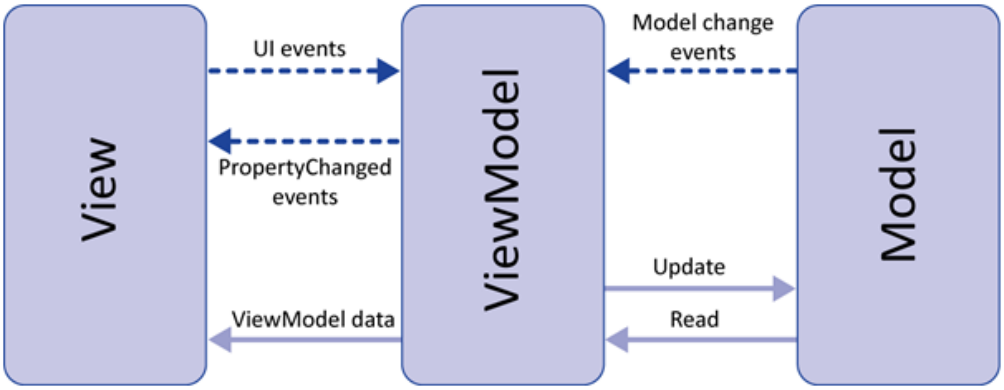


Рисунок 1.19 – Взаємодія компонентів архітектури MVVM

Даний підхід дозволяє пов'язувати елементи представлення з властивостями і подіями View-моделі. Можна стверджувати, що кожен шар цього патерну не знає про існування іншого шару.

Ознаки архітектури MVVM:

- двостороння комунікація з представленням;
- View-модель – це абстракція представлення. Зазвичай означає, що властивості представленням збігаються із властивостями View-моделі / моделі
- View-модель не має посилання на інтерфейс уявлення (IView). Зміна стану View-моделі автоматично змінює уявлення і навпаки, оскільки використовується механізм скріплення даних (Bindings);
- один екземпляр View-моделі пов'язаний з одним представленням.

При використанні цього патерну, представлення не реалізує відповідний інтерфейс (IView), оскільки взаємодія відбувається за допомогою інтерфейсу Observer, який надає інформацію про стан отримання даних.

Подання повинно мати посилання на джерело даних (DataContex), яким в даному випадку є View-модель. Елементи уявлення пов'язані (Bind) з відповідними властивостями і подіями View-моделі.

У свою чергу, View-модель реалізує спеціальний інтерфейс, який використовується для автоматичного оновлення елементів уявлення.

Отже, для розробки додатків було обрано архітектурний паттерн MVVM, оскільки він забезпечуватиме більшість потреб пов’язаних із життєвим циклом екранів додатків.

1.3 Конструювання програмної системи

1.3.1 Вибір мови та середовища розробки.

Swift — багатопарадигмова компільована мова програмування, розроблена компанією Apple для того, щоб співіснувати з Objective C і бути стійкішою до помилкового коду. Swift була представлена на конференції розробників WWDC 2014. Мова побудована з LLVM компілятором, включеного у Xcode 6 beta. Безкоштовний посібник мови програмування Swift доступний для завантаження у магазині iBooks.

Компілятор Swift побудований з використанням технологій вільного проекту LLVM. Swift успадковує найкращі елементи мов C і Objective-C, тому синтаксис звичний для знайомих з ними розробників, але водночас відрізняється використанням засобів автоматичного розподілу пам'яті і контролю переповнення змінних і масивів, що значно збільшує надійність і безпеку коду.

При цьому Swift-програми компілюються у машинний код, що дозволяє забезпечити високу швидкодію. За заявою Apple, код Swift виконується в 1.3 рази швидше коду на Objective-C. Замість збирача сміття Objective-C в Swift використовуються засоби підрахунку посилань на об'єкти, а також надані у LLVM оптимізації, такі як автовекторизація.

Мова також пропонує безліч сучасних методів програмування, таких як замикання, узагальнене програмування, лямбда-вирази, кортежі і словникові типи, швидкі операції над колекціями, елементи функційного програмування. Основним застосуванням Swift є розробка користувацьких застосунків для macOS, iOS, tvOS, watchOS з використанням тулкіта Cocoa і Cocoa Touch. При

цьому Swift надає об'єктну модель, сумісну з Objective-C. Сирцевий код мовою Swift може змішуватися з кодом на C і Objective-C в одному проекті.

Swift щільно інтегровано до власницького середовища розробки Xcode, проте може бути викликано з терміналу, що уможливлює її використання на операційний системах, відмінних від macOS, наприклад, на Linux.

Окремо варто відзначити, що Swift від компанії Apple не варто плутати з досить давно розроблюваною скриптовою мовою Swift, націленої на багатонитеве програмування і поставленого під вільною ліцензією Apache.

Отже, слід навести переваги мови swift:

По-справжньому досвідчених програмістів на swift немає досі. Вчитися і перевчатися доводиться всім. Якщо зайнятися Swift в 2015-му, то через скільки завгодно років будете відставати від найдосвідченіших програмістів в гіршому випадку на рік. Перевага досвідчених кодерів тільки в тому, що Swift запозичує кращі елементи інших мов — тим, хто їх знає, легше її освоїти.

Це молода мова. У нових версіях Swift, як і раніше відбуваються фундаментальні зміни в синтаксисі і типах даних. Мова дорослішає на очах, і так легше розуміти логіку змін, а при бажанні можна і допомогти у її доопрацюванні. Звичайно, хтось може назвати це недоліком, враховуючи, що проекти, написані на більш старих версіях Swift, не працюють на нових — але міграція займає не так багато часу, а синтаксис стає тільки простішим.

Це безпечна мова. В основі Swift — бажання залишити якомога менше вразливостей і максимально убезпечити код від помилок. Програміст не зобов’язаний все тримати в голові і стежити за всім. Це необхідно у C-подібних мовах, і якщо раніше це було обґрунтованим, тому що вони працювали швидше інших, то Swift наздоганяє по продуктивності навіть C++ — адже чим далі, тим краще вона буде оптимізована. Програміст, який пише код Swift, позбавлений величезної кількості головного болю, пов’язаного з менеджментом пам’яті і іншими речами. Саме тому після переходу з Objective-C дана мова програмування буде доставляти лише задоволення та систему спрощеного написання коду.

Легко читається синтаксис, натхненний python і ruby. Основна відмінність між, наприклад, Python (яка також доволі легко читається) і SWIFT в тому, що блоки не відокремлюються відступами, а фігурними дужками.

Це гарна мова. Дуже суб'єктивно, звичайно, але код Swift виглядає красиво — чого не скажеш про той же Objective-C з його нескінченними нагромадженнями. А головне, що краса і зручність не позначаються на потужності.

На рисунку 1.20 зображено логотип мови програмування swift.

Рисунок 1.20 – Логотип мови програмування swift

Також для розробки під мобільні платформи, а саме під операційну систему iOS використовується середовище програмування XCode. Особливістю розробки додатків під цю операційну систему є те, що розробка може здійснюватись виключно на ноутбуках фірми Apple або на комп'ютерах, на яких встановлена операційна система Hackintosh.

Отже, Xcode — це інтегроване середовище розробки (IDE) виробництва Apple. Дозволяє створювати програмне забезпечення з використанням таких технологій як GCC, GDB, Java та ін. На сьогодні є єдиним засобом написання «універсальних»(Universal Binary) прикладних програм для Mac OS X, а також мобільних пристроїв фірми Apple.

Xcode включає в себе більшу частину документації розробника від Apple та Interface Builder — застосунок, який використовується для створення графічних інтерфейсів.

Пакет Xcode містить змінену версію вільного набору компіляторів GNU Compiler Collection і підтримує мови C, C++, Objective-C, Swift, Java, AppleScript, Python і Ruby з різними моделями програмування, включаючи (але не обмежуючись) Cocoa, Carbon і Java. Сторонніми розробниками реалізована підтримка GNU Pascal, Free Pascal, Ada, C #, Perl, Haskell і D. Пакет Xcode використовує GDB як back-end для свого налагоджувача.

У серпні 2006 Apple оголосила про те, що DTrace, фреймворк динамічного трасування від Sun Microsystems, випущений як частина OpenSolaris, буде інтегрований в Xcode під назвою Xray. Пізніше Xray був перейменований в Instruments. На рисунку 1.21 зображено середовище розробки XCode.

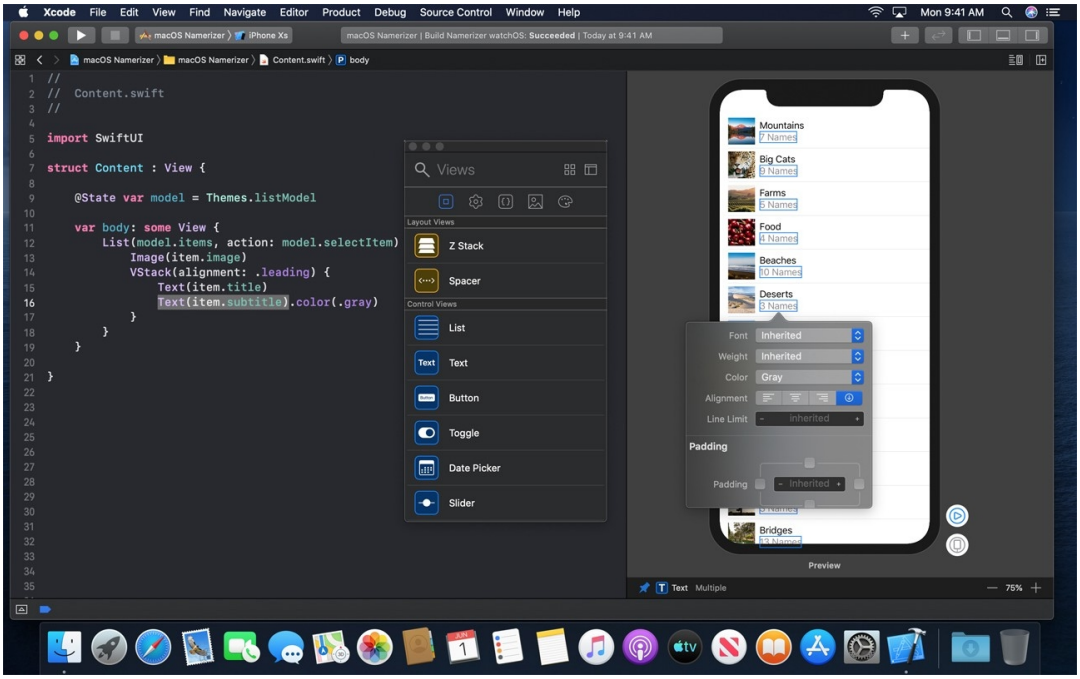


Рисунок 1.21 – середовище розробки XCode

4 червня 2018 року на WWDC 2018 була представлена бета-версія нового Xcode 10. У новій версії реалізована підтримка темної теми нової ОС

macOS Mojave, підтримка кастомних наборів Instruments, покращений Interface Builder - на 40% швидше відкриття документів і на 30% швидше робота, в ньому плаває нове вікно Library, яке замість Inspector почне представляти властивості об'єктів, з'явився новий NSGridView, який може впорядковувати View в таблицях, як в Numbers, також з'явився поліпшений редактор коду, підтримка Bitbucket Cloud, Bitbucket Server і GitLab, покращений дебагер, нові можливості автоматическо го тестування, новий збирач, який використовує на 20% менше пам'яті, і в два рази зменшує час повторних збірок. Крім того з'явилися API для комп'ютерного зору - виявлення об'єктів, визначення осіб, штрих-кодів; API для обробки мови - сенс пропозицій, виділення назв і т.п .; Core ML (англ.) Рос. 2 - швидший і настроюється фреймворк для використання машинного навчання і нейронних мереж; Create ML - фреймворк для тренування нейронних мереж, значно зменшує моделі і спрощує їх створення; ARKit (англ.) Рос. 2 - нова версія фреймворку доповненої реальності.

1.3.2 Вибір СКБД та опис її фізичної моделі.

В якості системи керування базою даних та самою базою даних було обрано сервіс Cloud Firestore. Оскільки даний сервіс має безліч переваг, а також добре відомий своєю простотою та швидкодією для невеликих проектів забезпечує хороший зв’язок хмарної бази даних із додатками.

Firebase — це платформи розробки мобільних та веб-застосунків. Firebase розвивається з 2011 року компанією Firebase Inc., яка була придбана Google у 2014. Також, це пакет інструментів розробки додатків (SDK) і засіб аналітики Google. Зв’язавши облікові записи Firebase і Google Ads між собою, можна отримати потужні інструменти для оцінювання ефективності кампаній Google Ads. На рисунку 1.22 зображено архітектуру побудови веб-застосувань на основі платформи Firebase.

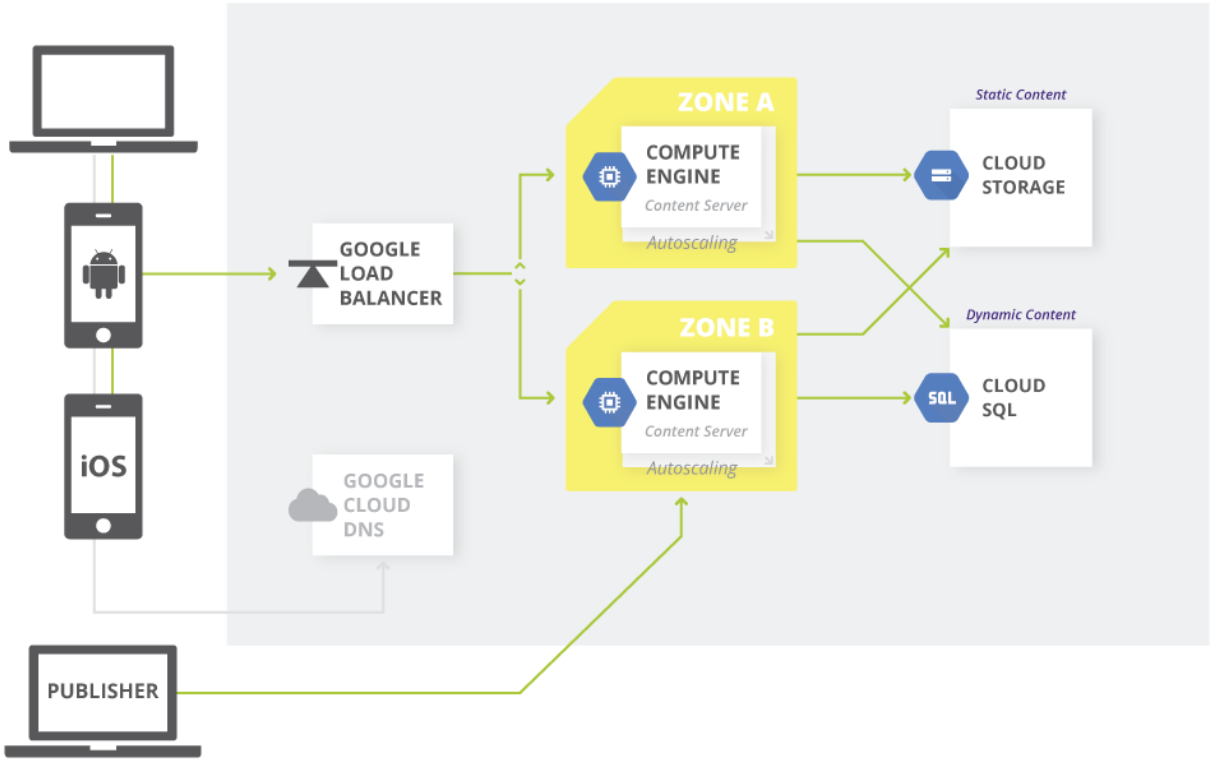


Рисунок 1.22 – архітектура побудови веб-застосувань на основі платформи Firebase.

Firebase веде свої витоки з Envolve, попереднього стартапу, заснованого Джеймсом Темпліном та Ендрю Лі в 2011 році. Envolve надав розробникам API, який дозволяв інтегрувати функціональні можливості онлайн-чатів на свої веб-сайти. Після випуску сервісу чату, Тамплін і Лі виявили, що сервіс використовуються для передачі даних застосунків, які не були повідомленнями чату. Розробники використовували Envolve для синхронізації даних застосунків, таких як стан гри в режимі реального часу серед своїх користувачів. Тамплін і Лі вирішили відокремити систему чату та архітектуру реального часу, яка працювала на ньому. Вони заснували Firebase як окрему компанію в квітні 2012 року.

Служби і готові рішення для розробки:

- Firebase Analytics;
- Firebase Cloud Messaging;
- Firebase Auth;
- Realtime Database;

– Firebase Storage;

– Firebase Hosting та Functions;

– ML Kit.

Firebase Analytics — безплатне рішення для оцінки застосунків, яке дає змогу ознайомитись із використанням застосунків та залученням користувачів, його вигляд зображено на рисунку 1.23.

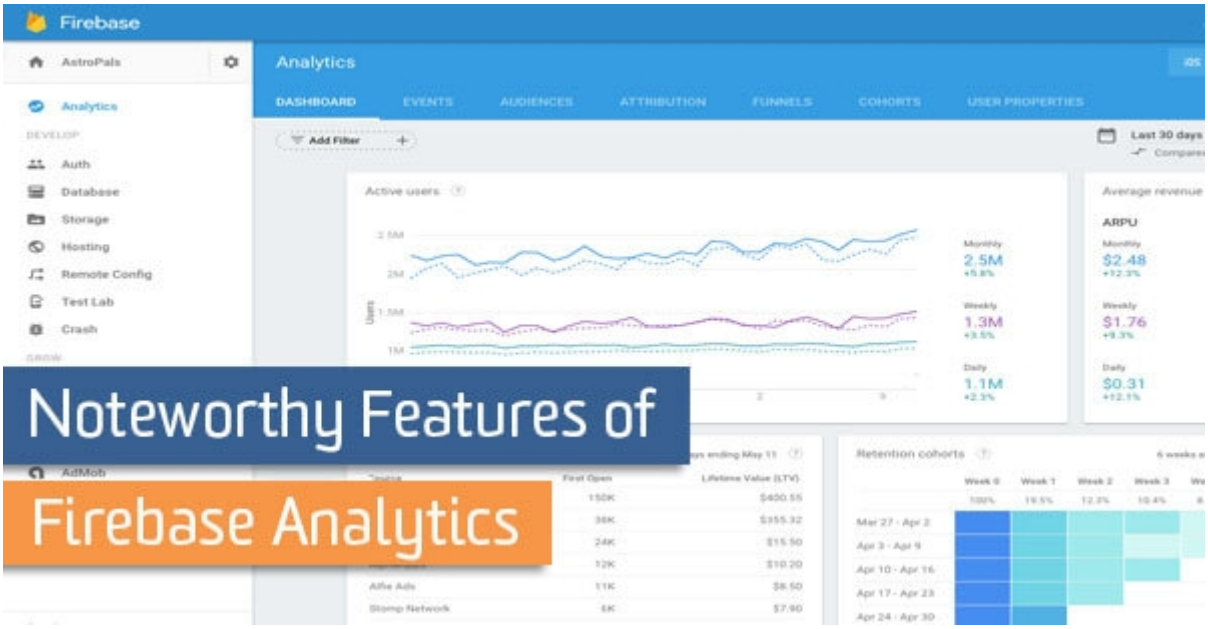


Рисунок 1.23 – Вигляд Firebase Analytics

Firebase Cloud Messaging – це крос-платформне рішення(раніше відоме як Google Cloud Messaging (GCM), Firebase Cloud Messaging (FCM)) для повідомлень і нотифікацій для Android, iOS та веб-застосунків, які наразі можна використовувати безкоштовно.

Firebase Auth – це служба, яка може аутентифікувати користувачів, використовуючи лише код на стороні клієнта. Він підтримує соціальні логін-провайдери Facebook, GitHub, Twitter і Google (і Google Play Games). Крім того, вона включає в себе систему управління користувачами, за допомогою якої розробники можуть ввімкнути автентифікацію користувача за допомогою входу з електронної пошти та пароля, що зберігаються в Firebase.

Realtime Database – Firebase надає в режимі реального часу базу даних та бекенд як службу. Ця служба надає розробникам застосунків API, який дозволяє синхронізувати дані застосунків між клієнтами та зберігати їх у хмарі Firebase. Схему роботи даної бази даних зображено на рисунку 1.24

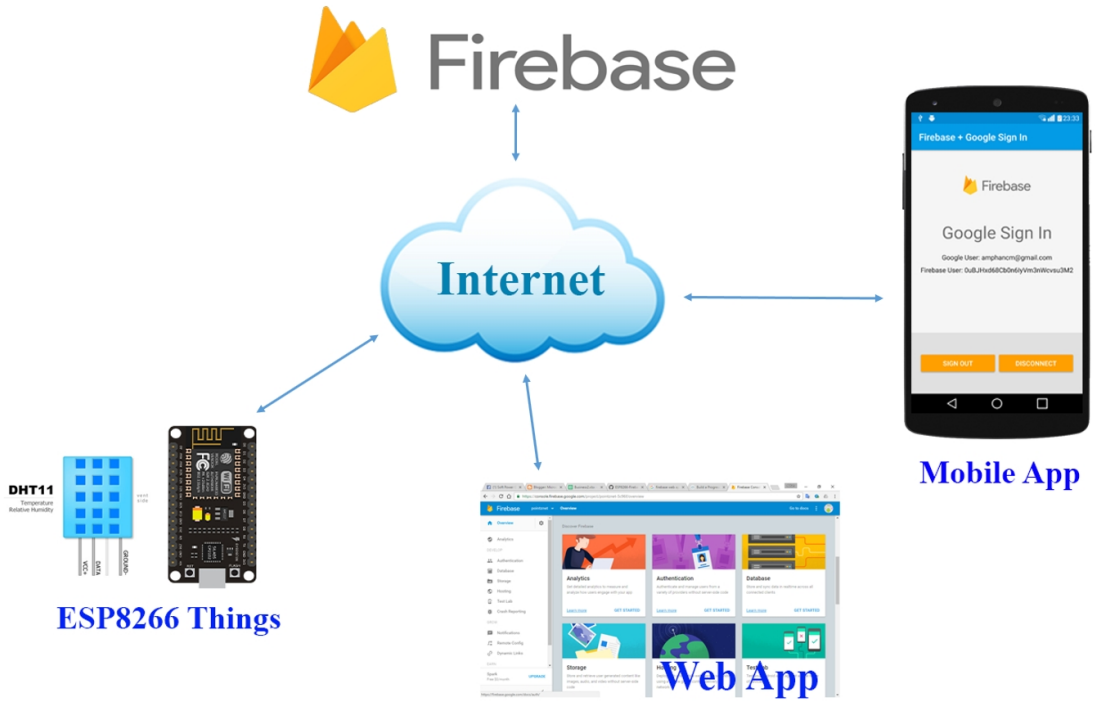


Рисунок 1.24 – Схема роботи Realtime Database

Компанія також надає клієнтські бібліотеки, які дозволяють інтеграцію із застосунками Android, iOS, JavaScript / Node.js, Java, Objective-C, Swift. База даних також доступна через REST API та прив'язки до декількох сценаріїв JavaScript, таких як AngularJS, React, Ember.js та Backbone.js. REST API використовує протокол подій із сервером, який є інтерфейсом для створення HTTP-з'єднань для отримання push-повідомлень від сервера. Розробники, які використовують Realtime Database, можуть захищати свої дані за допомогою правил безпеки, що застосовуються на сервері. Cloud Fiestore, яка є наступною генерацією Firebase Realtime Database, була випущена у бета-версії.

ML Kit – це мобільна система машинного навчання для розробників, яка була запущена в режимі бета-тестування 8 травня 2018 року під час Google I/O 2018.

ML Kit API містить різноманітні інструменти, серед яких розпізнавання тексту, розпізнавання облич, сканування баркодів, створення опису для зображень та розпізнавання наземних об'єктів. Наразі вона доступна для iOS та Android розробників. Також можливий імпорт власних моделей TensorFlow. API можна використовати у пристрої або у хмарі, що значно підвищує гнучкість та доступність сервісу, оскільки виконання аналізу у хмарі є дуже важливим моментом, коли у користувача не достатньо потужний мобільний девайс. На рисунку 1.25 зображено головне меню ML Kit.

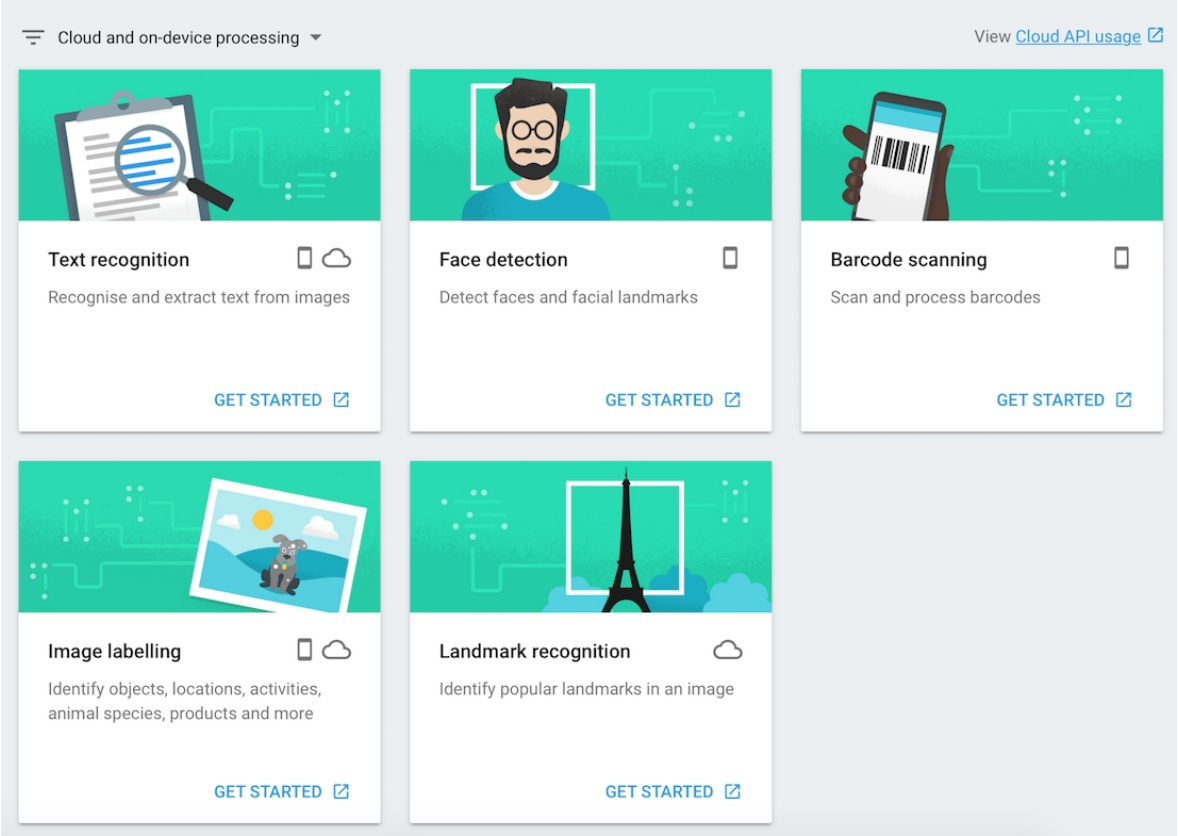


Рисунок 1.25 – Головне меню ML Kit

Firebase Storage – Firebase Storage забезпечує надійне завантаження та вивантаження файлів для застосунків Firebase незалежно від якості мережі. Розробник може використовувати його для зберігання зображень, аудіо-, відео- чи іншого вмісту, створеного користувачами. Зберігання Firebase підтримується Google Cloud Storage.

Firebase Inc. отримав початкове фінансування у травні 2012 року. Компанія також збільшила фінансування у червні 2013 року. У жовтні 2014 року компанія Firebase була придбана компанією Google. У жовтні 2015 року компанія Google придбала Divshot, щоб об'єднати її з командою Firebase.

На рисунку 1.26 зображено модель взаємодії сервісу Firebase з мобільними девайсами.

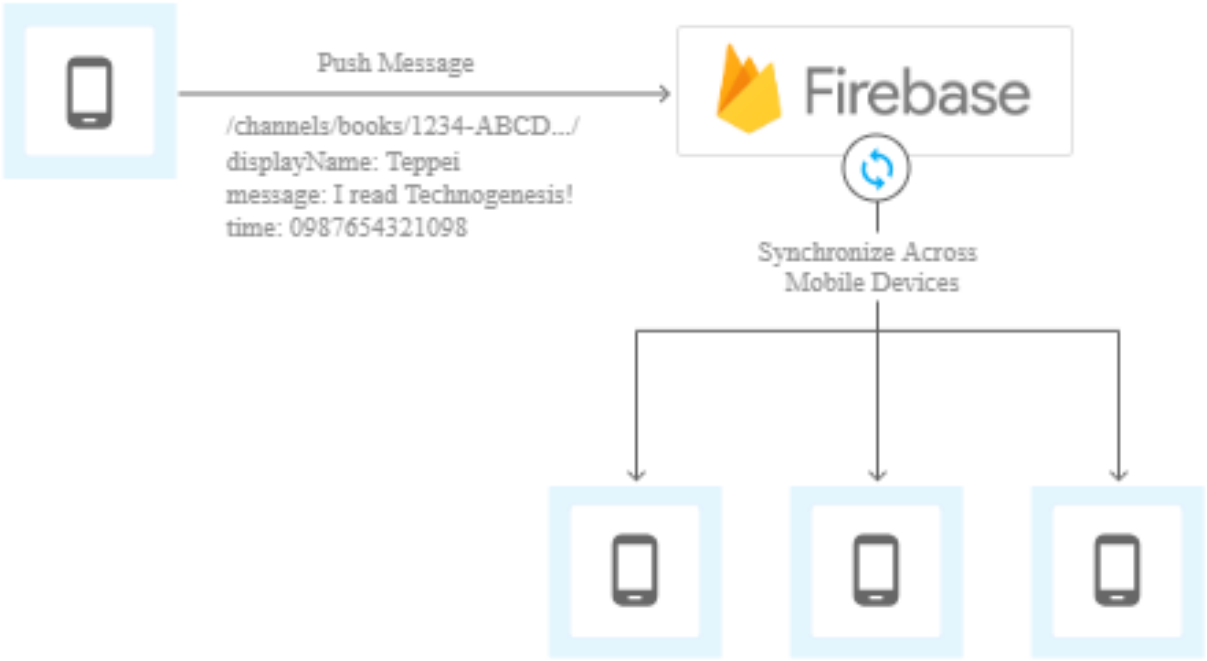


Рисунок 1.26 – Модель взаємодії сервісу Firebase з мобільними девайсами

Firebase Hosting та Functions – це статичний та динамічний веб-хостинг, який було запущено 13 травня 2014 року. Він підтримує хостинг статичних файлів, таких як CSS, HTML, JavaScript та інші файли, а також динамічну підтримку Node.js через Cloud Functions. Служба передає файли через мережу доставки контенту (CDN) за допомогою протоколу HTTPS та шифрування SSL. Firebase підтримує Fastly, CDN, щоб забезпечити підтримку CDN Firebase Hosting. Компанія стверджує, що хостинг Firebase вирiс із запитів клієнтів; розробники використовували Firebase для своєї бази даних в режимі реального часу, але вони потребували місця для розміщення їхнього вмісту.

З моменту придбання Firebase виросла всередині Google і розширила їхні послуги, щоб стати єдиною платформою для мобільних розробників. Firebase тепер інтегрується з різними іншими службами Google, щоб пропонувати нові продукти та масштаб для розробників. У січні 2017 року компанія Google придбала Fabric і Crashlytics з Twitter, щоб приєднати ці служби до команди Firebase. Firebase у жовтні 2017 року запустив Cloud Firestore, документ-орієнтовану базу даних. Ця база даних має свої переваги та недоліки. Хоча, переваги є надто суттєвими, щоб не скористатись даним хмарним рішенням для розробки невеликих сервісів.

Також, існують проекти з відкритим кодом, такі як:

- Firepad;
- Firechat;
- GeoFire.

Firepad — це редактор для спільної роботи у режимі реального часу із відкритим кодом. Випущений під ліцензією MIT, Firepad використовується декількома редакторами, включаючи редактор Atlassian Stash Realtime Editor та Koding.

Firechat — це програма чату з відкритим кодом у режимі реального часу. Він випущений під ліцензією MIT.

GeoFire — це бібліотека з відкритим кодом, яка використовує Firebase Realtime Database, що дозволяє розробникам застосунків зберігати та запитувати набір ключів на основі географічного розташування.

ValueEventListener() до посилання бази даних. Ця подія буде запускатися кожного разу, коли в реальному часі відбувається зміна даних. У onDataChange() можна виконувати бажані операції на нові дані, оновити їх, завантажити актуальний список даних, здійснити відображення прогресу завантаження на екрані, тощо.

На рисунку 1.27 наведено слухач події, який запускається кожного разу, коли змінюються дані профілю користувача.

```
mDatabase.child(userId).addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {

        User user = dataSnapshot.getValue(User.class);

        Log.d(TAG, "User name: " + user.getName() + ", email " + user.getEmail());
    }

    @Override
    public void onCancelled(DatabaseError error) {
        // Failed to read value
        Log.w(TAG, "Failed to read value.", error.toException());
    }
});
```

Рисунок 1.27 – Слухач події, який запускається кожного разу, коли змінюються дані профілю користувача

Правила Firebase дають змогу визначити роль користувача під час виконання операцій з читання та запису. Ці правила будуть діяти на рівні безпеки на сервері, перш ніж виконувати будь-яку операцію CRUD. За замовчуванням правила дозволяють користувачеві виконувати операцію читання та запису тільки після автентифікації.

Наведені правила на рисунку 1.28 дозволяють автентифікованим користувачам лише читати або записувати дані.

```
{
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
}
```

```
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

Рисунок 1.28 – Правила для читання та запису даних

Також можна використовувати ці правила для перевірки даних перед вставленням в базу даних. Наприклад, на рисунку 1.29 наведено правила, де

перевіряється, що ім'я має бути меншим, ніж 50 символів, а електронна адреса - дійсною за допомогою регулярного виразу електронної пошти.

ОС Андроїд надає кілька способів для правильного введення даних. Це є варіанти виведення помилок згідно Android Material Design. У разі вводу невалідних даних користувачу показуватиметься зрозуміла помилка, за допомогою якої він легко зможе зрозуміти що неправильно він записав у полях додатку.

Дослідження, зроблене дизайнерами Google, змінює підхід до інтерфейсу і робить його реальнішим. Інтерфейс перетворюється на живу приємну людині взаємодію.

```
{
    "rules": {
        ".read": true,
        ".write": true,
        "users": {
            "$user": {
                "name": {
                    ".validate": "newData.isString() && newData.val().length < 50"
                },
                "email": {
                    ".validate": "newData.isString() && newData.val().matches(/^[A-Z0-
                }
            }
        }
    }
}
```

Рисунок 1.29 – Правила, які перевіряють валідність даних, які вказує користувач

Далі можна приступати до роботи з андроїдом. Отже, потрібно створити та інтегрувати базу даних Realtime Database у андроїд додатку.

Перше, що потрібно зробити, це перейти на сторінку платформи Firebase та створити обліковий запис, щоб отримати доступ до своєї консолі. Після того як буде доступ до консолі, можна починати роботу, створивши свій перший проект.

Далі слід назвати пакет проекту (наприклад info.androidhive.firebase), в якому інтегруватиметься Firebase. Тут файл google-services.json буде завантажений після натискання кнопки додавання додатка. На рисунку 1.30 зображений інтерфейс для створення проекту у консолі розробника.

Щоб завантажити даний файл, потрібно:

– зайти у консоль розробника Firebase;

– натиснути на піктограму налаштувань;

– відкрити вкладку «Мої за стосунки»;

– настиснути на кнопку завантаження файлу google-services.json;

– знайти та вставити у проект;

– перевірити чи правильно вказаний шлях до файлу google-services.json у проекті.

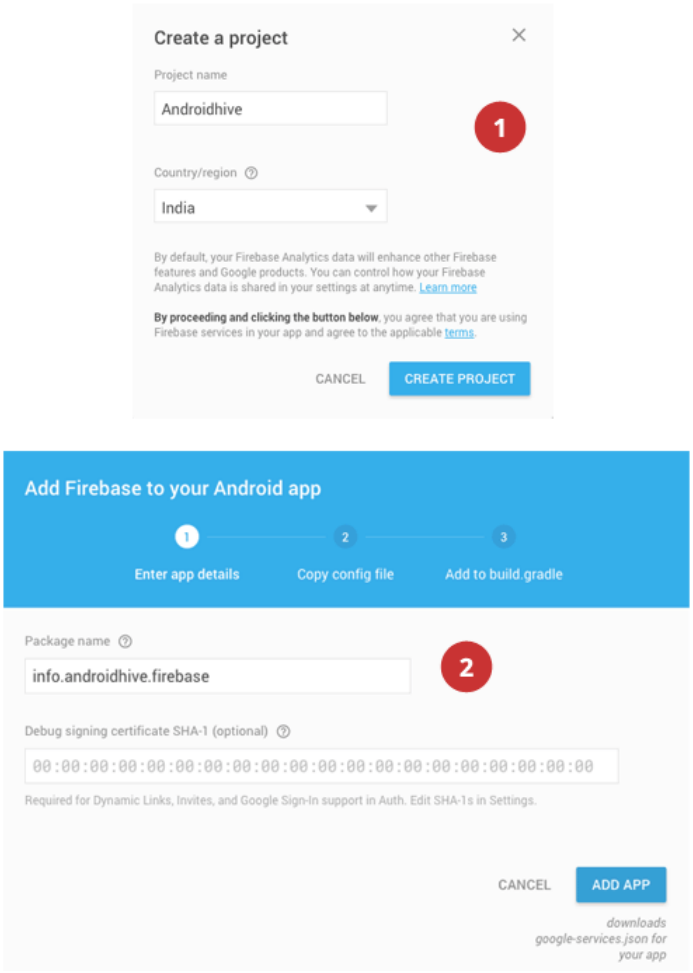


Рисунок 1.30 – Інтерфейс консолі розробника для створення проекту

Після цього потрібно створити новий проект в XCode за допомогою меню Файл ⇒ Новий проект. Під час заповнення деталей проекту слід використовувати те саме ім'я пакета, яке було надане у консолі Firebase. У даному випадку використовується та сама назва info.androidhive.firebase.

Коли проект створений, то перше, що потрібно зробити, це додати у проект файл google-services.json у папку додатка проекту. Цей крок дуже важливий, оскільки проект не буде побудовано без цього файлу.

1.3.3 Реалізація основних класів та методів.

Модель сутність-атрибут-значення – це модель даних, призначена для опису сутностей, в яких кількість атрибутів (властивостей, параметрів), що характеризують їх, потенційно величезна, але та кількість, яка реально буде використовуватися в конкретній суті, відносно мала.

Кандидатами у сутності системи є:

- офіціант;
- страва;
- категорія;
- замовлення;
- столик.

На рисунку 1.31 зображено сутність користувача.

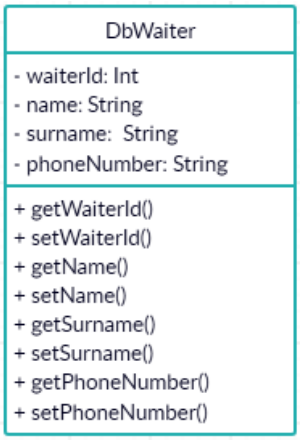


Рисунок 1.31 – Сутність офіціанта

Сутність офіціанта містить такі поля, як:

- порядковий номер офіціанта;
- ім’я офіціанта;
- прізвище офіціанта;
- номер телефону офіціанта.

Завдяки використанню SWinject можна побудувати графи залежностей компонентів у системі, а також згенерувати додаткові класи, які будуть мати допоміжну логіку для основних класів.

Залежність впорскування (DI) – це модель дизайну програмного забезпечення, яка реалізує Інверсію управління (IoC) для вирішення залежностей. Відповідно до цього, Swinject допомагає вашому додатку розбиватися на компоненти, що з’єднуються у слабкому зв’язку, які можна розробити, протестувати та підтримувати легше. Swinject працює від системи загального типу Swift та функцій першого класу для того, щоб просто та вільно визначати залежності вашого додатка.

На лістингу 3.1 наведено зв’язок із хмарною базою даних для реєстрації користувача.

Лістинг 3.1 – зв’язок із хмарною базою даних для реєстрації офіціанта:

```
override fun createUserWithEmailAndPassword(email: String, password: String): Single<Optional<AuthResult>> {
        return Flowable.create<Optional<AuthResult>>({ emitter ->
            RxTaskHandler({
                        firebaseAuth.createUserWithEmailAndPassword(email, password)
            }, emitter)
        }, BackpressureStrategy.LATEST)
                .firstOrError()
                                    .timeout(FIREBASE_OPERATION_TIMEOUT, TimeUnit.MILLISECONDS)
    }
```

На рисунку 1.32 зобажено сутність страви.

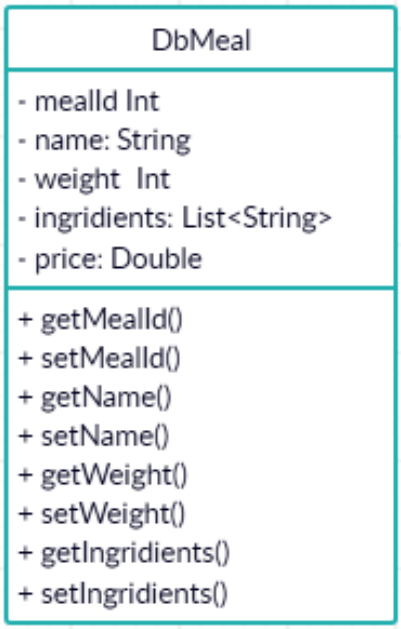


Рисунок 1.32 – Сутність страви

Отже, сутність страви містить такі поля:

- порядковий номер страви;
- назва страви;
- вага страви;
- список інгредієнтів страви;
- ціна страви.

За допомогою поля, яке містить список інгредієнтів офіціант може легко проглянути інформацію про страву та повідомити гостям у разі потреби, адже у людей бувають алергії на різні продукти.

Також, завдяки наявності списку інгрідієнтів у страві можна використати фільтри у додатку для офіціанта та знайти найбільш подібні страви по складниках та запропонувати гостям у разі необхідності.

За допомогою поля ціни можна здійснити сортування по ціні у додатку або ж знайти страву із потрібної цінової категорії, що дозволить офіціанту надавати сервіс на вищому рівні, ніж би це було без використання даного програмного продукту.

На рисунку 1.33 зображено сутність замовлення.

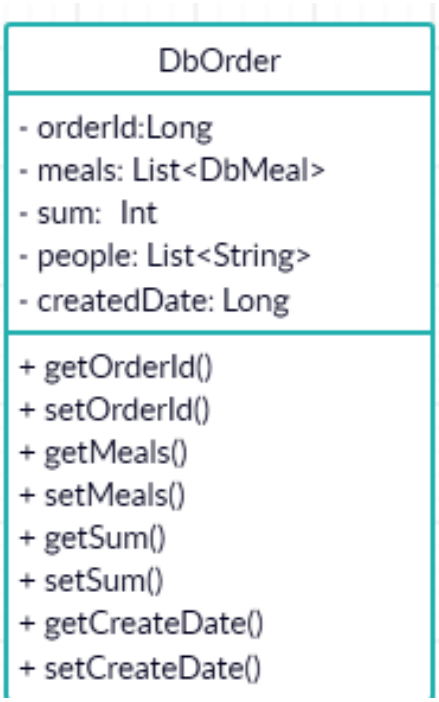


Рисунок 1.33 – Сутність замовлення

Сутність замовлення містить такі поля:

- порядковий номер замовлення;
- список страв у замовленні;
- сума замовлення;
- список гостей;
- дата створення замовлення.

Список страв у замовлення потрапляє зразу ж на кухню, а там, використовуючи порядковий номер замовлення, змінюється його статус в залежності від готовності страви і приходить у додаток офіціанту як пуш нотифікація.

Сутність столика у закладі зображено на рисунку 1.34, а також вона містить такі поля:

- порядковий номер столика;
- список замовлень;
- сума замовлень;
- статус столика;

- місткість столика (кількість людей, що може розміститись за даним столиком).

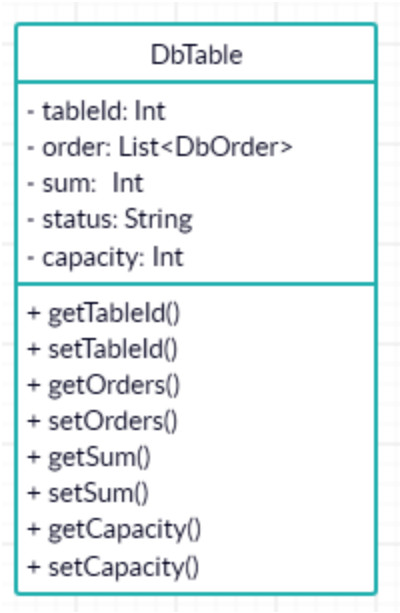


Рисунок 1.34 – Сутність столика у закладі

На лістингу 3.2 наведено приклад отримання даних про заклад із серверу та подальшу обробку за домогою rx-операторів. Далі дані з класу-репозиторію після перетворень потрапляють у клас view-model, а далі подаються на представлення.

Лістинг 3.2 – приклад отримання даних про столики із серверу:

```
override fun getTables(tableId: Int): Single<List<DbTable> {
        return Flowable.create<Optional<QuerySnapshot>>({ emitter ->

RxTaskHandler({                firestoreDb.collection(BuildConfig.TABLES)
.whereEqualTo("tableId", tableId).get()
            }, emitter)
        }, BackpressureStrategy.LATEST)
                .firstOrError()
                .map
{ it.value?.documents?.first()?.toObject(DbTable::class.java)!! }
                .onErrorResumeNext {

Single.error(ExpectedException(ExceptionStatus.DATA_GETTING_FAILED))
                }
                .timeout(FIREBASE_OPERATION_TIMEOUT,
TimeUnit.MILLISECONDS)    }
```

В даному лістингу описано отримання, а також обробку даних про столик у закладі, а також подальшу передачу цих даних по rx-ланцюжку.

Сустність категорії страв зображено на рисунку 1.35.

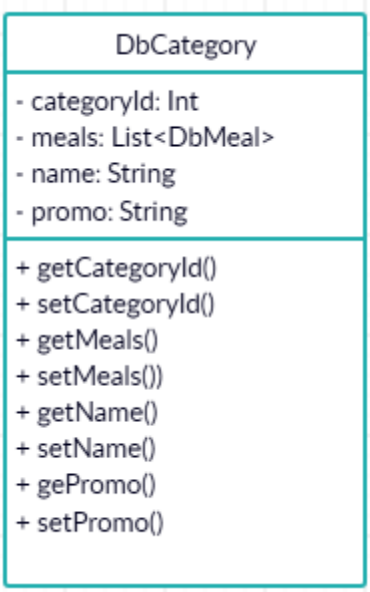


Рисунок 1.35 – Сутність категорії страв

Сутність категорії страв містить:

- порядковий номер категорії;
- назву категорії;
- список страв у категорії;
- додаткове повідомлення про категорію.

Таким чином, базуючись на основних сутностях системи, створюються інші суності, які також мають первинні ключі та унікальні ідентифікатори.

1.4 Використання програмної системи

1.4.1 Опис типових схем використання системи.

Дана програмна система призначена для спрощення та підвищення ефективності якості надання ресторанних послуг, оскільки офіціант має можливість скористатись даним додатком та підвищити комфорт надання послуг.

Обслуговування гостей стане швидким та легким при використанні даного сервісу, що дозволить зекономити час, який витрачається на похід офіціанта на кухню та викладенню персоналу кухні деталей замовлення. Достатньо лише підтвердити замовлення і воно автоматично потрапить до додатку на планшеті, що знаходиться у кухні.

Взаємодію компонентів сервісу наведено на рисунку 1.36.

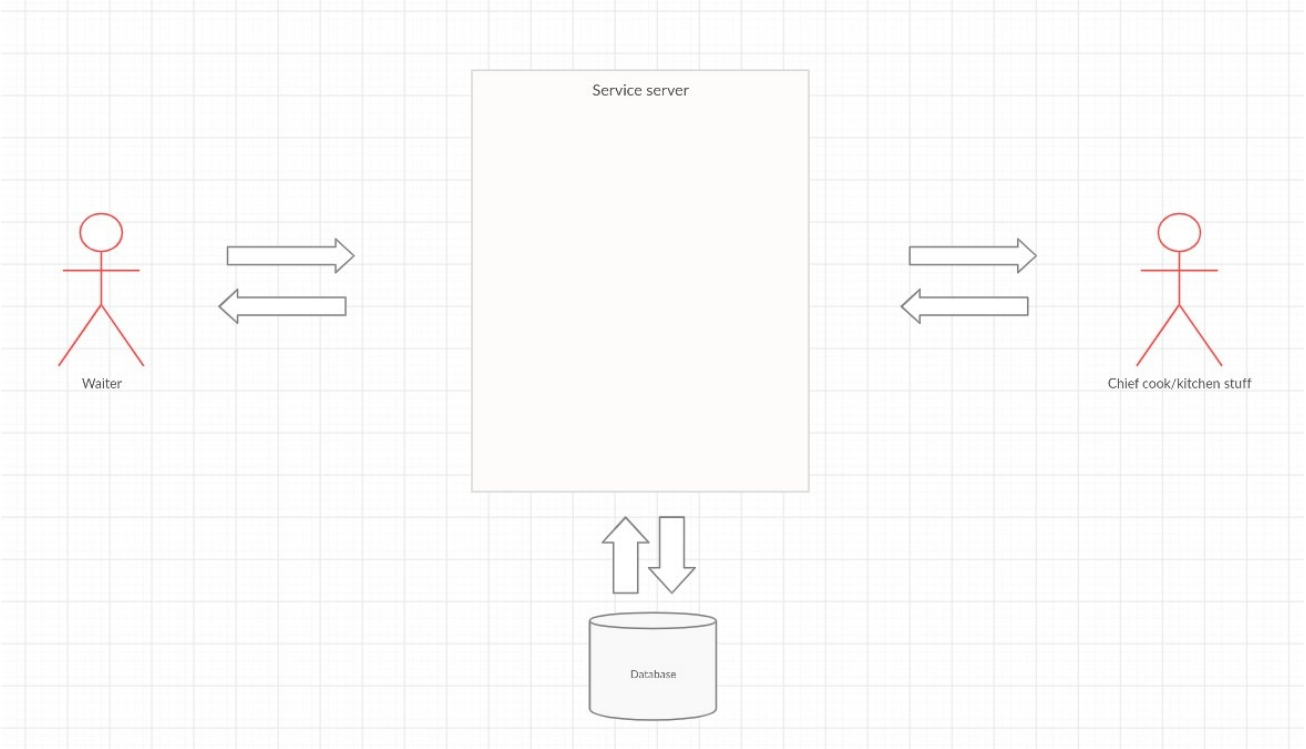


Рисунок 1.36 – Взаємодія сервісу вцілому

Сервер має своє API для взаємодії з мобільними додатками обох акторів. Саме через сервер проходять всі маніпуляції з даними та інформацією.

Для даного сервісу, сервер відіграє роль агрегатора, основна функція якого зберігати дані про страви, транзакції та офіціантів, а також опрацьовувати усі вхідні запити, для надсилання та отримання даних.

З переваг даного сервісу, можна перерахувати:

- швидкий доступ до даних;
- доступність усього функціоналу без заповненого профілю;

- наявність бази даних додатку, що забезпечує перегляд даних без підключення до мережі інтернет;
- підвищення інформативності про страви;
- підвищення рівня та якості ресторанного обслуговування;
- можливість дізнатись детальну інформацію про страву;
- простий та зрозумілий інтерфейс для офіціанта;
- легкий інтерфейс для додатку кухні;
- можливість вести статистику стосовно замовлень.

Саме завдяки даним можливостям, можна стверджувати, що даний сервіс є доцільним, а також набуватиме популярності серед закладів громадського харчування досить стрімко.

1.4.2 Верифікація програмної системи.

Верифікація програмного забезпечення — процес посвідчення, що програми та їх компоненти виконують запропоновані їм вимоги. Метою верифікації є посвідчення в тому, що програмне забезпечення відповідає висунутим вимогам. Паралельно з цим фіксуються нові дефекти, додані в процесі розробки. Процес верифікації є складовою частиною більш загального процес забезпечення домовленого рівня якості розроблюваної системи.

Верифіка́ція – доказ того, що вірогідний факт або твердження є істинним. Термін використовується в залежності від того, як обґрунтовується істина: базується вона на приведенні одного доказу або аргументу - чи вона повинна підтверджуватися можливістю багаторазово відтворювання, тобто перевірятися практикою.

Верифікація націлена на скорочення помилок. Але дуже важливо розуміти, що верифікація - це контрольований ззовні процес, що демонструє наявність у системі багів і умови їх прояву.

Верифікація дозволяє гарантувати, що програмна система реалізована без непередбачуваної функціональності, відповідає висунутим вимогам, специфікаціям і стандартам . Верифікація так само керований процес.

Процес верифікації вимог до ПЗ є невід'ємною частиною всього процесу розробки. Верифікація тісно пов'язана системи. Поняття верифікації іноді плутають з поняттями валідації, тестування і навіть налагодження, і метою цього поста є внесення ясності, що є що.

Мета процесу валідації — переконатися, що специфічні вимоги для програмного продукту виконано, і здійснюється це за допомогою:

- розробленої стратегії і критеріїв перевірки всіх робочих продуктів;
- обговорених дій з проведення валідації;
- демонстрації відповідності розроблених програмних продуктів вимогам замовника і правилам їхнього використання;
- узгодження із замовником отриманих результатів валідації продукту.

Верифікація програмного коду допомагає зробити висновок про коректність створеної програмної системи при її проектуванні і після завершення її розроблення.

Валідація дозволяє встановити здійсниміcть заданих вимог шляхом їх перегляду, інспекції і оцінки результатів проектування на процесах життєвого циклу для підтвердження того, що здійснюється коректна реалізація вимог, дотримання заданих умов і обмежень до системи. Верифікація і валідація забезпечують перевірку повноти, несуперечності і однозначності специфікації і правильності виконання функцій системи.

Верифікації і валідації піддаються:

- компоненти системи, їх інтерфейси (програмні, технічні і інформаційні) і взаємодія об'єктів (протоколи, повідомлення) у розподілених середовищах;
- описи доступу до баз даних, засоби захисту від несанкціонованого доступу до даних різних користувачів;
- документація до системи;
- тести, тестові процедури і вхідні набори даних.

Даний програмний продукт містить у собі різноманітні валідації полів з використанням регулярних виразів для таких даних, як: адреса електронної скриньки, мобільний номер телефону, пароль та інші.

1.4.3 Ілюстрація варіантів використання.

Для кращого розуміння взаємодії користувача з додатком слід навести наглядний приклад інтерфейсу додатку, а також проілюструвати гнучкість системи та адаптивність до змін. На рисунку 1.37 зображено вигляд екрану «Авторизації».

Рисунок 1.37 – Вигляд екрану авторизації

На екрані авторизації знаходиться лише 2 поля. В поле Stuff id потрібно ввести порядковий номер офіціанта або номер телефону і у поле Password ввести пароль та натиснути на “Log in”, після чого відбудеться вхід у додаток.

На рисунку 1.38 зображено вигляд профілю офіціанта у додатку.

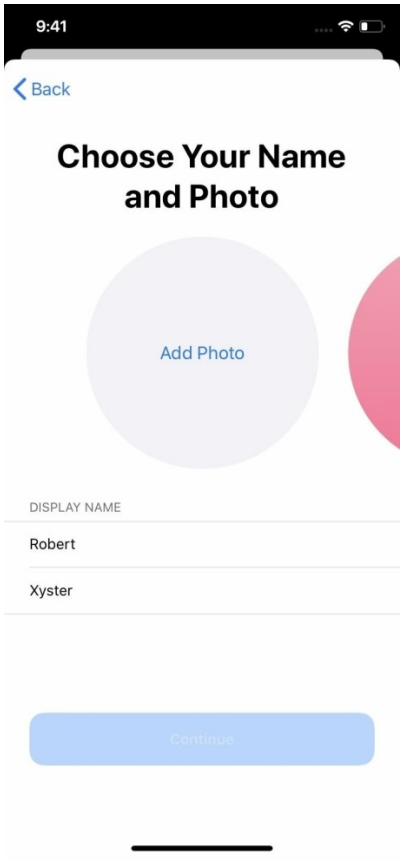


Рисунок 1.38 – Вигляд профілю офіціанта у додатку

Профіль доволі простий, оскільки у ньому не повинно бути надто багато інформації, адже в роботі надмірна інформація буду лише заважати.

Далі авторизувавшись у додатку офіціант потравпляє на список столів, де зможе переглянути статуси столів та дізнатись за який стіл можна посадити гостів. Це є досить зручним рішенням, оскільки головний функціонал додатку починається саме з цього екрану.

Вся навігація по додатку здійснюється із даного екрану, саме тому він є наче лицьовим екраном у додатку і надає швидку інформацію про статус зайнятості столів.

На рисунку 1.39 зображено екран списку столів. Слід також звернути увагу, що дана функція є однією із найнеобхідніших, що дозволить комунікувати офіціантам додатково за допомогою додатку.

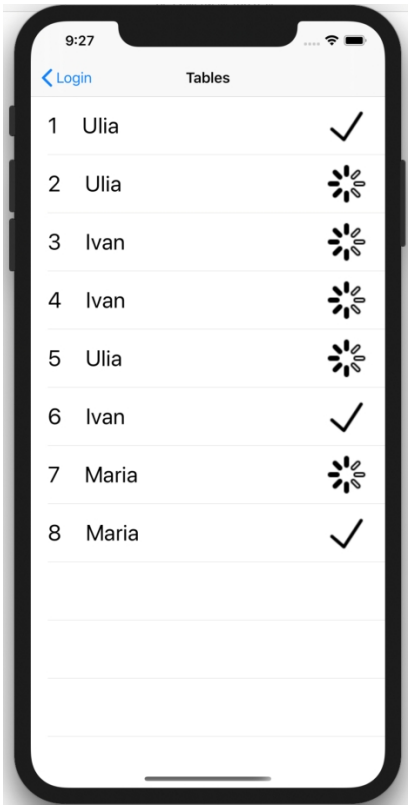


Рисунок 1.39 – Екран списку столів

На рисунку 1.40 зображено екран списку людей за столиком.

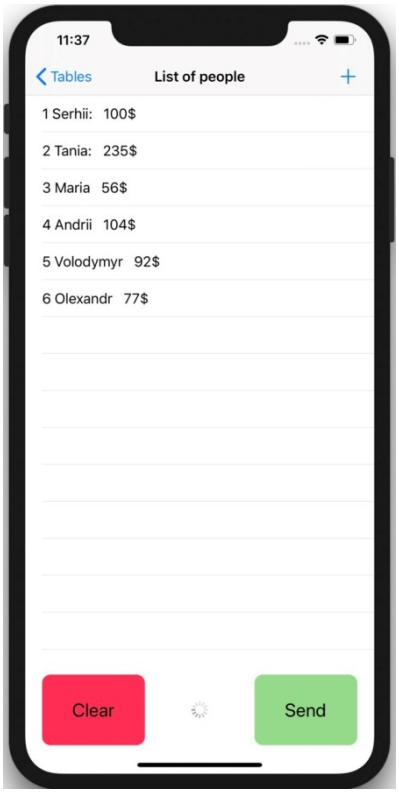


Рисунок 1.40 – Екран списку людей за столиком

На рисунку 1.41 зображено вигляд екрану списку страв.

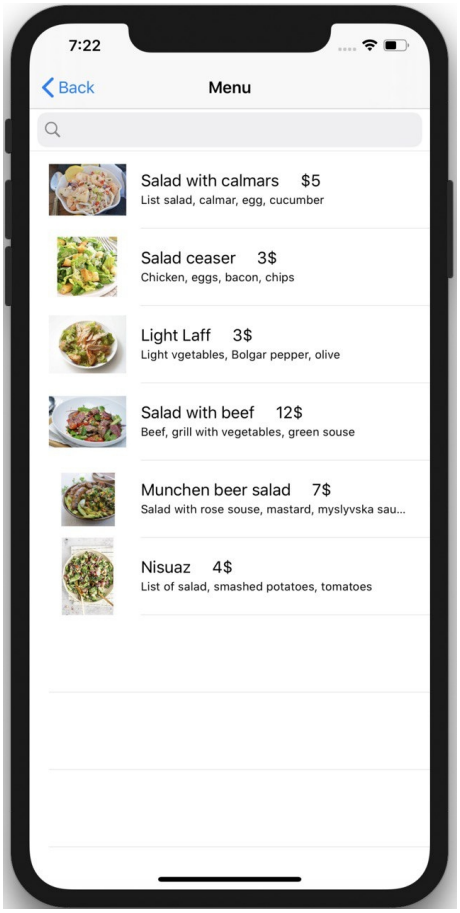


Рисунок 1.41 – Вигляд екрану списку страв

На екран списку страв можна перейти із списку категорій, або увівши в пошук назву потрібної страви. У додатку є методи для сортування страв, тому офіціант має змогу просто ввівши назву страви отримати список всіх страв, які можуть співпадати по:

- ціні;
- назві;
- вазі;
- інгредієнтах;
- категорії.

Також, із екрану зі списком страв офіціант при потребі може потрапити на екран опису страви, що зображено на рисунку 1.42.

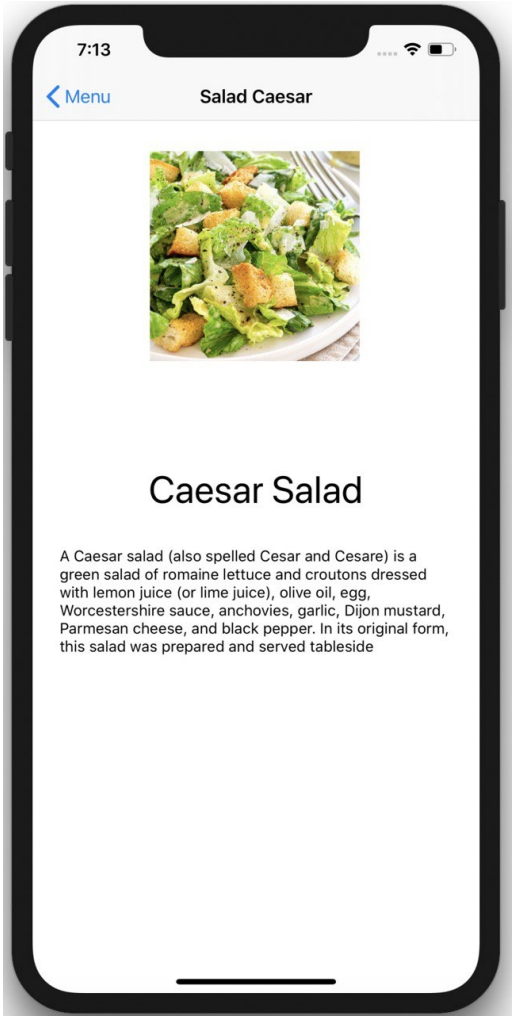


Рисунок 1.42 – екран опису страви

На екрані інформації про страву користувач може побачити її детальний опис, який включає в себе назву, основні складові, харчову цінність, інгредієнти страви.

2 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 План тестування

Ручне і автоматизоване тестування сьогодні грає істотну роль у будь-якій технологічній компанії. Будь то мобільне або веб-застосування або сайт, перевірка коду украй важлива. Правильне планування, коли і яке тестування використати, допомагає зберігати час і гроші.

Ручне тестування може займати багато часу, зате в короткостроковій перспективі заощадить в рази більше грошей. Його вартість залежить тільки від того хто тестує, а не інструментів для автоматизації.

Автоматизоване тестування – це написання коду. З його допомогою очікувані сценарії порівнюються з тим, що отримує користувач, вказуються розбіжності. Автоматизоване тестування відіграє важливу роль у важких застосуваннях з великою кількістю функцій.

Ручне тестування можна розглядати як взаємодію професійного тестувальника і софтвера з метою пошуку багів. Таким чином, під час ручного тестування можна отримувати фідбек, що неможливо при автоматизованій перевірці. Іншими словами, взаємодіючи з додатком безпосередньо, той хто тестує може порівнювати очікуваний результат з реальним і залишати рекомендації.

Обидві методики тестування мають свої переваги і недоліки, їх ми розглянемо нижче.

Тестування мобільних додатків суттєво відрізняється від тестування десктопного програмного забезпечення. Адже, монітор комп’ютера та телефон – пристрої зовсім різні, як на вигляд, розмір, внутрішнє начиння, а також по технічних характеристиках.

Зазвичай мобільні девайси, більшості середньостатистичних користувачів – це не надто дорогі пристрої, з не дуже потужною апаратною частиною, а ще трапляються непередбачувані сто відсоткові китайські гаджети.

Отже, мобільні девайси в рази слабкіші за персональний комп’ютер, тому не можуть працювати як вони. Але прогрес у цій ніші телекомунікації, сфері інформаційних додатків рухається доволі стрімко. Операційні системи мобільних телефонів, швидко старіють, крім того, межа на оновлення їх ОС.

В той же час ранжування версій ОС, від геть застарілих, до найновітніших, залишається навпаки – широким. Різноманіття екранів, їх розширень, характеристик кольорів, конфігурацій комплектуючих – у тому числі.

Отже, тестування здійснюється згідно наступних правил:

1) перевірка належного встановлення додатоку на різних мобільних пристроях, операційних системах (Android, iOS, Windows phone тощо) і різних версіях ОС;

2) перевірка належного встановлення додатоку, якщо його завантажити з Play Маркет, App Маркет, Android Маркет чи iTunes;

3) перевірка роботи додатку в комп'ютерній мережі і через бездротову локальну мережу;

4) варіанти взаємодійї з додатком в режимі польоту чи в умовах відсутності підключення до інтернету;

5) варіанти взаємодії з додатком, коли вхідний телефонний дзвінок перерве процес встановлення чи роботи вашого додатку;

6) варіанти взаємодії з додатком, коли вхідне повідомлення перерве роботу вашого додатку;

7) що станеться, якщо перезавантаження мобільного телефону перерве роботу вашого додатка;

8) як додаток зреагує на спливання поп-ап сповіщення про низький заряд батареї;

9) як додаток зреагує на поп-ап про те, що в пристрої мало пам'яті;

10) чи реагує додаток на обертання пристрою, відповідно розвертаючи екран вертикально чи горизонтально;

11) чи правильно визначається географічне місце розташування, якщо додаток інтегрується з сервісами, базованими на локації? Чи справно ця опція працює як в мережах даних, так і в бездротових мережах;

12) чи не деформуються малюнки, фотографії, піктограми? Чи задовільно завантажуються всі графічні елементи;

13) чи встановлена політика конфіденційності та чи налаштовані сповіщення, якщо додаток інтегрується з соцмережами і дозволяє користувачам використовувати їхні акаунти для реєстрації і входу;

14) яка мінімальна конфігурація необхідна для запуску програми належним чином? Чи не завадить додаток роботі інших додатків;

15) чи є які-небудь вразливі місця, такі, як шкідлива реклама чи будь-які інші кібер-загрози та чи захищений додаток від них.

а. Притримуючись цих правил можна забезпечити високу якість розроблюваного мобільного програмного забезпечення.

2.2 Розробка тестів

Написання тестів є важливою складовою забезпечення високої якості програмного коду, який буде вести себе коректно у всіх ситуаціях та правильно обробляти помилки. Мета написання модульних тестів – знаходження проґавин у архітектурі програмної системи. За допомогою модульних тестів визначається правильність обрання архітектури системи, її реалізація та надійність.

Модульне тестування – це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестованою. У процедурному програмуванні модулем вважають окрему функцію або процедуру. В об'єктно-орієнтованому програмуванні – інтерфейс, клас. Модульні тести, або unit-тести, розробляються в процесі розробки програмістами та, іноді, тестувальниками білої скриньки . Зазвичай unit-тести

застосовують для того, щоб упевнитися, що код відповідає вимогам архітектури та має очікувану поведінку.

Під час розробки програмного забезпечення методом TDD (Test-driven development), модульний тест стає частиною дизайну. Кожен тест визначає потрібні класи та методи, їх очікувану поведінку. Наведений нижче приклад на мові Java добре демонструє використання unit-тестів під час розробки методом TDD.

Тестування програмного забезпечення не може знайти всіх помилок у програмі. У більшості програм неможливо прорахувати кожен варіант виконання. Це також вірно для модульного тестування. Крім того, модульне тестування, власне, повинне тестувати тільки модулі. Так що цей вид тестування не зможе знайти інтеграційні помилки та інші: помилки архітектури, проблеми з витримкою навантажень на ПЗ. Unit-тестування має проводитись разом з іншими видами тестування програмного забезпечення.

Як і будь-який вид тестування, модульне тестування може визначити лише наявність помилок, а не їх відсутність.

На рисунку 2.1 наведено приклад написання модульного тесту для перевірки коректності отримання відповіді від сервера.

```
46      @Test
47      public void handleGitHubResponse_Success() {
48          Response response = Mockito.mock(Response.class);
49          SearchResponse searchResponse = Mockito.mock(SearchResponse.class);
50          Mockito.doReturn(true).when(response).isSuccessful();
51          Mockito.doReturn(searchResponse).when(response).body();
52          List<SearchResult> searchResults = new ArrayList<>();
53          searchResults.add(new SearchResult());
54          searchResults.add(new SearchResult());
55          searchResults.add(new SearchResult());
56          Mockito.doReturn(searchResults).when(searchResponse).getSearchResults();
57          // trigger
58          presenter.handleGitHubResponse(response);
59          // validation
60          Mockito.verify(viewContract, Mockito.times(1)).displaySearchResults(searchResults);
61      }
```

Рисунок 2.1 – Приклад написання модульного тесту для перевірки коректності отримання відповіді від сервера

На рисунку 2.2 наведено приклад написання тесту для перевірки відображення інформації для користувача на екрані.

```
@Test
public void ensureIntentDataIsDisplayed() throws Exception {
    SecondActivity activity = rule.getActivity();

    View viewById = activity.findViewById(R.id.target);

    assertThat(viewById,notNullValue());
    assertThat(viewById, instanceOf(TextView.class));
    TextView textView = (TextView) viewById;
    assertThat(textView.getText().toString(),is("Hello"));
}
```

Рисунок 2.2 – Приклад написання тесту для перевірки відображення інформації для користувача на екрані

Написання модульних тестів є важливою складовою перевірки якості програмного продукту. Отже, існує певна класифікація тестових типів об’єктів:

– манекенний об'єкт(dummy object*)* проходить навколо, але ніколи не використовується, тобто його методи ніколи не називаються. Наприклад, такий об'єкт може бути використаний для заповнення списку параметрів методу.

– фальшиві об'єкти(Fake object) мають робочі реалізації, але, як правило, спрощуються. Наприклад, вони використовують базу даних пам'яті, а не реальну базу даних.

– класи заглушки(stub class) – це часткова реалізація для інтерфейсу або класу з метою використання екземпляра цього класу підпунктів під час тестування. Заглушки зазвичай не реагують на щось за межами того, що запрограмовано для тесту. Заглушки також можуть записувати інформацію про дзвінки.

– макетний об'єкт(mock object) – це макетна реалізація для інтерфейсу або класу, в якому ви визначаєте вивід певних викликів методу.

Макети об'єкти налаштовані для виконання певної поведінки під час тесту. Вони зазвичай записують взаємодію з системою, і тести можуть підтвердити це.

Окрім модульного тестування є ще також тестування UI частини. Для цього використовується бібліотека для тестування Espresso.

Бібліотека досить молода, але працює таки стабільно і виходила вона відразу з версії 1.0, що вже говорить про те що вона вийшла не просто заради бета чи альфи для тестування. Писати з її допомогою тести дуже приємно, так як багато функціоналу вона бере на себе. Наприклад, не потрібно робити фокус на кнопці, перевіряти чи доступна вона на екрані і т.д., з Espresso достатньо звернутися до кнопки по її id або текстом чи описом і Espresso вже все зробить за вас, якщо кнопка невидима або недоступна на екрані то тест автоматом вже не пройде. Вони дійсно зробили великий акцент на UI.

Хлопці з тих підтримки намагаються максимально швидко і розгорнуто давати відповіді на ваші питання. Так ось, найголовніше питання який їм задавали «Чим Espresso краще Robotium?». Вони кажуть, що Robotium дійсно хороша, але найголовнішою причиною, по якій вони вирішили написати свій фреймворк це швидкість виконання тестів і їх не завжди стабільність. Вони запевняють що Espresso працює набагато швидше ніж інші фреймворки. Але ще одна не менш важлива особливість, це те що Espresso використовує синхронізацію з Main Thread.

Instrumentation запускається в окремому потоці, без синхронізації тестів з оновленням UI виникають проблеми зі швидкістю і можуть виникнути критичні помилки в самих тестах. Більшість розробників ігнорують цей факт і ставлять затримки в потоці (sleep), що є вже не правильно. З Espresso цього робити вже не треба буде. Robotium в свою чергу не використовує синхронізацію, що тягне за собою присипання потоків і повільне виконання тестів. Спочатку всі програми Google тестувалися за допомогою Robotium, тому що в той момент не було часу писати свій инструментал. Покрити тестами свої проекти можна до 95%!

На рисунку 2.3 наведено приклад написання тесту перевірки відправки повідомлення.

```
        @SmallTest
        public void testSendMessageUI()
        {
                Espresso.onView(ViewMatchers.withId(com.eleks.espresso.example.app.R.i
d.etEmail)).perform(ViewActions.typeText("test@test.com"));
                Espresso.onView(ViewMatchers.withId(com.eleks.espresso.example.app.R.i
d.etMessage)).perform(ViewActions.typeText("Espresso"));
                Espresso.onView(ViewMatchers.withId(com.eleks.espresso.example.app.R.i
d.btnSend)).perform(ViewActions.click());
        }
```

Рисунок 2.3 – Приклад написання тесту перевірки відправки повідомлення

Тут все досить просто. Шукається View на екрані за допомогою ViewMatchers по id, вводиться текст імітуючи введення людиною. Далі знаходиться кнопка і відбувається подія onClick по ній. Якщо заховати кнопку, то можна побачити, що тест не пройде так як Espresso просто не знайде її на екрані.

На рисунку 2.4 зображено тест для відкривання бокового меню.

```
        @SmallTest
        public void testOpenNavigationDrawer()
        {
                Espresso.onView(ViewMatchers.withId(com.eleks.espresso.example.app.R.i
d.content_frame)).perform(ViewActions.swipeRight());
                ListView lvDrawerMenu = (ListView) getActivity().findViewById(com.elek
s.espresso.example.app.R.id.lvDrawerMenu);
                Preconditions.checkNotNull(lvDrawerMenu, "lvDrawerMenu is null");
                final int count = lvDrawerMenu.getAdapter().getCount();
                Preconditions.checkPositionIndex(2, count, "No 1 index " + count + " si
ze");
                Object obj = lvDrawerMenu.getItemAtPosition(2);
                Espresso.onView(Matchers.allOf(ViewMatchers.withId(com.eleks.espresso.e
xample.app.R.id.tvItem), ViewMatchers.hasSibling(ViewMatchers.withText(obj.toString
()))))).perform(ViewActions.click());
        }
```

Рисунок 2.4 – Тест для відкривання бокового меню

Отже, тестування є дуже важливим етапом перевірки якості коду та програмного продукту вцілому. Їх написання надає підтвердження правильності вибору архітектури, а також правильність взаємодії компонентів на екрані.

3 ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНА ЧАСТИНА

3.1 Розрахунок норм часу на виконання науково-дослідної роботи

Метою розділу є встановлення економічної доцільності проведення розробки iOS додатку для керування замовленнями в ресторанних-мережах з використанням AES-256 шифрування при передачі даних в Firebase API з точки зору економічної ефективності.

Даний етап розробки програмного продукту потребує проведення комплексних і системних досліджень об’єктивних передумов підвищення економічної ефективності впровадження та використання ІКТ.

Від економічних показників залежить популярність програмної системи, кількість її продажів та поширеність серед різними сферами та користувачами.

Головною метою розділу є встановлення економічної доцільності проведення даного дослідження.

Ефективне використання часу має велике значення тому, що коефіцієнт корисної дії залежить від оптимального використання часу.

Розробку поділяють на декілька етапів, що дозволить полегшити і структуризацію виконання розробки.

Основні етапи при виконанні розробки інформаційної системи наступні:

1. Підготовка опису задачі.
2. Збір необхідної інформації для аналізу програмної системи.
3. Вибір програмного забезпечення.
4. Розробка алгоритму.
5. Розробка структури програми.
6. Розробка основних та додаткових можливостей програми.
7. Тестування програми.

Витрати часу по окремих операціях технологічного процесу відображені в таблиці 3.1.

Таблиця 3.1 – Операції технологічного процесу та час їх виконання

	Місячний оклад, грн	Денна зар. плата, грн	Трудомісткість, людино-дні		Основна заробітня плата, грн	
			Процедурний підхід	Об'єктно-орієнтова-ний підхід	Процедурний підхід	Об'єктно-орієнтова-ний підхід
Керівник проекту	5200	236	3	3	708	708
Інженер-програміст	4900	223	10	15	2230	3345
Інженер-тестувальник	3600	163	2	2	326	326
Всього			15	20	3264 грн	4379 грн

Для оцінки тривалості виконання окремих робіт використовують нормативи часу або попередній досвід.

3.2 Визначення витрат на оплату праці та відрахувань

Відповідно до Закону України "Про оплату праці" заробітна плата – це "винагорода, обчислена, як правило, у грошовому виразі, яку власник або уповноважений ним орган виплачує працівникові за виконану ним роботу".

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його праці та господарської діяльності підприємства. Заробітна плата складається з основної та додаткової оплати праці.

Основна заробітна плата нараховується на виконану роботу за тарифними ставками, відрядними розцінками чи посадовими окладами і не залежить від результатів господарської діяльності підприємства.

$ЗП_{осн1} = 3264\,грн\,; ЗП_{осн2} = 4379\,грн$

Додаткова заробітна плата – це складова заробітної плати працівників, до якої включають витрати на оплату праці, не пов'язані з виплатами за фактично відпрацьований час. Нараховують додаткову заробітну плату залежно від досягнутих і запланованих показників, умов виробництва, кваліфікації

виконавців. Джерелом додаткової оплати праці є фонд матеріального стимулювання, який створюється за рахунок прибутку.

$$ЗП_{дод} = 0{,}2 \cdot ЗП_{осн} \quad (3.1)$$

$$ЗП_{дод1} = 0{,}2 \cdot ЗП_{осн1} = 652\,грн.;$$

$$ЗП_{дод1} = 0{,}2 \cdot ЗП_{осн2} = 876\,грн.$$

Таким чином загальний фонд заробітної плати, що обчислюється за формулою:

$$ФЗП = ЗП_{осн} + ЗП_{дод}. \quad (3.2)$$

$$ФЗП_1 = 3264 + 652 = 3916\,грн.;$$

$$ФЗП_2 = 4379 + 876 = 5255\,грн.$$

Крім того, слід визначити відрахування на соціальні заходи:

- єдиний соціальний внесок – 3,6 %;
- військовий збір – 1,5 %;
- ПДФО (прибутковий податок) – 15 %.

Отже, сума відрахувань на соціальні заходи буде становити:

$$Відр_{ЄСВ1} = 0{,}036 \cdot ФЗП = 140{,}98\,грн.;$$

$$Відр_{ЄСВ2} = 0{,}036 \cdot ФЗП = 189{,}18\,грн.;$$

$$Відр_{вз1} = 0{,}015 \cdot ФЗП = 58{,}74\,грн.;$$

$$Відр_{вз2} = 0{,}015 \cdot ФЗП = 78{,}83\,грн.$$

$$Відр_{ПДФО1} = 0{,}15 \cdot ФЗП = 587{,}4\,грн.;$$

$$Відр_{ПДФО2} = 0{,}15 \cdot ФЗП = 788{,}3\,грн..$$

Нарахування на фонд оплати праці, які включають відрахування до Пенсійного фонду, фонду з тимчасової втрати працездатності, фонду з безробіття і фонду страхування від нещасних випадків на виробництві; для бюджетної організації тариф на фонд оплати праці встановлено на рівні 36,3%.

Зокрема, видання програмного забезпечення – 36,77%.

Нарахування на Фонд оплати праці (ФОП): $ФОП_{ЄСВ} = 0{,}3677 \cdot ФЗП$

$$ФОП_{ЄСВ1} = 0{,}3677 \cdot ФЗП = 1439{,}91\,грн.;$$

$$ФОП_{ЄСВ2} = 0{,}3677 \cdot ФЗП = 1932{,}26\,грн$$

Вього витрат:

$$B_{ЗП1} = ФЗП_1 + ФОП_{ЄСВ1} = 3916 + 1439{,}91 = 5355{,}91\,грн.;$$

$$B_{ЗП2} = ФЗП_2 + ФОП_{ЄСВ2} = 5255 + 1932{,}26 = 7187{,}26\,грн.;$$

3.3 Розрахунок матеріальних витрат

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни:

$$M_{ві} = q_i \cdot p_i\,, \qquad (3.3)$$

де: q_i – кількість витраченого матеріалу і–го виду;

p_i – ціна матеріалу і–го виду.

Звідси, загальні матеріальні витрати можна визначити:

$$З_{м.в.} = \sum M_{ві}\,. \qquad (3.2)$$

Таблиця 3.2 – Зведені розрахунки матеріальних витрат.

Найменування матеріальних ресурсів	Один. Виміру	Фактично витрачено матеріалів	Ціна 1-ці., грн.	Загальна сума витрат, грн
Папір для друку	листів	100	0,08	8
Чорнила для принтера	шт	1	72	72
Флеш-накопичувач	шт	1	120	120
Всього				200

Отже, загальна сума матеріальних витрат становить 200 гривень.

В багатьох випадках існує ряд додаткових витрат які пов’язані із реалізацією проекту, але в більшості випадків їхня сума не перевищує десяти відсотків від загальної собівартості реалізації проекту.

3.4 Розрахунок витрат на електроенергію

Затрати на електроенергію 1–ці обладнання визначаються за формулою:

$$З_е = W \cdot T \cdot S, \quad (3.5)$$

де W – необхідна потужність, кВт;

T – кількість годин роботи обладнання;

S – вартість кіловат-години електроенергії.

Вартість кіловат-години електроенергії слід приймати згідно існуючих на даний час тарифів. Отже, 1 кВт з ПДВ коштує 2,9 грн.

Потужність комп'ютера для створення проекту – 400 Вт, кількість годин роботи необхідних для проекту – 240 години.

$$З_е = 0{,}4 \cdot 240 \cdot 2{,}9 = 278{,}4$$

3.5 Розрахунок суми амортизаційних відрахувань

Характерною особливістю застосування основних фондів у процесі виробництва є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Амортизація – це процес перенесення вартості основних фондів на вартість новоствореної продукції з метою їх повного відновлення. Амортизаційні відрахування використовуються для повного відтворювання зношених основних фондів (на реновацію), а також для їх часткового відшкодування (на капітальний ремонт і модернізацію).

Для визначення амортизаційних відрахувань застосовуємо формулу:

$$A = \frac{C_Б \cdot N_A \cdot T_{ФАК}}{T_{год}} \quad (3.6)$$

де $C_Б$ – балансова вартість обладнання, грн;

N_A – норма амортизаційнихвідрахувань в рік, %;

$T_{год}$ – річний робочий фонд часу, год;

$T_{ФАК}$ –фактичний час роботи обладнання по написанню програми, год.

Комп'ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи річна норма амортизації дорівнює 60 % (квартальна – 15 %).

Отже, використовуючи в роботі 1 комп'ютер балансовою вартістю 8500 грн. Отже, амортизаційні відрахування будуть рівні:

$$A_1 = (8500 \cdot 0{,}6 \cdot 120)/2080 = 294{,}23 \text{ грн.}$$

$$A_2 = (8500 \cdot 0{,}6 \cdot 160)/2080 = 392{,}31 \text{ грн.}$$

3.6 Обчислення накладних витрат

Накладні витрати пов'язані з обслуговуванням виробництва, утриманням апарату управління спілкою та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб'єкта, накладні витрати можуть становити 20-60 % від суми основної та додаткової заробітної плати працівників.

$$НВ = 0{,}5 \cdot ЗП_{осн} \qquad (3.7)$$

де $Н_в$ – накладні витрати.

Отже, накладні витрати:

$Н_{в1} = 3264 \cdot 0{,}5 = 1632$ грн. $Н_{в2} = 4379 \cdot 0{,}5 = 2190$ грн.

3.7 Складання кошторису витрат та визначення собівартості

Проведемо розрахунок вартості створюваного програмного продукту. Вартість продукції включає у собі собівартість і планований прибуток.

Собівартість продукції – це сума грошових витрат підприємства (фірми) на виробництво і збут одиниці продукції, виконання робіт та надання послуг.

Повна собівартість програмного продукту дорівнює сумі усіх витрат на його виробництво:

Приймемо прибуток на рівні 30%. Для нових інноваційних продуктів,

що користуються високим попитом на ринку, ринкову вартість Вр можна втановити вищу.

Отже, вартість розробленого програмного забезпечення:

$B_{р1,2} = C_{в1,2} + 0,3 \cdot C_{в1,2} = 7760,54 + 0,3 \cdot 7760,54 = 10088,7$ грн..

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

3.8 Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Економічна ефективність ($E_р$) полягає у відношенні результату виробництва до затрачених ресурсів:

$$E_р = \frac{П}{C_В} \qquad (3.8)$$

де $П$ – прибуток;

$C_в$ – собівартість.

Плановий прибуток ($П_{пл}$) знаходимо за формулою:

$$П_{пл} = В_р – С_в \quad (3.9)$$

Розраховуємо плановий прибуток:

$П_{пл} = 10088,7 – 7760,54 = 2328,16$ грн.

Отже, формула для визначення економічної ефективності набуде вигляду:

$$E_р = \frac{П_{пл}}{С_в}. \quad (3.10)$$

Тоді,

$Е_р = 2328,16 / 7760,54 = 0,3.$

Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень ($Т_р$):

$$Т_{ок} = \frac{1}{E} \quad (3.11)$$

Термін окупності дорівнює:

$$Т_{ок} = 1 / 0,3 = 3,3 \text{ роки}$$

У нашому випадку $Т_{ок1} = Т_{ок2} = 1/0,30 = 3,33$ років, що є нормальним, оскільки допустимим вважається термін окупності до 5 років.

Даний розрахунок виконаний у розрахунку на 1 екземпляр програмного продукту без врахування його тиражування.

Загальна вартість пропонованих робіт по розробці програмного продукту становить 10088,7 грн. Оскільки ефективність для обидвох проектів відповідно до встановленого рівня прибутку становить 0,3, що є високим показником, то проводити дані роботи варто і вкладені кошти окупляться за 3,33 року. Також слід врахувати можливість не одиничного замовлення програми, відповідно її

ціна в такому випадку значно понизиться, а при продажі понад план прибуток зросте.

3.9 Визначення витрат на супровід і модернізацію програмного продукту, уточнений аналіз ефективності вкладених інвестицій

Ключові питання супроводу ПЗ – це управлінські, вимірювальні і вартісні. Відомий фахівець в області ПЗ Дж. Леман (1970 р.) запропонував розглядати супровід як еволюційну розробку програмних систем, оскільки здана в експлуатацію система не завжди цілком завершена, її треба змінювати протягом терміну експлуатації.

Внаслідок змін система стає більш складною і погано керованою. За різними оцінками фахівців витрати на підтримку і модернізацію програмного забезпечення, написаного процедурним методом, становлять більше 50% витрат на його створення, а інколи і перевищують доходи від його реалізації. Об'єктно-орієнтоване представлення програми дозволяє навіть середньому програмісту швидко і ефективно супроводжувати і модернізовувати програми довжиною до декількох десятків тисяч рядків, що значно скорочує подальші витрати на супровід і модернізацію.

Виходячи із експертних оцінок і складності програми, приймемо величину витрат на супровід і модернізацію програмного забезпечення, створеного за процедурним методом 60% від початкових витрат, а за об'єктно-орієнтованим – 20%.

Собівартість модернізації:

$C_в M_1 = 0{,}6 \cdot C_{в1} = 0{,}6 \cdot 7760{,}54 = 4656{,}32\, грн.$,

$C_в M_2 = 0{,}2 \cdot C_{в2} = 0{,}2 \cdot 7760{,}54 = 1552{,}11\, грн.$

Для споживача вартість модернізації:

$M_1 = 0{,}6 \cdot B_1 = 0{,}6 \cdot 10088{,}7 = 6053{,}22\, грн$;

$M_2 = 0{,}2 \cdot B_1 = 0{,}2 \cdot 10088{,}7 = 2017{,}74\, грн.$

Таким чином, уже після першої модернізації, загальні витрати на створення і супровід ПЗ для виробника за об’єктно-орієнтованим методом менші, ніж за процедурним, навіть якщо його собівартість є дещо дорожчою.

$ЗВ_{1(вир)} = 7760,54 + 4656,32 = 12416,86\ грн.$;

$ЗВ_{2(вир)} = 7760,54 + 1552,11 = 9312,65\ грн.$.

Як і для споживача:

$ЗВ_1 = 10088,7 + 6053,22 = 16141,92\ грн.$;

$ЗВ_2 = 10088,7 + 2017,74 = 12106,44\ грн.$.

Річна економія витрат за всіма можливими напрямами і додатковими витратами, пов'язаними з супроводом і тільки одноразовою модернізацією (у розрахунку на одиницю продукції) при об’єктно-орієнтованому методі порівняно із процедурним:

$\Delta C_{(вир)} = ЗВ_{1(вир)} - ЗВ_{2(вир)} = ¿\ 12416,86 - 9312,65 = 3104,21\ грн.$;

$\Delta C = ЗВ_1 - ЗВ_2 = 16141,92 - 12106,44 = 4035,48\ грн.$.

При постійному супроводі і модернізації програмного забезпечення різниця між загальними витратами за двома варіантами значно збільшується.

Чистий приведений дохід (ЧПД) визначається як різниця між сукупними доходами (сукупний грошовий потік) і сукупними витратами (сукупними інвестиціями) взятими за весь період життя інвестицій і дисконтова ними в кожному році на фактор часу. Дисконтування являє собою визначення вартості майбутніх грошових потоків у теперішній момент часу. Ефективним вважається той проект, який забезпечує максимум ЧПД, оскільки при цьому досягається найвища дохідність власників інвестицій.

Визначення ЧПД відбувається за формулою

$$ЧПД = \sum_{i=1}^{t} ГП_i \alpha_{TBi} - \sum_{i=1}^{t} ІК_i \alpha_{TBi};$$

де $ГП_i$– грошовий потік і-го розрахункового року;

$ІК_i$ – сума інвестицій і-го розрахункового року;

α_{TBi} – коефіцієнт дисконтування (коефіцієнт приведення інвестицій і грошового потоку до теперішньої вартості).

Грошовий потік – це фінансовий показник, що характеризує ефект інвестицій у вигляді грошових коштів, які повертаються інвестору.

Коефіцієнт дисконтування показує, яку величину грошових коштів ми отримаємо з урахуванням фактору часу та ризиків. Він дозволяє перетворити майбутню вартість у вартість на даний момент.

Для розрахунку коефіцієнта дисконтування (коефіцієнта приведення) грошових потоків за роками періоду економічного життя інвестицій використовується формула:

$$\alpha = \frac{1}{(1+i)^n};$$

де i – ставка дисконтування або норма дисконту, i = 0,2;

n – час або кількість періодів (років), протягом якого планується отримання доходу.

$$\alpha_0 = 1, \alpha_1 = \frac{1}{1+0,2} = 0,83.$$

Вважатимемо, що обидва програмних продукта одинаково забезпечують потреби і вимоги споживача, і тому придбання першої чи другої програми одинаково вплинуть на розмір його додаткових доходів на вкладений капітал. Тому приймемо цю величину за постійну, а порівняння дохідності двох проектів проведемо тільки за витратами.

$ЧПД_1' = ГП + 0,83 \cdot ГП - 10088,7 - 0,83 \cdot 6053,22 = 1,83\,ГП - 5065,53\,грн.;$

$ЧПД_1' = ГП + 0,83 \cdot ГП - 10088,7 - 0,83 \cdot 2017,74 = 1,83\,ГП - 8413,98\,грн..$

Чим менші витрати, тим більша дохідність проекту.

Саме тому важливо здійснити необхідні обрахунки для визначення економічної ефективності даного програмного продукту.

Таблиця 3.3 – Техніко–економічні показники програмного продукту

Показник	**Процедурний підхід**	**Об'єктно-орієнтований підхід**
Зарплата основна, грн	3264	4379
Зарплата додаткова, грн	652	876
Фонд заробітньої плати, грн	3916	3312
Відрахування на ФОП, грн	1439,91	1932,26
Разом на виплату плаці, грн	5355,91	7187,26
Матеріальні витрати, грн	200	200
Електроенергія, грн	278,4	278,4
Амортизація, грн	686,54	686,54
Накладні витрати, грн	1632	2190
Разом на ін.витрати, грн	2796,94	3354,94
Собівартість	7760,54	10088,7
Прибуток	2328,16	2597,58
Вартість розробленого ПЗ	7760,54	10088,7
Економічна ефективність	0,30	0,30
Термін окупності, років	3,33	3,33
Собівартість модернізації	4656,32	1552,11
Супровід і модернізація	6053,22	2017,74
Загальні витрати на розробку	12416,86	9312,65
Порівняльна економія витрат (для виробника)	-	3104,21
Загальні витрати (для споживача, на придбання програмного прод.)	16141,92	12106,44
Порівняльна економія витрат для споживача)	-	3528,19
Дохідність проекту для споживача за витратною частиною	−5065,53	−8413,98
Економія	-	3348,45

Економія витрат у випадку придбання, супроводу і одноразової модернізації програмного продукту, створеного за об’єктно-орієнтованим підходом, становить 3348,45 грн.

Отже, сучасну комп’ютеру програму доцільно виконувати по об’єктно орієнтованій парадигмі.

В даному розділі дипломної роботи магістра було розраховано основні техніко-економічні показники аналізу функціонування програмного продукту (таблиця 3.3).

Розраховане значення економічної ефективності, яке становить 0,82.

Загальна вартість пропонованих робіт по розробці програмного продукту становить 12416,86 грн. для першого варіанту та 9312,65 грн. для другого.

Оскільки ефективність для обидвох проектів відповідно до встановленого рівня прибутку становить 0,3, що є високим показником, то проводити дані роботи варто і вкладені кошти окупляться за 3,3 року, бо нормальним терміном окупності є термін, який коливається від 1 до 3 років, тоді розробка вважається доцільною і економічно вигідною.

При використанні об’єктно-орієнтовного підходу зменшується кількість працівників, які залучаються у проект, та зменшуються витрати на реалізацію проекту, але для підтримки проекту і його подальшої модернізації все ж в майбутньому потрібні набагато більші витрати.

Для функціонального підходу потрібна більша кількість працівників, часу і коштів, але на модернізацію і підтримку в загальному потрібно менше ресурсів, і у висновку програма виконана по функціональній парадигмі стає дешевшою для споживача, і приносить економію ресурсів для розробників.

Отже, програмний продукт може бути впроваджений та мати подальший розвиток, оскільки він є економічно вигідною за всіма основними техніко-економічними показниками.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці при роботі з ЕОМ

Під час написання дипломної роботи та розробки iOS додатку для керування замовленнями в ресторанних-мережах з використанням AES-256 шифрування при передачі даних в Firebase API використовувався персональний комп’ютер та мобільні пристрої, отже слід зауважити, що при роботі за комп’ютером та комп’ютерною технікою потрібно дотримуватись вимог з охорони праці задля збереження здоров’я.

Робота з комп'ютером характеризується значною розумовою напругою й нервово-емоційним навантаженням операторів, високою напруженістю зорової роботи й досить великим навантаженням на м'язи рук при роботі із клавіатурою ЕОМ. Тому раціональне планування робочого місця має забезпечувати: найкраще розміщення знарядь і предметів праці, не допускати загального дискомфорту, зменшувати втомлюваність працівника, підвищувати його продуктивність праці. Обладнання і організація робочого місця з ВДТ мають забезпечувати відповідність конструкції всіх елементів робочого місця та їх взаємного розташування ергономічним вимогам з урахуванням характеру і особливостей трудової діяльності.

Виявлення та аналіз шкідливих та небезпечних виробничих факторів слід починати з аналізу дотримання вимог, встановлених санітарними правилами і нормами ДСанПіН 3.3.2.007-98 “Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин” для виробничих приміщень та робочих місць.

Для об‘єктивної оцінки умов праці на виробництві проводиться атестація робочих місць за умовами праці й використовується “Гігієнічна класифікація праці” за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу.

Виходячи з принципів гігієнічної класифікації, умови праці діляться на 4 класи:

− 1 клас - Оптимальні умови праці – такі умови, при яких зберігається не лише здоров'я працюючих, а й створюються передумови для підтримання високого рівня працездатності;

− 2 клас - Допустимі умови праці – характеризуються такими рівнями факторів виробничого середовища і трудового процесу, які не перевищують встановлених нормативів, а можливі зміни функціонального стану організму відновлюються за час регламентованого відпочинку або до початку наступної зміни та не чинять несприятливого впливу на стан здоров'я працюючих та їх потомство в найближчому і віддаленому періодах;

− 3 клас - Шкідливі умови праці – характеризуються такими рівнями шкідливих виробничих факторів, які перевищують нормативи і здатні чинити несприятливий вплив на організм працюючого та/або його потомство;

− 4 клас - Небезпечні (екстремальні) умови праці – характеризуються такими рівнями шкідливих факторів виробничого середовища і трудового процесу, вплив яких протягом робочої зміни (або ж її частини) створює загрозу для життя, високий ризик виникнення важких форм гострих професійних уражень.

На підставі атестації робочого місця, слід охарактеризувати напруженість праці за такими напрямами:

− відповідність площі та обсягу, що зайняті робочим місцем, діючим нормам;

− відповідність обладнання вимогам нормативно - технічної

− документації, а також характеру та обсягу виконуваних робіт;

− технологічна оснащеність робочого місця (засоби захисту приладів і їх технічний стан);

− відповідність технологічного процесу, устаткування, інструментів, засобів контролю вимогам стандартів безпеки і нормам охорони праці.

Раціональне планування робочого місця має забезпечувати: найкраще розміщення знарядь і предметів праці, не допускати загального дискомфорту, зменшувати втомлюваність працівника, підвищувати його продуктивність праці.

Обладнання і організація робочого місця з ВДТ мають забезпечувати відповідність конструкції всіх елементів робочого місця та їх взаємного розташування ергономічним вимогам з урахуванням характеру і особливостей трудової діяльності (ГОСТ 12.2.032-78, ГОСТ 22.269-76, ГОСТ 21.889-76).

Заходи щодо усунення небезпеки ураження електричним струмом зводяться до правильного розміщення устаткування та електричних кабелів. Інші заходи щодо забезпечення електробезпеки, збігаються з загальними заходами пожежо- та електробезпеки.

В якості профілактичних заходів для забезпечення пожежної безпеки слід використовувати приховану електромережу, надійні розетки з пожежобезпечних матеріалів, силові мережі живлення устаткування виконувати кабелями, розрахованими на підключення в 3-5 разів більшого навантаження, включати й виключати живлення обладнання за допомогою штатних вимикачів. Треба регулярно робити чистку внутрішніх частин комп'ютерів, іншого устаткування від пилу, розташовувати комп'ютери на окремих неспалюваних столах. Для запобігання іскріння необхідно рідше встромляти і виймати штепсельні вилки з розеток.

Робочі місця мають бути розташовані на відстані не менше 1,5 м від стіни з вікнами, від інших стін на відстані 1м, між бічними поверхнями ВДТ - 1,2 м; від тильної поверхні одного ВДТ до екрана іншого - 2,5 м.

Відносно вікон робоче місце доцільно розташовувати таким чином, щоб природне світло падало на нього збоку, переважно зліва. Робочі місця слід розташовувати так, щоб уникнути попадання в очі прямого світла. Джерела освітлення рекомендується розташовувати з обох боків екрану паралельно напрямку погляду. Для уникнення світлових відблисків екрану, клавіатури в напрямку очей користувача, від світильників загального освітлення або

сонячних променів, необхідно використовувати антипроблискові сітки, спеціальні фільтри для екранів, захисні козирки, на вікнах - жалюзі.

Екран дисплея повинен бути розташованим перпендикулярно до напрямку погляду. Якщо він розташований під кутом, то стає причиною сутулості. Екран ВДТ має розташовуватися на оптимальній відстані від очей користувача, що становить 600...700 мм, але не ближче ніж за 600 мм з урахуванням розміру літерно-цифрових знаків і символів.

Конструкція робочого місця користувача ВДТ має забезпечити підтримання оптимальної робочої пози. Зручна робоча поза при роботі з комп'ютером забезпечується регулюванням висоти робочого столу, крісла та підставки для ніг. Раціональною робочою позою може вважатися таке положення, при якому ступні працівника розташовані горизонтально на підлозі або підставці для ніг, стегна зорієнтовані у горизонтальній площині, верхні частини рук - вертикальні. Кут ліктєвого суглоба коливається в межах 70-90°, зап'ястя зігнуті під кутом не більше ніж 20°, нахил голови 15-20°. Важливою є форма спинки крісла, яка повинна повторювати форму спини. Висота крісла повинна бути такою, щоб користувач не почував тиску на куприк або стегна. Крісло бажано обладнати бильцями. Його потрібно встановити так, щоб не треба було тягтися до клавіатури.

Клавіатуру слід розташовувати на поверхні столу на відстані 100...300 мм від краю, звернутого до працюючого. У конструкції клавіатури має передбачатися опорний пристрій (виготовлений із матеріалу з високим коефіцієнтом тертя, що перешкоджає мимовільному її зсуву), який дає змогу змінювати кут нахилу поверхні клавіатури у межах 5... 15°.

При оснащенні робочого місця з ВДТ лазерним принтером параметри лазерного випромінювання повинні відповідати вимогам ДСанПІН 3.3.2.007-98.

Дотримання усіх необхідних заходів з охорони праці забезпечує комфортні умови праці та відсутність шкоди здоров’ю, що сприяє підвищенню продуктивності роботи, а також меншому виснаженню при роботі за персональним комп’ютером.

4.2 Безпека в надзвичайних ситуаціях при роботі з ЕОМ

Під час дипломної роботи та розробки iOS додатку для керування замовленнями в ресторанних-мережах з використанням AES-256 шифрування при передачі даних в Firebase API при роботі за персональним комп’ютером існують такі види небезпек:

- небезпека ураження електричним струмом, внаслідок недотримання правил електробезпеки або виходу з ладу електроприладів;
- порушення роботи кістково-м’язового апарату внаслідок тривалих статичних навантажень при роботі з ПК;
- нервово-психічні перевантаження внаслідок постійного контакту з клієнтами, колегами по роботі, керівництвом при вирішенні робочих питань, які можуть носити конфліктний характер і призвести до емоційного дискомфорту, внутрішнього роздратування, емоційної нестабільності та захворювань нервової системи;
- незадовільні ергономічні характеристики робочого місця внаслідок нераціонального планування робочого місця, що може призвести до механічних травм, уражень електричним струмом та порушень кістково-м’язового апарату;
- негативний вплив недостатнього освітлення робочої зони на зір та продуктивність роботи працюючого, внаслідок несправності освітлювальних приладів або неправильного проектування освітлювальної системи;
- негативний вплив незадовільних параметрів повітряного середовища робочої зони на здоров’я працюючого, внаслідок неправильного проектування системи вентиляції або несправності її несправності;
- негативний вплив підвищеного рівня шуму на психоемоційний стан працюючого, який пов’язаний з використанням застарілої периферійної техніки, кондиціонерів, копіювальної техніки, освітлювальних приладів;
- небезпека загоряння у зв'язку із несправністю електричного обладнання, недотримання, або порушення правил протипожежної безпеки

обслуговуючим персоналом, що може призвести до пожежі. - неправильні дії персоналу у надзвичайних ситуаціях.

Приміщення, в яких перебувають розробники програмного забезпечення, відносяться до приміщень без підвищеної небезпеки ураження електричним струмом. Обладнання, що використовується в цих приміщеннях є споживачем електроенергії, що живиться від змінного струму 220 В від мережі з заземленою нейтраллю, та відноситься до електроустановок до 1000В закритого виконання. За способом захисту людини від ураження електричним струмом відповідає згідно з ГОСТ 12.2.007.0-75* (2001) «ССБТ. Изделия электротехнические. Общие требования безопасности» I (стаціонарні комп'ютери,) та II (освітлювальні прилади, кондиціонери, опалювальні пристрої, ноутбуки, сканери) класу захисту.

Згідно «Правилам улаштування електроустановок» виконані такі групи заходів з електробезпеки: Конструктивні заходи забезпечують захист від випадкового дотику до струмопровідних частин за допомогою їх ізоляції та захисних оболонок. Згідно з ГОСТ 12.1.009-76 (1999) «ССБТ. Электробезопасность. Термины и определения» у приладах II класу захисту використовується подвійна ізоляція – електрична ізоляція, що складається з робочої і додаткової ізоляції. Так як згідно з НПАОП 40.1-1.32-01 «Правила устройства электроустановок. Электрооборудование специальных установок» офісні приміщення у більшості своїй відносяться до класу пожеженебезпечної зони П-IIа (приміщення, в яких містяться тверді горючі речовини), тому передбачений ступінь захисту ізоляції обладнання IP44.

Ще одним важливим завданням безпеки в надзвичайних ситуаціях є створення заходів з пожежної безпеки. Закон України «Про пожежну безпеку» визначає загальні правові, економічні та соціальні основи забезпечення пожежної безпеки на території України, регулює відносини державних органів, юридичних і фізичних осіб у цій галузі незалежно від виду їх діяльності та форм власності.

Пожежна безпека – стан об’єкта, при якому з регламентованою ймовірністю виключається можливість виникнення та розвиток пожежі і впливу на людей її небезпечних факторів, а також забезпечується захист матеріальних цінностей. Для забезпечення пожежної безпеки в установах проводять пожежну профілактику, яка включає в себе комплекс організаційних і технічних заходів, спрямованих на забезпечення безпеки людей, на запобігання пожежі, обмеження її поширення, а також на створення умов для успішного гасіння пожежі.

Для ліквідації пожежі у початковій стадії її розвитку силами персоналу об’єктів застосовуються первинні засоби пожежогасіння. До них відносяться:

– вогнегасники;

– пожежний інвентар (покривала з негорючого теплоізоляційного полотна, ящики з піском, пожежні відра, совкові лопати, ломи, сокири тощо);

– системи автоматичного пожежогасіння.

Первинні засоби пожежогасіння, в залежності від категорії приміщень, можуть розташовуватись як окремо, так і в складі пожежних щитів. Залежно від агрегатного стану й особливостей горіння різних горючих речовин і матеріалів пожежі за ДБН В.1.1.7-2002 «Пожежна безпека об’єктів будівництва» поділяються на відповідні класи. В офісному приміщенні знаходяться дерев`яні меблі, електронна апаратура, паперові носії інформації. Клас пожежі у офісному приміщенні (згідно із ДБН В.1.1.7-2002 «Захист від пожежі. Пожежна безпека об’єктів будівництва») – пожежі твердих речовин, переважно органічного походження, горіння яких супроводжується тлінням (деревина, пластмаси, папір) – визначається як клас А.

Категорія приміщення (згідно із НАПБ Б.03.002-2007 «Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою») – визначається як категорії Д. Визначення типу та розрахунок кількості первинних засобів пожежегасіння (згідно із ДБН В.1.1.7-2002 «Захист від пожежі. Крім цього адміністративні приміщення повинні бути обладнані автоматичними пожежними

сповіщувачами, що реагують на підвищення температури, дим, полум'я. Наприклад, сповіщувачі моделей ДТЛ, ІТМ.

Також, вагомою небезпекою є ударна хвиля. Ударною хвилею називається область різкого стиску середовища, що поширюється у вигляді сферичного шару від місця вибуху з надзвуковою швидкістю. Ударні хвилі класифікуються в залежності від середовища поширення.

Для профілактики пожежі надзвичайно важлива правильна оцінка пожежонебезпеки будинку, визначення небезпечних факторів і обґрунтування способів і засобів пожежопопередження і захисту.

Одне з умов забезпечення пожежобезпеки – ліквідація можливих джерел запалення. У лабораторії джерелами запалення можуть бути:

- несправне електроустаткування, несправності в електропроводці, електричних розетках і вимикачах. Для виключення виникнення пожежі з цих причин необхідно вчасно виявляти й усувати несправності, проводити плановий огляд і вчасно усувати всі несправності;
- несправні електроприлади. Необхідні міри для виключення пожежі містять у собі своєчасний ремонт електроприладів, якісне виправлення поломок, не використання несправних електроприладів;
- обігрівання приміщення електронагрівальними приладами з відкритими нагрівальними елементами. Відкриті нагрівальні поверхні можуть спричинити пожежу, тому що в приміщенні знаходяться паперові документи і довідкова література у виді книг, посібників, а папір – легкозаймистий матеріал. З метою профілактики пожежі пропоную не використовувати відкриті обігрівальні прилади в приміщенні лабораторії;
- коротке замикання в електропроводці. З метою зменшення імовірності виникнення пожежі внаслідок короткого замикання необхідно, щоб електропроводка була схованою;
- влучення в будинок блискавки. У літній період під час грози можливе влучення блискавки внаслідок чого можливий пожежа. Щоб уникнути цього я рекомендую установити на даху будинку блискавковідвід;

– недотримання мір пожежної безпеки і паління в приміщенні також може спричинити пожежу. Для усунення загоряння в результаті паління в приміщенні лабораторії пропоную категорично заборонити паління, а дозволити тільки в строго відведеному для цього місці.

З метою запобігання пожежі пропоную проводити з інженерами, що працюють у лабораторії, протипожежний інструктаж, на якому ознайомити працівників із правилами протипожежної безпеки, а також навчити використанню первинних засобів пожежогасіння.

У випадку виникнення пожежі необхідно відключити електроживлення, викликати по телефоні пожежну команду, евакуювати людей із приміщення відповідно до плану евакуації, приведеному на рисунку 1 і приступити до ліквідації пожежі вогнегасниками. При наявності невеликого вогнища полум'я, можна скористатися підручними засобами з метою припинення доступу повітря до об'єкта загоряння.

Під час експлуатації ЕОМ забороняється здійснювати ремонт та налагодження їх на робочому місці, працювати на зіпсованій техніці, загромаджувати робочі місця матеріалами, які не використовуються для поточної роботи.

Обслуговування, ремонт та налагоджений ЕОМ, інші операції у цьому плані мають здійснюватися тільки при повному відключенні живлення. У випадках коли ремонтні та інші операції неможливо здійснити при відключеному живленні, необхідно, щоб устаткування, допоміжна апаратура та прилади були заземлені, роботу виконували два і більше працівників з використанням інструментів з ізольованими ручками, а на підлозі були діелектричні килимки.

Будь-яка пожежа починається із загорання, яке інколи може ліквідувати одна людина, якщо має відповідні навички та знає правила поведінки під час пожежі. Тому, у разі виникнення пожежі необхідно заздалегідь знати: де і які засоби пожежогасіння розміщуються та як ними користуватися.

Ні в якому разі не слід панікувати. Під час пожежі необхідно остерігатися високої температури, задимленості та загазованості, обвалу конструкцій будинків і споруд, вибухів технологічного обладнання і приладів, падіння обгорілих дерев, а також провалів. Небезпечно входити в зону задимлення.

Ремонт відеотермінала без футляра, а також усі види робіт з відкритим кінескопом повинні проводитися в захисних окулярах або масці.

При паянні, промиванні, знежиренні деталей, блоків і плат слід дотримуватись пожежної безпеки. Ці роботи виконуються у спеціально обладнаних приміщеннях.

Увійшовши в будь незнайоме приміщення вперше – ознайомтеся з планом евакуації. За нормами і вимогами він зобов'язаний висіти на стіні в приміщенні. Витратьте 5 хв на його дослідження на предмет готовності (чи є пожежні виходи тощо) Якщо плану в будівлі немає – не вкладайте свої гроші в цей заклад і не проводите там час.

Режим праці та відпочинку працівників електронно-обчислювальної техніки визначається ДСанНіП 3.3.2-007-98. Через кожні 40-50 хв. роботи необхідно робити 3-5-хвилинні перерви для відпочинку. Сумарна тривалість роботи на день не повинна перевищувати 4 год., а на тиждень - 20 год.

До роботи з профілактичного обслуговування, налагодження і ремонту ЕОМ допускаються працівники віком старше 18 років, які пройшли попереднє спеціальне навчання, мають відповідне посвідчення, не мають медичних протипоказань, пройшли інструктаж з охорони праці та пожежної безпеки.

Таким чином, дотримуючись правил безпеки в надзвичайних ситуаціях можна запобігти виникненню критичних наслідків.

ВИСНОВОК

Тема розробки мобільних застосувань під платформу iOS є досить прибутковою та доцільною і представляє широке поле для подальших досліджень в галузі розробки мобільного ПЗ. Специфіка даного сегмента полягає в тому, що розробка iOS-додатків повинна проводитися з урахуванням особливостей мобільних пристроїв: відмінностями інтерфейсу, іншим розміром екрану, сенсорним управлінням. Актуальність теми підкреслюється широким спектром можливостей для втілення ідей у вигляді мобільного додатку.

Даний програмний продукт не лише спрощує роботу для офіціантів, а покращує якісь обслуговування та надання ресторанних послуг, а також покращує комунікацію між персоналом кухні та офіціантами, що позитивно впливає на роботу та підвищення коефіцієнту корисної дії.

Головною із переваг даного сервісу перед іншими подібними є висока якість та простота використання сервісу, швидкість обслуговування клієнтів, відсутність помилок при оформленні замовлення, обробка та передача замовлення в автоматичному режимі, абсолютний контроль всіх процесів від моменту прийому замовлення до його виконання, можливість безперервно відстежувати фінансові результати роботи закладу.

У третьому розділі дипломного проекту прораховано економічну ефективність даного програмного продукту. Його цінова політика дозволить скористатись даним рішенням навіть невеличким барам та кафе чи кав’ярням, оскільки не є дорогою. Відносна дешевизна даного програмного продукту зумовлена там, що тут вбудовано лише найпотрібніший функціонал, за допомогою якого і буде відбуватись основна частина роботи у закладах громадського харчування.

Тому впровадження даного сервісу є гарним рішенням для покращення та підвищення продуктивності роботи та якості обслуговуваннях у закладах громадського харчування, що призведе до збільшення прибутку закладу, який використовуватиме даний сервіс.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація по мові програмування SWIFT [Електронний ресурс] – 2019 – Режим доступу: https://www.apple.com/ru/swift/

2. Документація по ОС iOS [Електронний ресурс] – 2019 – Режим доступу: https://developer.apple.com/documentation/

3. Документація по базі даних Cloud Firestore [Електронний ресурс] – 2018 – Режим доступу: https://firebase.google.com/docs/firestore/

4. Документація по мові програмування Kotlin [Електронний ресурс] – 2018 – Режим доступу: https://kotlinlang.org/docs/reference/

5. Документація по бібліотеці для реактивного програмування// Режим доступу: http://reactivex.io/documentation

6. Методичні вказівки до виконання магістерської роботи освітнього рівня “магістр” студентами усіх форм навчання для напряму підготовки 121 – “Інженерія програмного забезпечення” / Укладачі : Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 26 с.

7. Методичні рекомендації по виконанню розділу техніко-економічного обґрунтування дипломних робіт студентами технічних спеціальностей напряму підготовки 8.05010302 «Інженерія програмного забезпечення» освітньо-кваліфікаційного рівня «Магістр» / Укладачі: Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль: Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.

8. Документація по бібліотеці Dagger2 [Електронний ресурс] – 2019 – Режим доступу: https://google.github.io/dagger/

9. Тестування мобільних додатків [Електронний ресурс] – 2019 – Режим доступу: https://internetdevels.ua/blog/mobile-testing-checklist

10. Unit Testing Guidelines from GeoSoft [Electronic resource] – Режим доступу: http://geosoft.no/development/unittesting.html.

11. Закон України «Про оподаткування прибутку підприємств» від 28.12.1994 № 334/94-ВР. – [Електронний ресурс] 2019 – Режим доступу: http://portal.rada.gov.ua/6

12. Документація по середовищу розробки XCode [Електронний ресурс] – 2019 – Режим доступу: http://wnfx.ru/rukovodstvo-po-razrabotke-prilozheniy-v-xcode-8-2/

13. Огляд світової практики щодо впровадження інформаційних систем у сферу ресторанного бізнесу [Електронний ресурс]. 2019 – Режим доступу: http://tourlib.net/statti_ukr/borzenko.htm

14. Розробка мобільних додатків для смартфонів [Електронний ресурс] – 2019 – Режим доступу: https://ittel.com.ua/informacijni-texnologiyi/rozrobka-mobilnih-dodatkiv/

15. Інструменти для тестування мобільних додатків [Електронний ресурс] – 2019 – Режим доступу: https://www.quality-assurance-group.com/instrumenty-dlya-testuvannya-mobilnyh-dodatkiv/