МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

МУДРАК ПЕТРО РУСЛАНОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

ПРОГРАМА АСИСТЕНТ ДЛЯ МАЙСТРА З РЕМОНТУ КОМП'ЮТЕРНОЇ ТЕХНІКИ

Спеціальність 122 Комп'ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Р. М. Бабаков, професор кафедри
інформаційних технологій,
д. т. н., доцент
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

Вінниця - 2024

АНОТАЦІЯ

Мудрак П.Р. Програма асистент для майстра з ремонту комп'ютерної техніки. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2024.

У кваліфікаційній (бакалаврській) роботі досліджена та проаналізована робота із програмним середовищем Python та Django, за допомогою якого був створений додаток який оптимізовує задачі для майстрів з ремонту комп'ютерної техніки.

Ключові слова: Python, Django, ремонт, алгоритм.

76 ст., 6 дод., 40 джерел.

ABSTRACT

Mudrak P.R. Assistant programme for a computer repair technician. Speciality 122 "Computer Science", educational programme "Computer Science". Vasyl' Stus Donetsk National University, Vinnytsia, 2024.

In the qualification (bachelor's) work, the work with the Python and Django software environment was investigated and analysed, with the help of which an application was created that optimises tasks for computer repair technicians.

ЗМІСТ

ВСТУП

Актуальність. Оскільки різноманітність і складність технологічних пристроїв продовжує зростати, попит на кваліфіковані послуги з ремонту, які можуть швидко й точно діагностувати та виправляти проблеми, стає актуальнішим, ніж будь-коли. Це особливо актуально в секторах, де простої через апаратні або програмні збої призводять до значних операційних і фінансових витрат.

Крім того, сучасні інструменти для ремонту часто включають ручну діагностику та ведення записів, що може зайняти багато часу та бути вразливим до людських помилок. Інтеграція автоматизованих програмних рішень обіцяє не тільки оптимізувати ці процеси, але й підвищити точність і надійність діагностики та ремонту. Тому розробка таких інструментів має велике значення, пропонуючи суттєві покращення ефективності роботи для підприємств, які займаються обслуговуванням і ремонтом комп'ютерів.

Крім того, застосування Python і Django у створенні цих інструментів є особливо актуальним через гнучкість цих технологій, масштабованість і надійну підтримку вебоперацій і операцій з базами даних. Великі бібліотеки Python і здатність Django працювати зі складними програмами, що керуються даними, роблять їх ідеальними для розробки програми-помічника, яка може виконувати складні вимоги ремонту комп'ютера. [17,20]

Об'єктом дослідження є задачі при ремонті техніки, які можна автоматизувати або розширити за допомогою програмних засобів.

Предметом дослідження є програмне забезпечення, розроблене з використанням Python і Django для допомоги технікам з ремонту комп'ютерів шляхом автоматизації процесів діагностики та ремонту та підвищення ефективності роботи. [15]

Метою дослідження є проєктування та розробка програми-помічника для техніків з ремонту комп'ютерів, які використовують Python і Django,

зосереджуючись на підвищенні ефективності та результативності ремонтних операцій за допомогою автоматизованих процесів і зручних інтерфейсів. Згідно з встановленою метою, було визначено такі **завдання:**

1. Розглянути існуючі програмні рішення у сфері;
2. Дослідити бібліотеки Python для адміністрування комп'ютерних систем;
3. Постановити конкретні завдання, що програмне забезпечення має вирішувати;
4. Розробити алгоритмічну частину програмного забезпечення;
5. Створити надійну серверну частину за допомогою вебфрейморку Django;
6. Розробити користувацький інтерфейс;
7. Протестувати та проаналізувати ефективність розробленого продукту. [21]

Теоретичне значення дослідження. Дослідження сприяє теоретичній основі розробки програмного забезпечення та системного адміністрування шляхом вивчення інтеграції Python та Django у розробці програмних засобів для ремонту комп'ютерів. Він розширює наявні знання про автоматизацію алгоритмів у галузях технічної підтримки, пропонуючи зрозуміти, як конкретні бібліотеки Python можна використовувати для вдосконалення процесів діагностики та відновлення системи. Дослідження доповнює обсяг знань, демонструючи застосування теоретичних концепцій у реальних умовах, покращуючи розуміння можливостей програмного забезпечення в технічній діагностиці та обслуговуванні. [16]

Практична значущість дослідження. На практиці це дослідження має значні наслідки для індустрії ремонту комп'ютерів. Він забезпечує відчутне розв'язання типових проблем, з якими стикаються майстри з ремонту, завдяки розробці програмного засобу, який автоматизує багато аспектів їхньої роботи. Така автоматизація сприяє підвищенню ефективності, зменшенню кількості помилок і підвищенню задоволеності клієнтів. Програмне забезпечення, розроблене в результаті цього дослідження, може слугувати моделлю для подібних інструментів в інших технічних галузях, що потенційно призведе до ширшого впровадження та

інновацій у секторах технічних послуг. Крім того, практичне застосування цього дослідження дає підприємствам конкурентну перевагу шляхом оптимізації операцій і покращення надання послуг, що зрештою сприяє підвищенню продуктивності та прибутковості.

Структура роботи. Робота складається з трьох розділів, шести підрозділів, вступу, висновків та списку використаних джерел. Загальний обсяг роботи - 59 сторінок.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ РОЗРОБОК У СФЕРІ РЕМОНТУВАННЯ ТА НАЛАГОДЖЕННЯ КОМП'ЮТЕРНОЇ ТЕХНІКИ

1.1. Огляд існуючих програмних рішень

Ринок програмного забезпечення для ремонту комп'ютерів різноманітний і охоплює цілу низку рішень - від діагностичних утиліт до комплексних систем управління ремонтом. Ці інструменти значно відрізняються за функціональністю: від простих утиліт, які можуть виконувати базові перевірки обладнання, до просунутих систем, здатних проводити комплексну діагностику та автоматизоване розв'язання проблем. Вибір інструментів у репертуарі технічного фахівця може суттєво вплинути на ефективність та результативність його роботи.

Аналізуючи тенденції ринку, останніми роками спостерігається сплеск попиту на програмне забезпечення, яке не тільки діагностує і ремонтує, але й прогнозує потенційні несправності за допомогою предиктивної аналітики. Це зумовлено зростаючою складністю комп'ютерного обладнання та фінансовими наслідками простоїв для користувачів [13, p. 14].

Конкурентне середовище на цьому ринку також заслуговує на увагу. Домінують кілька великих гравців, які пропонують комплексні набори інструментів для управління та діагностики. Ці великі гравці часто отримують вигоду від впізнаваності бренду та розгалуженої мережі підтримки клієнтів. Однак існує також динамічна екосистема менших розробників, які впроваджують інновації, пропонуючи нішеві рішення, орієнтовані на конкретні типи обладнання або певні діагностичні підходи. Ці менші компанії часто використовують новітні технології, такі як штучний інтелект, щоб диференціювати свої пропозиції [8, p. 138].

Крім того, рівень впровадження цих інструментів може свідчити про їхню ефективність та задоволеність користувачів. Високий рівень впровадження

зазвичай свідчить про те, що інструмент є ефективним і зручним для користувача, що робить його кращим вибором серед технічного персоналу. І навпаки, інструменти з низьким рівнем впровадження можуть бути або новими на ринку, або такими, що не мають певних функціональних можливостей, або недостатньо зручними для користувача.

При проведенні порівняльного аналізу можливостей та обмежень сучасних програмних рішень у сфері ремонту та налаштування комп'ютерного обладнання важливо вивчити різні функціональні можливості цих інструментів та обмеження, з якими вони стикаються в реальних умовах застосування.

По-перше, можливості програмного забезпечення для ремонту комп'ютерів, як правило, охоплюють широкий спектр функцій, від базового пошуку інформації про систему до розширеного виявлення та усунення несправностей. Серед найпоширеніших функцій - моніторинг обладнання, діагностика системи, відновлення даних та автоматизоване усунення несправностей. Просунуті інструменти можуть також пропонувати можливості віддаленого управління, що дозволяє технічним фахівцям керувати та ремонтувати системи без фізичного доступу, що стає все більш цінним в умовах глобальної мережі [4, p. 205].

Наприклад, таке програмне забезпечення, як HWMonitor, забезпечує моніторинг апаратних компонентів, таких як температура і швидкість обертання вентиляторів, в режимі реального часу, що має вирішальне значення для профілактичного обслуговування. З іншого боку, такі інструменти, як PC-Doctor і AIDA64, пропонують більш комплексну діагностику, оцінюючи широкий спектр компонентів системи - від процесорів і графічних процесорів до зовнішніх пристроїв. Ці інструменти також добре інтегруються з іншим програмним забезпеченням, надаючи технічним фахівцям цілісний інструментарій. [27, 28]

Однак, ці рішення також мають свої обмеження. Одним з основних обмежень є сумісність; певні інструменти можуть підтримувати лише певні операційні системи або конфігурації обладнання, що може обмежити їх застосування в різних

ІТ-середовищах. Іншим обмеженням є складність використання. Хоча деякі інструменти розроблені для досвідчених користувачів, пропонуючи детальну діагностику та безліч функцій, вони можуть бути непосильними для менш досвідчених техніків. Ця складність не тільки впливає на зручність використання, але й збільшує час навчання, що потенційно уповільнює процес ремонту. [29, 30]

Крім того, надійність результатів діагностики може бути різною. Деякі інструменти можуть пропонувати швидку діагностику, яка є менш ретельною, що може призвести до упущення тонких проблем. І навпаки, більш ретельні інструменти можуть працювати повільніше, що може затримати ремонт. Баланс між швидкістю і глибиною аналізу є критичним фактором корисності діагностичного програмного забезпечення.

Можливості інтеграції цих інструментів з іншими системами можуть бути вузьким місцем. У середовищах, де технічні фахівці використовують кілька інструментів, відсутність безшовної інтеграції може перешкоджати продуктивності, вимагаючи ручного перенесення даних між системами або повторного введення інформації [5, p. 38].

Нарешті, вартість цих інструментів може бути суттєвим фактором. Висококласні рішення з широкими можливостями та підтримкою можуть бути дорогими, що може бути невиправданим для невеликих підприємств або позаштатних технічних спеціалістів. Цей фактор вартості може впливати на рівень впровадження певних просунутих інструментів, незважаючи на їхні чудові функціональні можливості.

Впровадження програмних засобів для ремонту комп'ютерного обладнання та відгуки їхніх користувачів дають важливу інформацію про ефективність, зручність використання та загальну задоволеність цими продуктами. Ця оцінка допомагає зрозуміти, наскільки добре інструменти відповідають потребам технічного персоналу, і може слугувати орієнтиром для подальшого вдосконалення та розвитку індустрії програмного забезпечення.

На рівень впровадження програмних інструментів у секторі ремонту комп'ютерів впливає кілька факторів, включаючи складність програмного забезпечення, його вартість, сумісність з існуючими системами та сприйняту цінність, яку воно приносить в інструментарій технічного фахівця. Високий рівень впровадження часто спостерігається у програмних продуктах, які поєднують розширену функціональність зі зручним інтерфейсом, мають конкурентоспроможну ціну, а також пропонують надійну підтримку і регулярні оновлення [12, p. 169].

Відгуки користувачів є ще одним важливим компонентом оцінки впливу та корисності програмних інструментів. Вони надають безпосередню інформацію від кінцевих користувачів, які щодня взаємодіють з програмним забезпеченням. Позитивні відгуки часто висвітлюють такі аспекти, як точність діагностики, простота використання та підвищення ефективності ремонтних завдань завдяки програмному забезпеченню. Такі відгуки не тільки підвищують цінність інструменту, але й приваблюють нових користувачів завдяки позитивним відгукам з вуст в уста і відгукам в галузі.

Однак, відгуки можуть також виявити сфери незадоволення або запропонувати потенційні покращення. Серед найпоширеніших проблем, на які посилаються користувачі, - проблеми сумісності з певним обладнанням або операційними системами, складний процес навчання та неналежна підтримка клієнтів. Крім того, ефективність рекомендацій щодо усунення несправностей та ремонту, які надає програмне забезпечення, є частою темою відгуків, оскільки користувачі значною мірою залежать від цих функцій для швидкого та ефективного розв'язання проблем.

Наприклад, користувачі можуть похвалити інструмент за його комплексну діагностику, але критикувати його за надмірну ресурсомісткість, що може сповільнювати роботу системи під час перевірок. З іншого боку, вони можуть

оцінити дизайн інтерфейсу користувача інструменту, але зауважити, що його процеси ремонту не завжди відповідають останнім випускам обладнання.

Підсумовуючи, порівняльний аналіз поточних програмних рішень для ремонту комп’ютерів підкреслює різноманітність функцій, адаптованих до різних потреб і середовищ, але також підкреслює обмеження, пов’язані з сумісністю, зручністю використання, діагностичною надійністю, інтеграцією та вартістю.

1.2. Огляд бібліотек Python для адміністрування комп’ютерних систем

У сфері розробки програмного забезпечення для адміністрування та ремонту комп'ютерних систем Python пропонує кілька бібліотек, які допомагають створювати ефективні інструменти. Серед них виділяються бібліотеки `psutil`, `os` та `subprocess` завдяки їх широкій корисності у створенні додатків, які тісно взаємодіють з апаратним забезпеченням системи та процесами операційної системи.

psutil (process and system utilities) - крос-платформна бібліотека, що надає інтерфейс для отримання інформації про запущені процеси та використання системи (процесор, пам'ять, диски, мережа, датчики) на мові Python. Вона широко використовується у програмах моніторингу, де видимість використання ресурсів у реальному часі має вирішальне значення для діагностики вузьких місць та аномалій системи. Для спеціалістів з ремонту комп'ютерів `psutil` може бути особливо цінним в автоматизованих перевірках працездатності та оцінці продуктивності, дозволяючи їм швидко оцінювати стан системи та реагувати на такі проблеми, як витік пам'яті, високе навантаження на процесор або ненормальна мережева активність.

Бібліотека os слугує засобом взаємодії з операційною системою і використовується для виконання залежних від операційної системи функцій, таких

як операції з файлами та каталогами, виконання команд командного інтерпретатора, а також читання і встановлення системної інформації, наприклад, змінних оточення. Ця бібліотека є ключовою у створенні програмних засобів, яким потрібно маніпулювати файловою системою, змінювати конфігурацію системи або безпосередньо взаємодіяти з ОС на низькому рівні. У сценаріях відновлення бібліотека `os` дозволяє писати сценарії пакетного відновлення, автоматизувати резервне копіювання і навіть відновлювати налаштування системи до заздалегідь відомого гарного стану [12, p. 167].

subprocess - ще один потужний модуль Python, який використовується для створення нових процесів, підключення до їхніх каналів вводу/виводу/помилок та отримання кодів повернення. Цей модуль особливо корисний, коли інструменту відновлення потрібно запускати зовнішні скрипти або команди, керувати іншими програмами або автоматизувати відновлення, які включають складні послідовності завдань, які в іншому випадку виконуються вручну в командному рядку [3]. Наприклад, підпроцес можна використовувати для запуску діагностичних утиліт, які не є частиною програми на Python, збираючи їхні результати та інтегруючи їх у ширшу систему діагностики, що надається інструментом відновлення.

Кожна з цих бібліотек має свої сильні сторони:

- `psutil` є безцінним інструментом для моніторингу та діагностики системи в реальному часі.

- `os` надає широкі можливості для керування файлами та системою.

- `subprocess` чудово справляється з міжпроцесною взаємодією та оркеструванням зовнішніх інструментів.

Однак, вони також мають обмеження. Бібліотека `psutil`, попри свою потужність, може не охоплювати всі деталі апаратного забезпечення, особливо на менш поширених платформах або новому обладнанні. Бібліотека `os`, будучи сильно залежною від базової операційної системи, може призвести до проблем з крос-платформною сумісністю, вимагаючи додаткової обробки у коді для

підтримки функціональності у різних середовищах. `subprocess`, хоч і є потужним засобом керування процесами, потребує обережного поводження з ним, щоб уникнути проблем безпеки, таких як атаки з використанням командного інтерфейсу, а також для витонченого керування складною обробкою помилок при збоях у виконанні зовнішніх команд.

Інтеграція бібліотек Python, таких як `psutil`, `os` та `subprocess`, у програмні інструменти для адміністрування та ремонту комп'ютерних систем має важливе значення для максимізації ефективності та забезпечення безперебійної роботи у різноманітних обчислювальних середовищах. Ефективні методи інтеграції можуть значно підвищити функціональність і продуктивність цих інструментів, роблячи їх більш надійними і пристосованими до різних завдань і сценаріїв. [31, 32, 33]

Одним з основних методів ефективної інтеграції є застосування модульної архітектури при розробці програмного забезпечення. Цей підхід передбачає організацію програмного забезпечення в окремі модулі, кожен з яких відповідає за окремий аспект функціональності системи. Наприклад, один модуль може використовувати `psutil` для моніторингу системи, інший - `subprocess` для запуску зовнішніх команд, а третій - `os` для взаємодії з файловою системою. Така модульність полегшує обслуговування та оновлення кожного компонента, не впливаючи на інші, сприяє повторному використанню коду у різних частинах програми, а також спрощує налагодження та тестування [1, p. 218].

Включення керованого подіями програмування є ще однією ефективною технікою інтеграції, особливо коли йдеться про моніторинг системи у реальному часі або коли дії мають бути ініційовані певними системними подіями. Бібліотеки типу `psutil` можна використовувати для моніторингу параметрів системи, а при виявленні певних порогових значень або аномалій можна генерувати події для запуску певних дій, наприклад, ініціювання діагностики або ремонту за допомогою підпроцесу `subprocess` або реєстрації інформації за допомогою `os`. Цей метод

гарантує, що системи швидко і ефективно реагують на зміни або проблеми, зменшуючи час простою і втрати ресурсів.

Використання асинхронного програмування має важливе значення при інтеграції цих бібліотек для ефективної обробки операцій, пов'язаних з вводом/виводом та процесором. Асинхронні методи запобігають блокуванню системних ресурсів під час довготривалих операцій, таких як обширне сканування системи або резервне копіювання даних, які часто використовуються в інструментах системного адміністрування. Завдяки застосуванню асинхронних викликів, зокрема, при використанні `subprocess` для виконання зовнішніх процесів або `os` для файлових операцій, програмне забезпечення може підтримувати чуйність і продовжувати виконувати інші завдання, очікуючи на завершення цих операцій [10, p. 1794].

Надійні механізми обробки помилок і журналювання є життєво важливими для ефективної інтеграції цих бібліотек. Забезпечення того, що всі потенційні винятки та помилки будуть коректно оброблені й зареєстровані, може значно підвищити надійність і зручність використання програмного забезпечення. Це передбачає використання блоків try-except для викликів таких бібліотек, як підпроцес, які можуть завершитися невдачею через помилки зовнішніх команд, і реєстрацію цих інцидентів за допомогою можливостей бібліотеки `os` для запису до файлів журналів. Це не лише полегшує налагодження, але й допомагає підтримувати стабільність системи, запобігаючи збоям через необроблені помилки.

Нарешті, забезпечення крос-платформної сумісності є критично важливим, особливо при інтеграції бібліотек, які тісно взаємодіють з операційною системою. Такі методи, як використання незалежних від платформи функцій бібліотеки `os` для маніпулювання шляхами та файлами, а також забезпечення адаптивності команд підпроцесів до різних середовищ оболонки, є дуже важливими. Це також може включати умовний імпорт і використання змінних середовища для точного

налаштування роботи кожної бібліотеки у різних системах, що підвищує універсальність інструменту і розширює можливості його застосування [11, p. 56].

Отже, бібліотеки `psutil`, `os` і `subprocess` є базовими для розробки програмних засобів для ремонту комп'ютерів і системного адміністрування. Вони забезпечують широкий спектр системних взаємодій від низькорівневого управління процесами до високорівневого моніторингу, відіграючи вирішальну роль в автоматизації та ефективності сучасних інструментів для ремонту комп'ютерів. Їх подальший розвиток та інтеграція в ремонтні фреймворки визначатимуть можливості та надійність майбутніх програмних рішень у цій галузі.

Завдяки цим методам інтеграція `psutil`, `os` і `subprocess` в інструменти для ремонту комп'ютерів і системного адміністрування не тільки максимізує ефективність, але і підвищує масштабованість, ремонтопридатність і надійність програмних рішень, що в кінцевому підсумку призводить до кращої продуктивності та задоволеності користувачів.

1.3. Постановка завдання бакалаврської роботи

Під час ремонту комп'ютерів технічні фахівці часто стикаються з різноманітними проблемами, які можуть перешкоджати їхній ефективності та результативності. Ці виклики можуть включати діагностику періодичних збоїв обладнання, управління різноманітними програмними середовищами, а також відстеження численних ремонтних робіт і пов'язаної з ними документації. Кожне з цих завдань являє собою проблему, яку можна вирішити за допомогою спеціалізованих програмних засобів.

Однією з поширених проблем при ремонті комп’ютерної техніки є значна кількість часу, що витрачається на діагностику системних збоїв через відсутність

комплексних діагностичних інструментів, які можуть ефективно вирішувати широкий спектр проблем з апаратним та програмним забезпеченням [9, p. 195].

Іншою виявленою проблемою є неавтоматизований процес відстеження ремонтних робіт, який схильний до помилок і неефективності. Технічний персонал часто використовує базові електронні таблиці або навіть паперові системи для управління ремонтами, що може призвести до неправильного управління інформацією про клієнтів, інвентаризації запчастин та виставлення рахунків [4, p. 206].

Розробка програми допомоги технічним спеціалістам з ремонту комп'ютерів спрямована на розв'язання конкретних проблем у цій галузі та підвищення загальної ефективності та результативності ремонтних процесів. Цілі та очікувані результати такої програми покликані забезпечити як безпосередні, так і довгострокові переваги для технічного персоналу, їхніх роботодавців та клієнтів.

Таким чином, програма має бути спрямована на автоматизацію багатьох рутинних завдань, пов'язаних з ремонтом комп'ютерів, таких як введення даних, управління запасами та оновлення статусу про хід ремонту. Автоматизація покликана звільнити час технічного персоналу, дозволяючи йому більше зосередитися на складних ремонтних завданнях, які потребують експертної уваги.

Точна і повна документація є невіддільною складовою в ремонтних роботах, особливо для відстеження історії ремонтів і підтримки якості обслуговування. Помічник покликаний сприяти покращенню практики документування шляхом автоматичного створення та зберігання записів про ремонт, взаємодію з клієнтами та результати діагностики [8, p. 148].

Надаючи інструменти для кращого відстеження статусу ремонту та полегшення комунікації з клієнтами, програма має на меті покращити обслуговування клієнтів. Такі функції, як автоматичні оновлення, що надсилаються клієнтам про статус ремонту або виявлені проблеми, можуть допомогти у підтримці прозорості та побудові довіри.

Завдяки вдосконаленій діагностиці та автоматизованим процесам, очікується, що програма скоротить загальний час, необхідний для завершення ремонту, тим самим збільшивши пропускну здатність ремонтних робіт і зменшивши час простою клієнтів.

Автоматизація та розширені можливості діагностики повинні призвести до зменшення кількості людських помилок у процесі ремонту, таких як помилковий діагноз або неправильне введення даних, тим самим підвищуючи надійність послуг, що пропонуються.

Завдяки покращенню комунікації та обслуговування клієнтів, а також наданню швидших і точніших ремонтних послуг, очікується, що програма допоможе ремонтним службам утримати існуючих клієнтів і залучити нових завдяки позитивним відгукам з вуст в уста.

Програма зробить роботу технічного персоналу менш виснажливою та більш корисною завдяки зменшенню рутинного адміністративного навантаження та підвищенню їхньої спроможності розв'язувати складні проблеми, що може призвести до більшої задоволеності роботою та продуктивності.

Масштабованість програмного забезпечення дозволить ремонтним майстерням легко справлятися зі зростаючим обсягом ремонтних робіт без відповідного збільшення кількості помилок або затримок, легко адаптуючись до зростання бізнесу.

Досягнувши цих цілей і реалізувавши ці результати, асистентська програма має на меті трансформувати ландшафт ремонту комп'ютерів, зробивши його більш ефективним, надійним і дружнім до клієнтів. Успіх програми в кінцевому підсумку буде вимірюватися рівнем її впровадження серед технічних спеціалістів та її впливом на їхню повсякденну діяльність і рівень задоволеності клієнтів.

РОЗДІЛ 2
РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Розробка алгоритмічної частини

В основі розробки логічної складової програмного забезпечення в цьому контексті лежить розробка складних алгоритмів управління робочим процесом. Вони необхідні для організації послідовності завдань і дій, що беруть участь у процесі ремонту. Це, своєю чергою, допомагає гарантувати, що кожен крок реєструється, відстежується і виконується відповідно до заздалегідь визначених термінів і стандартів.

Наприклад, коли створюється новий запит на ремонт, система автоматично генерує робочий процес, який включає такі завдання, як призначення технічного персоналу, замовлення запчастин, планування завдань і комунікація з клієнтом. Кожне завдання має позначку часу і відстежується на предмет прогресу.

Програмне забезпечення надійно підтримує виконання протоколів ремонту і технічного обслуговування, забезпечуючи виконання всіх процедурних кроків за допомогою інтерфейсу контрольного списку або списку завдань. Для кожного типу ремонту система може генерувати індивідуальний набір завдань на основі типу пристрою та історії ремонтів. Це гарантує, що всі необхідні процедури будуть виконані та задокументовані для забезпечення якості та подальшого використання.

Механізми обробки помилок і усунення несправностей в програмному забезпеченні зосереджені на управлінні невідповідностями та проблемами, які виникають в процесі управління ремонтом. Це охоплює обробку помилок, таких як неправильне введення даних або конфлікти в плануванні. Система розроблена так, щоб спонукати до коригувальних дій, пропонуючи рішення або альтернативи, коли виникають проблеми.

Невіддільною частиною алгоритмічної розробки є також звітність та документація. Система призначена для автоматичного створення детальних звітів і

записів для кожного ремонту, включаючи інформацію про клієнта, історію обслуговування, статус виконання завдання та деталі виставлення рахунків. Ця вичерпна документація має вирішальне значення для підтримки високих стандартів обслуговування та полегшення будь-яких подальших дій або гарантійних претензій.

Вебзастосунок базується на чотирьох ключових операціях, які є важливими для керування та відстеження процесів ремонту: створення запиту, оновлення запиту, перегляд усіх незавершених завдань і перевірка стану конкретний запит. Кожна з цих функцій допомагає для підтримки безперебійного робочого процесу в умовах ремонту та забезпечення високого рівня обслуговування клієнтів і ефективності роботи.

Алгоритм створення запиту розроблений так, щоб отримати всю необхідну інформацію про ремонтну роботу з моменту її початку. Це починається зі зручної для користувача форми, куди технічні спеціалісти вводять такі деталі, як ім’я клієнта, контактна інформація, тип пристрою, бренд, модель, серійний номер і опис проблеми.

Кроки алгоритму:

1. Введення даних за допомогою структурованої форми.

2. Перевірка наявності та формату всіх обов’язкових полів (наприклад, перевірка правильності електронної пошти, дійсності серійних номерів).

3. Створення запису після перевірки даних для реєстрації нового ремонту в базі даних.

4. Сповіщення клієнта та призначення технічного спеціаліста в залежності від наявності чи досвіду.

5. Планування завдань за ініціюванням серії стандартних завдань для вказаного типу ремонту в системі управління завданнями.

Оновлення запиту передбачає зміну існуючих записів у міру появи нової інформації або зміни обставин ремонту. Ця функція має ефективно обробляти зміни даних, щоб процес ремонту був безперебійним і точним.

Кроки алгоритму:

1. Доступ до запису за допомогою унікальних ідентифікаторів, таких як ідентифікатор квитанції або ім'я клієнта.

2. Зміна даних через інтерфейси для оновлення деталей, як-от стан ремонту, використовувані запчастини, додаткові витрати або оновлені повідомлення клієнтів.

3. Перевірка оновлених даних на відсутність помилок.

4. Оновлення запису в базі даних з забезпеченням цілісності даних.

5. Реєстрація аудиту з фіксацією змін для подальшого використання та підзвітності, включаючи інформацію про те, хто і коли вніс зміни.

Перегляд усіх незавершених завдань є важливим для керування робочим процесом і забезпечення своєчасного завершення ремонту. Ця функція об’єднує всі завдання з різних ремонтних робіт і організовує їх у спосіб, який легко контролювати.

Кроки алгоритму:

1. Вилучити з бази даних усі завдання, позначені як невиконані.

2. Розташувати завдання за пріоритетністю або терміном виконання, виділивши термінові завдання.

3. Представити завдання на зручній інформаційній панелі, яка дозволяє технікам швидко бачити їх робоче навантаження та оновлювати статус.

4. Надати параметри для фільтрації або сортування завдань за типом, датою, технікою або терміновістю.

Перевірка статусу конкретного запиту дозволяє як персоналу, так і клієнтам відстежувати хід ремонту. Ця функція має забезпечувати оновлення в реальному часі в доступному форматі.

Кроки алгоритму:

1. Використовувати функцію пошуку, щоб знайти запит за допомогою таких параметрів, як ідентифікатор квитанції або ім’я клієнта.

2. Отримати останні відомості щодо запиту з бази даних.

3. Аналізувати поточний прогрес та статус запиту, обчислюючи будь-які затримки або проблеми.

4. Показувати докладну інформацію про статус у чіткій та стислій формі, включаючи очікувані дати завершення та будь-які незавершені дії.

Ці алгоритмічні функції складають основу можливостей програмного забезпечення для ефективного керування процесами ремонту. Автоматизуючи ці аспекти, програмне забезпечення не тільки підвищує ефективність роботи, але й покращує якість обслуговування, забезпечуючи точне та своєчасне виконання всіх процесів.

2.2. Розробка програмної частини

У Django представлення (views) - це або функції Python, або класи, які отримують вебзапит і повертають вебвідповідь. Вони виконують цілий ряд завдань від автентифікації користувача до обробки даних форм і рендерингу шаблонів з контекстними даними. Ключові аспекти включають:

Файл views (додаток А) включає такі представлення, як login_view і logout_view, які керують автентифікацією користувачів.

login_view:

```
def login_view(request):
    if request.method == "POST":

        # Attempt to sign user in
        username = request.POST["username"]
        password = request.POST["password"]
        user = authenticate(request, username=username, password=password)

        # Check if authentication successful
        if user is not None:
            login(request, user)
            return HttpResponseRedirect(reverse("index"))
        else:
            return render(request, "auctions/login.html", {
                "message": "Invalid username and/or password."
            })
    else:
        return render(request, "auctions/login.html")


def logout_view(request):
    logout(request)
    return HttpResponseRedirect(reverse("dashboard"))
```

Цей фрагмент коду містить два представлення (view functions): `login_view` та `logout_view`.

1. `login_view(request)`:

- Якщо HTTP-запит є POST (тобто, користувач надсилає дані форми), функція намагається автентифікувати (увійти) користувача.

- Вона отримує значення `username` та `password` з даних POST-запиту.

- За допомогою функції `authenticate` з Django, вона перевіряє, чи існує користувач з такою комбінацією імені користувача та паролю.

- Якщо користувач існує, функція `login` з Django автентифікує користувача та переадресовує його на головну сторінку (`reverse("index")`).

- Якщо автентифікація не вдалася, функція відображає сторінку входу (`auctions/login.html`) з повідомленням про помилку.

- Якщо запит не є POST, функція просто відображає сторінку входу.

2. `logout_view(request)`:

- Ця функція викликає метод `logout` з Django, який розлогінює поточного автентифікованого користувача з веб-сайту.

У підсумку, цей код реалізує функціональність входу та виходу для веб-додатку на Django. `login_view` обробляє процес входу користувача, перевіряючи його облікові дані та автентифікуючи його у разі успіху. `logout_view` просто розлогінює користувача з веб-сайту.

Ці представлення використовують вбудовані в Django методи автентифікації, входу та виходу для керування сеансами користувачів, що є важливим для підтримки безпеки, особливо при обробці конфіденційних даних клієнтів та ремонтних робіт. [37, 39]

Такі функції, як enquiryForm:

```
    def enquiryForm(request):
        Id = Enquiry.objects.latest('receiptID')
        #Id = Enquiry.objects.earliest('receiptID')
        Id.receiptID +=1

        currentDate = datetime.datetime.now().date()
        currentDate = str(currentDate)

        return      render(request,'repair/enquiryForm.html',      {'receiptID':
Id.receiptID, 'currentDate' : currentDate})
```

Ця функція відповідає за відображення форми запиту на ремонт і передає деякі дані в контекст шаблону.

Ось що робить ця функція:

1. `Id = Enquiry.objects.latest('receiptID')`: Ця лінія отримує останній об'єкт з моделі `Enquiry`, впорядкований за полем `receiptID` у спадному порядку. Ця змінна `Id` буде містити об'єкт з найбільшим значенням `receiptID`.

2. `Id.receiptID += 1`: Значення поля `receiptID` цього останнього об'єкту збільшується на одиницю. Це дозволяє згенерувати нове значення `receiptID` для наступного запиту.

3. `currentDate = datetime.datetime.now().date()`: Отримується поточна дата за допомогою модуля `datetime` Python.

4. `currentDate = str(currentDate)`: Дата перетворюється на рядок для подальшого використання в шаблоні.

5. `return render(request,'repair/enquiryForm.html', {'receiptID': Id.receiptID, 'currentDate' : currentDate})`: Функція `render` Django використовується для відображення шаблону `repair/enquiryForm.html`. Два значення передаються в контекст шаблону: `receiptID` (нове значення для наступного запиту) і `currentDate` (поточна дата).

Отже, ця функція представлення генерує нове значення `receiptID` для наступного запиту на ремонт, отримує поточну дату і передає ці значення в контекст шаблону `enquiryForm.html`.

enquiryReceipt:

```
    def enquiryReceipt(request):
        if(request.method == 'POST'):
            enquiryDate             =             request.POST.get('enquiryDate',
datetime.datetime.now().date())
            customerName = request.POST.get('customerName','')
            contactNo = request.POST.get('contactNo','')
            email = request.POST.get('email','')
            address = request.POST.get('address','')

            deviceType = request.POST.get('deviceType','')
            brand = request.POST.get('brand','')
            deviceModel = request.POST.get('deviceModel','')
            serialNo = request.POST.get('serialNo','')
```

```
            deviceCondition = request.POST.get('deviceCondition','')

            problemCategory = request.POST.get('problemCategory','')
            problem = request.POST.get('problem','')
            problemDescription = request.POST.get('problemDescription','')

            estimatedCost = request.POST.get('estimatedCost',0)
            advance = request.POST.get('advance',0)

            if advance == '':
                advance = 0

            enq = Enquiry(enquiryDate=enquiryDate, customerName=customerName, contactNo = contactNo, email=email, address=address, deviceType = deviceType, brand=brand, deviceModel= deviceModel, serialNo = serialNo, deviceCondition=deviceCondition, problemCategory = problemCategory, problem = problem, problemDescription = problemDescription, estimatedCost = estimatedCost, advance = advance)
            enq.save()

            enquiryObject = Enquiry.objects.latest('receiptID')
            return render(request,'repair/enquiryReceipt.html', {'receiptID' : enquiryObject.receiptID})
        else:
            message = {
                'errorMessage':'Invalid Request',
            }
            return render(request,'repair/errorPage.html', message)
```

Цей код опрацьовує POST-запит від форми запиту на ремонт пристрою. Основна функціональність полягає в збереженні даних з форми в базі даних і генерації ідентифікаційного номера квитанції для подальшого відстеження.

Якщо запит є POST, код витягує всі поля форми з даних запиту за допомогою методу request.POST.get(). Він визначає значення за замовчуванням, якщо поле залишилося порожнім (наприклад, поточна дата або порожній рядок).

Далі створюється об'єкт моделі Enquiry з отриманими даними, і він зберігається в базі даних за допомогою методу save().

Після збереження об'єкта, він отримує останній збережений об'єкт Enquiry з найбільшим значенням receiptID за допомогою Enquiry.objects.latest('receiptID'). Це значення receiptID передається в контекст шаблону enquiryReceipt.html для відображення як квитанційний номер для подальшого відстеження.

Якщо запит не є POST, код відображає сторінку помилки errorPage.html з повідомленням про неприпустимий запит.

updateForm:

```
    def updateForm(request):
        if(request.method=='POST'):
            if(request.POST.get('requestType','')=='receiptID'):
                receiptID = request.POST.get('receiptID',0)
                customerDetails = Enquiry.objects.filter(receiptID = receiptID)

                try:
                    enquiryInstance = Enquiry.objects.get(receiptID = receiptID)
                except ObjectDoesNotExist:
                    message = {
                    'errorMessage':'No Record Exists for Receipt ID/No : ' +
receiptID,
                    }
                    return render(request,'repair/errorPage.html', message)
```

Тут первіряється, чи надійшов POST-запит, і якщо тип запиту (`requestType`) є `'receiptID'`, то витягує значення `receiptID` з POST-даних. Потім він фільтрує об'єкти `Enquiry` за цим `receiptID` та намагається отримати конкретний об'єкт `Enquiry` з таким `receiptID`. Якщо такий об'єкт не знайдений, код повертає сторінку помилки з відповідним повідомленням.

```
            elif(request.POST.get('requestType','')=='serialNo'):
                serialNo = request.POST.get('serialNo','')

                customerDetails = Enquiry.objects.filter(serialNo = serialNo)

                try:
                    enquiryInstance = Enquiry.objects.get(serialNo = serialNo)
                except ObjectDoesNotExist:
                    message = {
```

```
                    'errorMessage':'No Record Exists for Device Serial No. : ' +
serialNo,
                    }
                    return render(request,'repair/errorPage.html', message)
            else:
                message = {
                    'errorMessage':'Undefined Request Type',
                }
                return render(request,'repair/errorPage.html', message)
```

Далі перевіряється, чи тип запиту (`requestType`) є `'serialNo'`. Якщо так, він отримує значення `serialNo` з POST-даних, фільтрує об'єкти `Enquiry` за цим `serialNo` та намагається отримати конкретний об'єкт `Enquiry` з таким `serialNo`. Якщо такого об'єкта не знайдено, повертається сторінка помилки з відповідним повідомленням. Якщо тип запиту не є `'receiptID'` або `'serialNo'`, також повертається сторінка помилки з повідомленням про невизначений тип запиту.

```
            # Add logic Here
            # Check Status
            if customerDetails[0].status == 'EN':
                data = {
                    'customerDetails' : customerDetails,
                }
                return render(request, 'repair/updateForm.html', data)
            elif customerDetails[0].status == 'CH':
                testDetails = TestDetail.objects.filter(Enquiry = enquiryInstance)
                data = {
                    'customerDetails' : customerDetails,
                    'testDetails' : testDetails,
                }
                return render(request, 'repair/updateForm.html', data)
            elif customerDetails[0].status == 'RE' or customerDetails[0].status ==
'CO':
                testDetails = TestDetail.objects.filter(Enquiry = enquiryInstance)
                repairDetails      =      RepairDetail.objects.filter(Enquiry      =
enquiryInstance)
                componentsUsed = str(repairDetails[0].componentsUsed)

                componentsUsedBackup = str(componentsUsed)
```

```
                componentsUsed = componentsUsed.split('~')
                componentsUsed = [i.split(':') for i in componentsUsed]
                del(componentsUsed[0])
                componentsUsedPriceTotal    =    str(sum([int(i[1])    for    i    in
componentsUsed]))

                balance        =        int(repairDetails[0].totalPrice)        -
int(customerDetails[0].advance)
```

Далі перевіряється статус замовлення на ремонт (`customerDetails[0].status`) і виконує відповідні дії залежно від його значення:

1. Якщо статус "EN" (запит створений), він передає дані замовника (`customerDetails`) шаблону `updateForm.html`.

2. Якщо статус "CH" (тестування завершене), він отримує деталі тестування (`testDetails`) і передає їх разом із даними замовника шаблону.

3. Якщо статус "RE" (ремонт завершений) або "CO" (замовлення завершене), він отримує деталі тестування та ремонту, розраховує загальну вартість використаних комплектуючих (`componentsUsedPriceTotal`), визначає залишок до оплати (`balance`) і передає всі ці дані шаблону.

Таким чином, він адаптує контекст даних, які передаються шаблону `updateForm.html`, залежно від поточного стану замовлення на ремонт.

```
                data = {
                    'customerDetails' : customerDetails,
                    'testDetails' : testDetails,
                    'repairDetails' : repairDetails,
                    'componentsUsed' : componentsUsed,
                    'componentsUsedPriceTotal' : componentsUsedPriceTotal,
                    'balance' : balance,
                }
                return render(request, 'repair/updateForm.html', data)
```

```
            elif customerDetails[0].status == 'RJ':
                message = {
```

```
                    'errorMessage':'This Record is Rejected',
                }
                return render(request,'repair/errorPage.html', message)
            else:
                message = {
                    'errorMessage':'Your Database contains some invalid values for
Enquiry.Status',
                }
                return render(request,'repair/errorPage.html', message)
        else:
            message = {
                'errorMessage':'Invalid Request',
            }
            return render(request,'repair/errorPage.html', message)
```

В кінці функції опрацьовуються додаткові випадки та ситуації помилок:

1. Якщо статус замовлення "RJ" (відхилено), він повертає сторінку помилки з повідомленням "This Record is Rejected".

2. Якщо статус замовлення не належить до жодного з очікуваних значень, він повертає сторінку помилки з повідомленням "Your Database contains some invalid values for Enquiry.Status".

3. Якщо вхідний запит не є POST-запитом, він повертає сторінку помилки з повідомленням "Invalid Request".

Таким чином, цей фрагмент забезпечує належну обробку помилкових ситуацій та неприпустимих запитів, повертаючи відповідні сторінки помилок з описовими повідомленнями.

finalReceipt:

```
    def finalReceipt(request):
        if(request.method=='GET'):
            receiptID = request.GET.get('receiptID','')

            # Changing Status
            enquiryObj = Enquiry.objects.get(receiptID=receiptID)
            enquiryObj.status = 'CO'
            enquiryObj.save()
```

```
            data = {
                'receiptID' : receiptID,
            }
            return render(request, 'repair/finalReceipt.html', data)
        else:
            message = {
                'errorMessage':'Invalid Request',
            }
            return render(request,'repair/errorPage.html', message)
```

Ця функція finalReceipt обробляє GET-запити. Вона отримує receiptID з параметрів запиту і знаходить відповідний об'єкт Enquiry в базі даних. Потім змінює статус цього об'єкта на 'CO' (завершено) і зберігає зміни. Після цього функція відображає шаблон finalReceipt.html, передаючи receiptID в контекст. Якщо запит не є GET, вона повертає сторінку помилки. Таким чином, ця функція використовується для завершення процесу ремонту та генерації фінальної квитанції.

Всі ці функції поєднані між собою і функціонують так, щоб зберігати та отримувати дані з бази даних. Це передбачає створення нових записів і оновлення існуючих на основі даних, введених користувачем, що має вирішальне значення для відстеження ремонтних робіт і їхніх статусів. [25]

Багато цих функцій включають механізми управління та реагування на невірні запити або проблеми з введенням даних, гарантуючи, що користувачі отримують відповідний зворотній зв'язок і вказівки для покращення зручності використання та запобігання пошкодженню даних.

Диспетчер URL-адрес у Django спрямовує вхідні запити до відповідного подання на основі URL-адреси, що є життєво важливим для функціональності вебдодатку. Кожна функціональність, така як оновлення запиту, перевірка статусу або керування запитами, має власний шаблон URL-адреси. Така логічна організація допомагає підтримувати додаток.

Використання іменованих URL-адрес (додаток Б) у Django дозволяє вам відокремити URL-адреси від кодової бази, що полегшує модифікацію URL-адрес у майбутньому, не впливаючи на код, який їх генерує.

```
    from django.urls import path
    from .views import (
        dashboard, enquiryForm, enquiryReceipt, enquiryReceiptFrameContent,
        updateRequest, updateForm, updateTestDetails, updateRepairDetails,
        finalReceipt, finalReceiptFrameContent, pendingRepairRequest,
        pendingRepairList,   checkStatusRequest,   checkStatusShow,   login_view,
logout_view,
    )

    urlpatterns = [
```

" - порожній шлях, пов'язаний з функцією dashboard, яка, є домашньою сторінкою.

```
        path('', dashboard, name='dashboard'),
```

'login' - шлях для сторінки входу, пов'язаний з функцією login_view.

```
        path("login", login_view, name="login"),
```

'logout' - шлях для виходу з системи, пов'язаний з функцією logout_view.

```
        path("logout", logout_view, name="logout"),
```

'enquiryForm/' - шлях для форми запиту, пов'язаний з функцією enquiryForm.

```
        path('enquiryForm/', enquiryForm, name='enquiryForm'),
```

'enquiryReceipt/' - шлях для квитанції про запит, пов'язаний з функцією enquiryReceipt.

```
        path('enquiryReceipt/', enquiryReceipt, name='enquiryReceipt'),
```

'enquiryReceiptFrameContent/' - шлях для контенту фрейму квитанції запиту, пов'язаний з функцією enquiryReceiptFrameContent.

```
        path('enquiryReceiptFrameContent/',            enquiryReceiptFrameContent,
name='enquiryReceiptFrameContent'),
```

'updateRequest/' - шлях для оновлення запиту, пов'язаний з функцією updateRequest.

```
        path('updateRequest/', updateRequest, name='updateRequest'),
```

'updateForm/' - шлях для форми оновлення, пов'язаний з функцією updateForm.

```
        path('updateForm/', updateForm, name='updateForm'),
```

'updateTestDetails/' - шлях для оновлення деталей тестування, пов'язаний з функцією updateTestDetails.

```
        path('updateTestDetails/', updateTestDetails, name='updateTestDetails'),
```

'updateRepairDetails/' - шлях для оновлення деталей ремонту, пов'язаний з функцією updateRepairDetails.

```
        path('updateRepairDetails/', updateRepairDetails,
name='updateRepairDetails'),
```

'finalReceipt/' - шлях для фінальної квитанції, пов'язаний з функцією finalReceipt.

```
        path('finalReceipt/', finalReceipt, name='finalReceipt'),
```

'finalReceiptFrameContent/' - шлях для контенту фрейму фінальної квитанції, пов'язаний з функцією finalReceiptFrameContent.

```
        path('finalReceiptFrameContent/', finalReceiptFrameContent,
name='finalReceiptFrameContent'),
```

'pendingRepairRequest/' - шлях для запиту на ремонт, що очікує, пов'язаний з функцією pendingRepairRequest.

```
        path('pendingRepairRequest/', pendingRepairRequest,
name='pendingRepairRequest'),
```

'pendingRepairList/' - шлях для списку запитів на ремонт, що очікують, пов'язаний з функцією pendingRepairList.

```
        path('pendingRepairList/', pendingRepairList, name='pendingRepairList'),
```

'checkStatusRequest/' - шлях для запиту на перевірку статусу, пов'язаний з функцією checkStatusRequest.

```
        path('checkStatusRequest/', checkStatusRequest,
name='checkStatusRequest'),
```

'checkStatusShow/' - шлях для відображення статусу, пов'язаний з функцією checkStatusShow.

```
        path('checkStatusShow/', checkStatusShow, name='checkStatusShow'),
    ]
```

Ці маршрути, використовуються для різних функцій веб-додатку, таких як обробка запитів, відстеження статусу, оновлення інформації тощо.

Взаємодія між бекендом і фронтендом Django управляється за допомогою рендерингу HTML-шаблонів. Представлення завантажують дані, обробляють їх за необхідності та передають шаблонам, які потім генерують остаточний HTML для відправки клієнту. Таке розділення завдань полегшує керування та налагодження додатка.[26]

Управління базами даних у додатку Django для управління завданнями з ремонту комп'ютерів побудовано на основі декількох моделей, які охоплюють різні аспекти процесу ремонту, від початкового запиту до завершення ремонту. [18,19] Використання фреймворку Django ORM (Object-Relational Mapping) полегшує взаємодію з базою даних в ефективний, безпечний та масштабований спосіб.

Огляд моделей (додаток В):

1. Модель Enquiry:

```
class Enquiry(models.Model):
    PROBLEM_CATEGORY_CHOICES = (
        ('HW','Hardware'),
        ('SW','Software'),
    )

    STATUS_CHOICES = (
        ('EN','Enquired'),
        ('CH','Checked'),        # This is amazing
        ('RE','Repaired'),       # Changed from Update Form when Components are
added and Repair Charge is added
        ('CO','Completed'),      # Changed to when Final Receipt is Generated
        ('RJ','Rejected'),
    )

    receiptID = models.AutoField(primary_key=True)
    enquiryDate = models.DateField()
    customerName = models.CharField(max_length = 50)
    contactNo = models.CharField(max_length = 20)
    email = models.CharField(max_length = 50, blank=True)
    address = models.TextField(blank=True)
    deviceType = models.CharField(max_length = 50)
    brand = models.CharField(max_length = 50, blank=True)
```

```
    deviceModel = models.CharField(max_length = 50)
    serialNo = models.CharField(max_length = 50, blank=True)
    problemCategory = models.CharField(max_length = 3, choices =
PROBLEM_CATEGORY_CHOICES)
    problem = models.CharField(max_length = 50)
    problemDescription = models.TextField(blank=True)
    deviceCondition = models.TextField(blank=True)
    estimatedCost = models.IntegerField()
    advance = models.IntegerField(default=0)
    status = models.CharField(max_length=3, choices = STATUS_CHOICES,
default='EN')
```

Ця модель є центральним елементом програми, що містить всі основні деталі кожного запиту на ремонт. Поля включають інформацію про клієнта (ім'я, контактний номер, електронну пошту, адресу), дані про пристрій (тип, марка, модель, серійний номер) і специфіку проблеми (категорія, опис, орієнтовна вартість).

Вона також містить прапорці статусів, які відображають хід ремонту, такі як «Запитано», «Перевірено», «Відремонтовано», «Завершено» і «Відхилено». Ці статуси допомагають відстежувати робочий процес кожного ремонтного завдання.

Модель `Enquiry` використовує поля моделі Django, такі як `CharField` для тексту, `DateField` для дат і `IntegerField` для числових значень, забезпечуючи належне форматування і зберігання даних. [14]

Зв'язки управляються за допомогою зовнішніх ключів, а вибірки використовуються для обмеження значень полів певними наперед визначеними опціями, що підвищує цілісність даних.

2. Модель TestDetail:

```
class TestDetail(models.Model):
    Enquiry = models.ForeignKey(Enquiry, on_delete=models.CASCADE)
    actualProblem = models.CharField(max_length = 50)
    actualProblemDescription = models.TextField(blank=True)

    def __str__(self):
```

```
            return str(self.Enquiry.receiptID) + " : " + str(self.Enquiry.status)
+ " : " + self.Enquiry.customerName + " : " + self.actualProblem
```

Ця модель пов'язана з моделлю `Enquiry` за допомогою відношення ForeignKey, гарантуючи, що кожен запис про деталі тесту пов'язаний з конкретним запитом.

Вона зберігає деталі про діагностику, виконану на пристрої, включно з фактично виявленими проблемами та їхніми детальними описами.

Структура підтримує робочий процес, де після початкового запиту та оцінки можуть бути виконані детальні тести для точного визначення проблем з пристроєм.

3. Модель RepairDetail:

```
class RepairDetail(models.Model):
    Enquiry = models.ForeignKey(Enquiry, on_delete=models.CASCADE)
    componentsUsed = models.TextField(blank=True)
    repairCharge = models.IntegerField(blank=True)
    otherCharge = models.IntegerField(blank=True)
    totalPrice = models.IntegerField(blank=True)

    def __str__(self):
        return str(self.Enquiry.receiptID) + " : " + str(self.Enquiry.status)
+ " : " + self.Enquiry.customerName + " : " + str(self.totalPrice)
```

Ця модель, також пов'язана з моделлю `Enquiry` через ForeignKey, фіксує специфіку процесу ремонту для кожного запиту.

Вона фіксує використані компоненти, витрати на ремонт, інші витрати та загальну вартість ремонту. Цей детальний фінансовий облік має вирішальне значення для виставлення рахунків і управління запасами.

Використання `TextField` для компонентів дозволяє гнучко вводити дані, які можуть значно відрізнятися від одного ремонту до іншого.

Структура бази даних передбачає добре нормалізований дизайн, де надлишкові дані зведені до мінімуму, зв'язки логічно визначені, а цілісність даних підтримується.

Ключові поля, особливо ті, що використовуються в пошукових запитах, такі як `receiptID` або `serialNo`, проіндексовані для покращення продуктивності запитів та прискорення пошукових операцій.

Внутрішня база даних програми ефективно налаштована для обробки різних аспектів процесу управління ремонтами, від відстеження запитів на ремонт до управління детальними ремонтними діями та витратами. Таке налаштування не тільки ефективно підтримує поточні функції, але й забезпечує основу для майбутніх удосконалень, таких як інтеграція більш детальних функцій управління взаємовідносинами з клієнтами або розширення аналітичних можливостей для отримання бізнес-інсайтів.

У розробці додатка на основі Django для керування завданнями з ремонту комп'ютерів, конфігураційний файл «settings.py» (додаток Г) відіграє ключову роль у визначенні налаштувань і поведінки різних компонентів програми. Цей файл містить налаштування та декларації, які впливають на поведінку Django у таких аспектах, як конфігурація бази даних, налаштування безпеки, встановлені програми, проміжне програмне забезпечення, шаблони та інше. [35, 36]

Параметр `BASE_DIR`:

```
import os


# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
```

Цей параметр визначає базовий каталог проєкту. Він використовується для побудови шляхів до різних інших компонентів і файлів, таких як база даних і статичні файли. Цей параметр є фундаментальним, оскільки він допомагає визначити місцезнаходження ресурсів відносно структури проєкту.

Ключ `SECRET_KEY`:

```
SECRET_KEY = ')5az3&z10_k&ijl20f=ansf_k8b(_s9aw4@))^&fp7)3a9jysu'
```

Цей ключ впроваджує додатковий рівень безпеки, оскільки він використовується для криптографічного підпису. Його слід тримати в таємниці у виробничих середовищах.

`INSTALLED_APPS`:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.admindocs',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'repair.apps.RepairConfig'
]
```

Цей параметр включає додатки за замовчуванням Django та власний додаток «repair». Ця конфігурація контролює, які програми є активними у проєкті Django, дозволяючи вам додавати або видаляти функціональні можливості за потреби.

Налаштування `MIDDLEWARE`:

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

Цей параметр визначає ряд рівнів проміжного програмного забезпечення, через які проходять відповіді на запити. Проміжне програмне забезпечення може використовуватися для керування сеансами, покращення безпеки тощо, впливаючи на те, як обробляються запити та відповіді.

Параметр `DATABASES`:

```
DATABASES = {
    'default': {
```

```
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Цей параметр визначає, що SQLite використовується як серверна база даних. [22, 24]

Параметр `TEMPLATES`:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Цей параметр визначає спосіб завантаження та відображення шаблонів у Django. За замовчуванням використовується рушій шаблонів Django з налаштуваннями, які визначають, де Django має шукати файли шаблонів і як він має їх завантажувати.

Параметри `STATIC_URL` і `STATIC_ROOT`:

```
STATIC_URL = '/static/'


#STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

Ці параметри визначають базову URL-адресу і каталог, звідки Django завантажує статичні файли, такі як CSS, JavaScript і зображення. [23]

Параметри `LANGUAGE_CODE`, `TIME_ZONE`, `USE_I18N`, `USE_L10N` та `USE_TZ`:

```
LANGUAGE_CODE = 'en-us'
```

```
TIME_ZONE = 'Europe/Kiev'

USE_I18N = True

USE_L10N = True

USE_TZ = True
```

Ці параметри налаштовуються для керування тим, як додаток обробляє локалізацію, часові пояси та інтернаціоналізацію, гарантуючи, що додаток може належним чином задовольнити потреби різних локалей та часових параметрів.

Параметр `X_FRAME_OPTIONS`:

```
X_FRAME_OPTIONS = 'SAMEORIGIN'
```

Цей параметр має значення `SAMEORIGIN`, щоб запобігти обрамленню вмісту сайту іншими сайтами, що може допомогти у зменшенні атак клікджекінгу.

В цілому, бекенд в Django ефективно управляє потоком даних в додатку, забезпечуючи цілісність і безпеку даних, одночасно надаючи чуйний користувацький інтерфейс. Django ORM для взаємодії з базами даних абстрагується від багатьох складнощів управління базами даних, дозволяючи вам зосередитися на реалізації бізнес-логіки. Оскільки додаток обробляє конфіденційну інформацію, дуже важливо забезпечити надійність механізмів автентифікації, перевірки даних та управління сеансами. Таке налаштування не лише ефективно підтримує поточні функціональні можливості, але й забезпечує масштабованість, що дозволяє легко розширювати та підтримувати додаток у міру його зростання або появи нових функцій. [34]

2.3. Розробка інтерфейсу користувача

Принципи проєктування зручних інтерфейсів у вебдодатку зосереджені на тому, щоб зробити додаток доступним, інтуїтивно зрозумілим та ефективним для

взаємодії з ним користувачів. Це охоплювали цілий ряд стратегій від макета і навігації до адаптивності та естетики.

Інтерфейс користувача додатка широко використовує HTML і CSS для створення чистої та зручної для навігації структури. Наприклад, у макеті форми запиту, запитів на оновлення та інформаційної панелі послідовно використовуються заголовки, колонтитули та навігаційні посилання, які підтримують однорідність на всіх сторінках.

Додаток також адаптується до різних розмірів пристроїв за допомогою принципів адаптивного вебдизайну. Використання метатегу viewport з параметрами width=device-width і initial-scale=1 гарантує, що макет буде оптимізовано для пристрою перегляду, хай то настільний комп'ютер, планшет або смартфон.

Додаток використовує просту, але ефективну колірну схему і типографіку, які сприяють візуальній привабливості та читабельності. Файли CSS, на які посилаються шаблони, визначають стиль елементів, використовуючи кольори та шрифти, які легко сприймаються очима. Наприклад, налаштування фону, вибір шрифтів і дизайн кнопок створюють візуально цілісний і привабливий інтерфейс.

Навігація полегшується завдяки чітко позначеним посиланням і кнопкам, які стратегічно розміщені у фіксованому нижньому колонтитулі. Це гарантує, що користувачі можуть легко переходити від одного розділу до іншого без зайвих пошуків або плутанини. Посилання в нижньому колонтитулі, що повторюються на різних сторінках, такі як «Запит», «Оновлення», «Очікує» і «Перевірити статус», слугують опорними точками для навігації, роблячи подорож користувача по додатку передбачуваною і простою. Форма запиту (додаток Д) є ключовим інтерфейсним елементом у програмному забезпеченні для ремонту комп'ютерної техніки. Вона дозволяє користувачам вносити інформацію про нові ремонтні завдання, включаючи дані про клієнта, тип пристрою, бренд, модель, серійний номер, а також детальний опис проблеми. [38, 40]

Інтерактивні елементи, такі як форми і таблиці, розроблені з урахуванням досвіду користувача. Наприклад, форми мають чіткий дизайн з позначеними полями, які відповідають очікуваному введенню, що зменшує кількість помилок і розчарувань. Таблиці, такі як ті, що використовуються в списку очікуваних ремонтів, стилізовані для покращення читабельності та інтерактивності. Ефекти наведення та клікабельні рядки в таблицях надають візуальний зворотний зв'язок користувачам, вказуючи на інтерактивність елементів, що підвищує зручність користування додатком.

Наприклад, таблиця поточних завдань (додаток Е) відображає всі ремонтні роботи, які очікують на обробку. Вона включає поля, такі як ID квитанції, дата запиту, тип пристрою, а також опис проблеми. Цей інтерфейсний елемент допомагає орієнтуватися в поточних завданнях, спланувати робочий процес та ефективно розподіляти ресурси для ремонту.

Реалізація адаптивного дизайну у вебдодатку мала вирішальне значення для забезпечення належної адаптації інтерфейсу на різних пристроях, забезпечуючи безперебійний користувацький досвід незалежно від того, чи здійснюється доступ до нього з настільного комп'ютера, планшета або смартфона. Така адаптивність була досягнута завдяки поєднанню технік CSS, гнучких макетів і медіазапитів, які адаптували відображення відповідно до характеристик пристрою, що використовується.

В основі адаптивного дизайну лежить використання гнучких, масштабованих блоків для елементів макета і тексту. У додатку використовувалися таблиці стилів CSS, де розміри (ширина, висота, відступи, поля) визначалися у відносних одиницях, таких як відсотки (`%`), `em` або `rem`, а не в абсолютних одиницях, таких як пікселі (`px`). Це дозволило макету динамічно підлаштовуватися під розмір екрана.

Медіазапити були ключовим інструментом в адаптивному дизайні, дозволяючи CSS застосовувати різні стилі залежно від умов пристрою перегляду.

Додаток використовував медіазапити для визначення різних властивостей пристрою, таких як ширина екрана, роздільна здатність і орієнтація. Ось як вони зазвичай реалізуються.

Медіазапити визначали різні стилі для різних розмірів екрана. Наприклад, було встановлено точки розриву при ширині екрана, характерній для телефонів (<768 пікселів), планшетів (≥768 пікселів і <992 пікселів) і настільних комп'ютерів (≥992 пікселів). Кожна точка розбиття налаштовує сітки макета, змінює розмір тексту або змінює стиль навігаційного меню (наприклад, з горизонтального на випадаючий на менших екранах).

Також було визначено коригування залежно від того, чи пристрій перебуває у книжковій, чи альбомній орієнтації. Це особливо корисно для таких пристроїв, як планшети та смартфони, де користувачі можуть часто змінювати орієнтацію.

Використання метатега viewport у розділі HTML `<head>` кожного документа гарантувало, що рушій верстки відповідатиме ширині екрана у пікселях, незалежних від пристрою. Цей тег контролював розмір і масштаб області перегляду макета і був критично важливим для адаптивного дизайну. Зазвичай він мав такий вигляд:

<meta name=«viewport» content=«width=device-width, initial-scale=1»>

Цей тег вказує браузеру встановити ширину області перегляду на ширину пристрою і не масштабувати сторінку спочатку, що означало, що браузер відображав ширину сторінки по ширині його власного екрана.

У дизайні використовувалися сітки CSS або макети флексбоксів, які були визначені у гнучкий спосіб. Наприклад, використання властивостей `flexbox` дозволило розмістити контент на однаковій відстані або вирівняти його в різних конфігураціях, які адаптуються до доступного простору екрану.

Адаптивне CSS-стилізування застосовується за допомогою таблиць стилів, пов'язаних з HTML-шаблонами. Використання CSS дозволяє створювати гнучкі макети та стилі, які адаптуються до розміру екрану.

Додаток використовує зовнішні CSS-файли для стилізації, які включають правила для макетів, шрифтів, кольорів тощо. Посилання на ці таблиці стилів містяться у розділі `<head>` HTML-документів.

Адаптивний дизайн у вебдодатку був ретельно реалізований завдяки поєднанню гнучких макетів, технік CSS та стратегічного використання медіазапитів. Такий підхід гарантував, що незалежно від пристрою або розміру екрана, додаток залишався зручним та естетично привабливим, покращуючи загальний користувацький досвід та полегшуючи повсякденну роботу користувачів на різних пристроях.

Принципи дизайну, застосовані в інтерфейсі користувача додатка, зосереджені на створенні середовища, яке є привітним і простим у використанні. Наголошуючи на послідовності, оперативності, естетичній якості, чіткій навігації та інтерактивному зворотному зв'язку, додаток не лише відповідає функціональним вимогам, але й забезпечує позитивний користувацький досвід, що є важливим для інструменту, який підтримує щоденні операції в технічному та потенційно швидкозмінному середовищі.

РОЗДІЛ 3
ТЕСТУВАННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО ПРОДУКТУ

Стратегія обробки помилок у додатку в першу чергу полягає у відображенні повідомлення про помилку користувачеві на спеціальній сторінці помилок, коли досягається умова помилки. Цей підхід реалізовано у різних поданнях у файлі `repair/views.py`.

Помилка автентифікації з’являється, коли аутентифікація не вдалася через неправильні облікові дані або спробу отримати доступ до інформаційної панелі без входу в систему. У такому випадку користувача перенаправляють на сторінку помилок з відповідним повідомленням про помилку. Наприклад, при неправильному збігу облікових даних для входу відображається повідомлення «Невірні облікові дані для входу».

Обробка форми запиту здійснюється через подання, які перевіряють, чи метод запиту є POST або GET. Якщо метод не збігається з очікуваним, то користувача перенаправляють на сторінку помилок з повідомленням «Invalid Request».

Щодо обробки помилок запитів до бази даних, то програма спробує отримати певні записи з бази даних за допомогою методу get. Якщо вказаного запису не існує, програма перехоплює виняток ObjectDoesNotExist і виводить повідомлення про помилку, відповідне до контексту, наприклад «No Record Exists for Receipt ID/No».

Операції надсилання та оновлення форм перевіряють метод запиту і продовжують роботу тільки якщо це POST. Якщо це не так, або якщо будь-які обов'язкові поля форми відсутні чи недійсні, виводиться повідомлення про помилку «Недійсний запит». Якщо статус запиту не відповідає очікуваним значенням, відображається повідомлення про помилку, що вказує на наявність недійсних значень у базі даних.

У всіх випадках, коли виникає помилка, додаток використовує функцію `render` для відображення сторінки помилки (`repair/errorPage.html`), передаючи контекстний словник, який містить повідомлення про помилку та іноді додаткову інформацію, наприклад, кнопку для перенаправлення користувача або додатковий текст.

Аналіз ефективності цього програмного продукту включає вивчення багатьох аспектів, таких як організація коду, швидкість реагування, обробка помилок та потенційні сфери для оптимізації. Нижче наведено комплексний аналіз на основі наданих фрагментів.

Програмний продукт організовано в окремі шаблони та CSS-файли, що дозволяє чітко розділити презентаційний рівень від бізнес-логіки, полегшуючи обслуговування та масштабування системи. Використання окремих HTML-шаблонів і CSS-файлів, таких як 'dashboard.html' та 'style.css', для різних компонентів програми сприяє повторному використанню коду і модульності.

У додатку реалізовані принципи адаптивного дизайну, що забезпечує комфортне використання на різних пристроях, що є важливим для залучення користувачів та забезпечення доступності. Для належного масштабування інтерфейсу програми для різних пристроїв у HTML-шаблони включений метатег viewport. Також, використання властивостей CSS, пов'язаних з flexbox та медіазапитами, покращує адаптивність програми, хоча конкретні приклади медіазапитів не були надані.

Обробка помилок є важливим аспектом ефективності програмного забезпечення, що сприяє надійності та довірі користувачів. Додаток надає користувацький зворотний зв'язок у разі виникнення помилок, відображаючи певні сторінки помилок з контекстними повідомленнями. Цей підхід допомагає користувачам орієнтуватися, коли щось йде не так, але може бути розширений за допомогою більш детальних механізмів зворотного зв'язку.

Програмний продукт демонструє міцну основу з погляду організації, швидкості реагування та базової обробки помилок. Поточна структура програмного забезпечення, з чітким розподілом завдань та модульністю, закладає хороший фундамент для масштабованості.

ВИСНОВКИ

Аналіз існуючих програмних рішень виявив різноманітність інструментів, призначених для вирішення завдань ремонту комп'ютерів. Дослідження підкреслило, що, незважаючи на те, що існує низка доступних надійних рішень, залишається прогалина для інструментів, які краще інтегруються з новітніми технологіями та пропонують більш персоналізовані функції, адаптовані до конкретних потреб бізнесу. Це вказує на потенційну область для інновацій, зокрема в інтеграції функцій діагностики та ремонту з системами планування ресурсів підприємства.

Універсальність Python була очевидною в його широкому наборі бібліотек, які підходять для завдань системного адміністрування. Такі бібліотеки, як `psutil`, `os` і `subprocess`, були особливо відзначені своєю здатністю пропонувати моніторинг у реальному часі, керування процесами та виконання команд системного рівня. Використання цих бібліотек у розробці додатків підкреслює ефективність Python у створенні гнучких потужних інструментів для ремонту комп'ютера та адміністрування системи.

Постановка завдання на бакалаврську роботу передбачала визначення конкретних проблем, на вирішення яких спрямоване розроблене програмне забезпечення. Цей процес сприяв для визначення чітких, досяжних цілей, яким програмне забезпечення мало відповідати. Здатність окреслити ці цілі на ранніх стадіях процесу розробки допомогла спрямувати проєкт на задоволення реальних потреб майстрів з ремонту комп'ютерів.

Розробка алгоритмічної частини програмного забезпечення була зосереджена на створенні ефективних робочих процесів і автоматизації рутинних завдань, які є ключовими для підвищення продуктивності технічних працівників. Розроблені алгоритми полегшили керування завданнями, автоматизацію робочого процесу та обробку помилок, що є невіддільною частиною скорочення часу та зусиль,

необхідних для діагностики та ремонту комп'ютерних систем. Це підкреслює важливість ефективності алгоритму в загальній ефективності програмних засобів у цій області.

При розробці програмної частини наголос був зроблений на створенні надійної архітектури серверної частини за допомогою Django. Це включало створення захищеної масштабованої бази даних та ефективну інтеграцію зовнішньої взаємодії. Використання функцій Django, таких як його ORM, проміжне програмне забезпечення та механізм створення шаблонів, сприяло створенню безпечної та зручної програми. Конфігурація цих елементів була критично важливою для того, щоб програма могла ефективно працювати зі сценаріями реального використання.

Розробка інтерфейсу користувача була зосереджена на тому, щоб програма була доступною та зручною для користувача. Було застосовано принципи адаптивного дизайну, щоб програмне забезпечення можна було використовувати на різних пристроях і з різними розмірами екранів. Цей аспект розробки був для того, щоб техніки могли взаємодіяти з додатком інтуїтивно. Реалізація послідовної, чистої естетики та чіткості навігації покращила загальну взаємодію з користувачем, демонструючи значний вплив міркувань UI/UX у розробці програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Azvine, B., Djian, D., Tsui, K. C., & Wobcke, W. The intelligent assistant: An overview. Intelligent systems and soft computing: Prospects, tools and applications, 2000. P. 215-238.
2. Costa, A., Julian, V., & Novais, P. (Eds.). Personal Assistants: Emerging Computational Technologies. Springer, 2017. 132. P. 41-49.
3. de Barcelos Silva, A., Gomes, M. M., da Costa, C. A., da Rosa Righi, R., Barbosa, J. L. V., Pessin, G., ... & Federizzi, G. Intelligent personal assistants: A systematic literature review. Expert Systems with Applications, 2020. 147, 113193.
4. de Santiago, C., Serpa, T. P., Benevides, C. A. (Year). Repair Software: a Software Platform to Monitor and Optimise the Repair Process for Power Meters. Journal of Software Innovation, 15(3), 202-210.
5. Fernandez, E. B. Building systems using analysis patterns. In Proceedings of the third international workshop on software architecture. Institute of Electrical and Electronics Engineers, 1998. P. 37-40.
6. Gore, H., Singh, R. K., Singh, A., Singh, A. P., Shabaz, M., Singh, B. K., & Jagota, V. Django: Web development simple & fast. Annals of the Romanian Society for Cell Biology, 2021. 25(6), 4576-4585.
7. Guan, F., Liu, H., Bai, Y., & Qian, X. Agile Development Method Based on Django Framework. In Proceedings of the 4th International Conference on Power, Electronics and Computer Applications, ICPECA 2024. Institute of Electrical and Electronics Engineers (IEEE), 2024. P. 898-902.
8. Houston, L. Mobile phone repair knowledge in downtown Kampala: Local and trans-local circulations. In S. Thompson (Ed.), Repair Work Ethnographies: Revisiting Breakdown, Relocating Materiality. Springer, 2019. P, 129-160.

9. Kraus, K. Is your support services train derailing? How one integrated software package got us back on track. In Proceedings of the 35th annual ACM SIGUCCS fall conference. Association for Computing Machinery, 2007. P. 194-197.
10. Santos, J., Rodrigues, J. J., Casal, J., Saleem, K., & Denisov, V. Intelligent personal assistants based on internet of things approaches. IEEE Systems Journal, 2016. 12(2). P. 1793-1802.
11. Shaw, B., Badhwar, S., Bird, A., KS, B. C., & Guest, C. Web pythont with Django: Learn to build modern web applications with a Python-based framework. Birmingham: Packt Publishing Ltd, 2021. 307 p.
12. Tam, C., da Costa Moura, E. J., Oliveira, T., & Varajão, J. The factors influencing the success of on-going agile software development projects. International Journal of Project Management, 2020. 38(3). P. 165-176.
13. Valero, C., Pérez, J., Solera-Cotanilla, S., Vega-Barbas, M., Suarez-Tangil, G., Alvarez-Campana, M., & López, G. Analysis of security and data control in smart personal assistants from the user's perspective. Future Generation Computer Systems, 2023. P. 12-23.
14. Mark Reed, Independently published. Python Programming and SQL: 5 books in 1 - The #1 Coding Course from Beginner to Advanced. Learn it Well & Fast
15. Офійційний сайт Django Web Framework. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django (дата звернення: 14.08.2023)
16. Israel Joshua Chukwubueze. Python/Django for Beginners: Building Your First Todo List Web App (chatgpt book writing and ai tools 17). 2024, 84 c.

17. Eric Matthes, No Starch Press; 2nd edition. Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming. 2019, 544 с.
18. William S. Vincent, WelcomeToCode. Django for Beginners: Build Websites with Python and Django. 2020, 345 с.
19. William S. Vincent, WelcomeToCode. Django for Professionals: Production websites with Python & Django. 2020, 314 с.
20. Офіційний сайт Python for Web Development. URL: https://djangostars.com/blog/python-web-development/ (дата звернення: 23.03.2024)
21. Doug Hellmann, Addison-Wesley Professional; 1st edition. Python 3 Standard Library by Example, The (Developer's Library). 2017, 1456 с.
22. John Canning, Addison-Wesley Professional; 1st edition. Data Structures & Algorithms in Python (Developer's Library). 2022, 928 с.
23. David DuRocher, ClydeBank Media LLC. HTML and CSS QuickStart Guide: The Simplified Beginners Guide to Developing a Strong Coding Foundation, Building Responsive Websites, and Mastering ... (Coding & Programming - QuickStart Guides). 2021, 352 с.
24. Peggy Dummitt. Database and Framework project in Python: build modern web applications with a Python framework. 2023, 372 с.
25. Офіційний сайт How to implement a Python. URL: https://dataheadhunters.com/academy/how-to-implement-a-python-based-inventory-management-system-for-retail/ (дата звернення: 06.04.2024)
26. Офіційний сайт Gjango Tutorial. URL: https://www.w3schools.com/django/ (дата звернення: 04.11.2023)
27. Scott Mueller, Que Publishing; 22nd edition. Upgrading and repairing PCs. 2015, 1184 с.

28. Morris Rosenthal, Foner Books; 3rd ed. Edition. Computer Repair with Diagnostic Flowcharts Third Edition: Troubleshooting PC Hardware Problems from Boot Failure to Poor Performance. 2013, 170 с.
29. Heinz Lange, Abacus Software Inc. PC Bios: Improve and Upgrade Your PC'S Computing Power!. 1999, 254 с.
30. Офіційний сайт Programs to Analyze and Benchmark Your Hardware. URL: https://www.techspot.com/article/2009-monitoring-analysis-benchmarking-software-apps/ (дата звернення: 01.05.2024)
31. Офіційний сайт The subprocess Module. URL: https://realpython.com/python-subprocess/. (Дата звернення: 13.02.2024)
32. Офіційний сайт OS Module in Python with Examples. URL: https://www.geeksforgeeks.org/os-module-python-examples/. (Дата звернення: 19.12.2023)
33. Офіційний сайт Psutil documentation. URL: https://psutil.readthedocs.io/en/latest/. (Дата звернення: 08.04.2024)
34. Офіційний сайт Django ORM and QuerySets. URL: https://tutorial.djangogirls.org/en/django_orm/. (дата звернення: 12.05.2024)
35. Daniel Roy Greenfeld, Audrey Roy Greenfeld, Two Scoops Press; 3rd edition. Two Scoops of Django: Best Practices for Django 1.8. 2015, 532 с.
36. Harry Percival, O'Reilly Media; 2nd edition. Test-Driven Development with Python: Obey the Testing Goat: Using Django, Selenium, and JavaScript. 2017 622 с.
37. Arun Ravindran, Packt Pub Ltd. Django Design Patterns and Best Practices. 2015, 222 с.
38. Asad Jibran Ahmed, Packt Publishing. Django Project Blueprints. 2016, 264 с.
39. Nigel George, GNW Independent Publishing; 1st edition. Mastering Django: Core: The Complete Guide to Django 1.8 LTS. 2016, 645 с.

40.Офіційний сайт Tkinter — Python interface to Tcl/Tk. URL: https://docs.python.org/3/library/tkinter.html. (Дата звернення 30.01.2024)

ДОДАТКИ

ДОДАТОК А

Лістинг вебзастосунку (файл views)

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
from django.urls import reverse
from django.contrib.auth import authenticate, login, logout


from django.core.exceptions import ObjectDoesNotExist,  EmptyResultSet

import datetime

from repair.models import *


def login_view(request):
    if request.method == "POST":

        # Attempt to sign user in
        username = request.POST["username"]
        password = request.POST["password"]
        user = authenticate(request, username=username, password=password)

        # Check if authentication successful
        if user is not None:
            login(request, user)
            return HttpResponseRedirect(reverse("index"))
        else:
            return render(request, "auctions/login.html", {
                "message": "Invalid username and/or password."
            })
    else:
        return render(request, "auctions/login.html")


def logout_view(request):
    logout(request)
    return HttpResponseRedirect(reverse("dashboard"))
```

```
# Create your views here.
def dashboard(request):
    if request.method == "GET":
        return render(request,'repair/dashboard.html')
    elif not request.user.is_authenticated:
        # Redirect to login page or show error

        if(request.method == 'POST'):

            username = request.POST.get('username','');
            password = request.POST.get('password','');

            user = authenticate(username=username, password=password)
            if user is not None:
                return render(request,'repair/dashboard.html')
            else:
                message = {
                    'errorMessage':'Invalid Login Credentials',
                    'button':'../',
                    'buttonText':'Login',
                }
                return render(request,'repair/errorPage.html', message)
        else:
            message = {
                'errorMessage':'You must login first',
                'button':'../',
                'buttonText':'Login',
            }
            return render(request,'repair/errorPage.html', message)
    else:
        return render(request,'repair/dashboard.html')

def enquiryForm(request):
    Id = Enquiry.objects.latest('receiptID')
    #Id = Enquiry.objects.earliest('receiptID')
    Id.receiptID +=1

    currentDate = datetime.datetime.now().date()
    currentDate = str(currentDate)
```

```
        return       render(request,'repair/enquiryForm.html',       {'receiptID':
Id.receiptID, 'currentDate' : currentDate})

    def enquiryReceipt(request):
        if(request.method == 'POST'):
            enquiryDate             =             request.POST.get('enquiryDate',
datetime.datetime.now().date())
            customerName = request.POST.get('customerName','')
            contactNo = request.POST.get('contactNo','')
            email = request.POST.get('email','')
            address = request.POST.get('address','')

            deviceType = request.POST.get('deviceType','')
            brand = request.POST.get('brand','')
            deviceModel = request.POST.get('deviceModel','')
            serialNo = request.POST.get('serialNo','')
            deviceCondition = request.POST.get('deviceCondition','')

            problemCategory = request.POST.get('problemCategory','')
            problem = request.POST.get('problem','')
            problemDescription = request.POST.get('problemDescription','')

            estimatedCost = request.POST.get('estimatedCost',0)
            advance = request.POST.get('advance',0)

            if advance == '':
                advance = 0

            enq   =   Enquiry(enquiryDate=enquiryDate,   customerName=customerName,
contactNo  =  contactNo,  email=email,  address=address,  deviceType  =  deviceType,
brand=brand,      deviceModel=      deviceModel,      serialNo      =      serialNo,
deviceCondition=deviceCondition,    problemCategory  =  problemCategory,  problem  =
problem,  problemDescription  =  problemDescription,  estimatedCost  =  estimatedCost,
advance = advance)
            enq.save()

            enquiryObject = Enquiry.objects.latest('receiptID')
            return  render(request,'repair/enquiryReceipt.html',  {'receiptID'  :
enquiryObject.receiptID})
```

```
        else:
            message = {
                'errorMessage':'Invalid Request',
            }
            return render(request,'repair/errorPage.html', message)

    def enquiryReceiptFrameContent(request):
        if(request.method=='GET'):
                receiptID = request.GET.get('receiptID','')

                customerDetails = Enquiry.objects.filter(receiptID = receiptID)

                return  render(request,  'repair/enquiryReceiptFrameContent.html',
{'customerDetails' : customerDetails})
        else:
            message = {
                'errorMessage':'Invalid Request',
            }
            return render(request,'repair/errorPage.html', message)


    def updateRequest(request):
        return render(request,'repair/updateRequest.html')

    def updateForm(request):
        if(request.method=='POST'):
            if(request.POST.get('requestType','')=='receiptID'):
                receiptID = request.POST.get('receiptID',0)
                customerDetails = Enquiry.objects.filter(receiptID = receiptID)

                try:
                    enquiryInstance = Enquiry.objects.get(receiptID = receiptID)
                except ObjectDoesNotExist:
                    message = {
                    'errorMessage':'No  Record  Exists  for  Receipt  ID/No  :  '  +
receiptID,
                    }
                    return render(request,'repair/errorPage.html', message)
            elif(request.POST.get('requestType','')=='serialNo'):
                serialNo = request.POST.get('serialNo','')
```

```
                customerDetails = Enquiry.objects.filter(serialNo = serialNo)

                try:
                    enquiryInstance = Enquiry.objects.get(serialNo = serialNo)
                except ObjectDoesNotExist:
                    message = {
                    'errorMessage':'No Record Exists for Device Serial No. : ' +
serialNo,
                    }
                    return render(request,'repair/errorPage.html', message)
            else:
                message = {
                    'errorMessage':'Undefined Request Type',
                }
                return render(request,'repair/errorPage.html', message)

            # Add logic Here
            # Check Status
            if customerDetails[0].status == 'EN':
                data = {
                    'customerDetails' : customerDetails,
                }
                return render(request, 'repair/updateForm.html', data)
            elif customerDetails[0].status == 'CH':
                testDetails = TestDetail.objects.filter(Enquiry = enquiryInstance)
                data = {
                    'customerDetails' : customerDetails,
                    'testDetails' : testDetails,
                }
                return render(request, 'repair/updateForm.html', data)
            elif customerDetails[0].status == 'RE' or customerDetails[0].status ==
'CO':
                testDetails = TestDetail.objects.filter(Enquiry = enquiryInstance)
                repairDetails      =      RepairDetail.objects.filter(Enquiry      =
enquiryInstance)
                componentsUsed = str(repairDetails[0].componentsUsed)

                componentsUsedBackup = str(componentsUsed)
                componentsUsed = componentsUsed.split('~')
```

```
                componentsUsed = [i.split(':') for i in componentsUsed]
                del(componentsUsed[0])
                componentsUsedPriceTotal    =    str(sum([int(i[1])    for    i    in
componentsUsed]))

                balance         =         int(repairDetails[0].totalPrice)         -
int(customerDetails[0].advance)

                data = {
                    'customerDetails' : customerDetails,
                    'testDetails' : testDetails,
                    'repairDetails' : repairDetails,
                    'componentsUsed' : componentsUsed,
                    'componentsUsedPriceTotal' : componentsUsedPriceTotal,
                    'balance' : balance,
                }
                return render(request, 'repair/updateForm.html', data)
            elif customerDetails[0].status == 'RJ':
                message = {
                    'errorMessage':'This Record is Rejected',
                }
                return render(request,'repair/errorPage.html', message)
            else:
                message = {
                    'errorMessage':'Your Database contains some invalid values for
Enquiry.Status',
                }
                return render(request,'repair/errorPage.html', message)
        else:
            message = {
                'errorMessage':'Invalid Request',
            }
            return render(request,'repair/errorPage.html', message)

    def updateTestDetails(request):
        if(request.method=='POST'):
            receiptID = request.POST.get('receiptID','')
            actualProblem = request.POST.get('actualProblem','')
            actualProblemDescription                                          =
request.POST.get('actualProblemDescription','')
```

```
            enquiryObj = Enquiry.objects.get(receiptID=receiptID)
            # Changing Status
            enquiryObj.status = 'CH'
            enquiryObj.save()
            # Updating Test Details
            testDetailsObj = TestDetail(Enquiry = enquiryObj, actualProblem =
actualProblem, actualProblemDescription = actualProblemDescription)
            testDetailsObj.save()
            message = {
                'errorMessage':'Checking Details Updated Successfully',
                'button':'../',
                'buttonText':'Back',
            }
            return render(request,'repair/errorPage.html', message)
        else:
            message = {
                'errorMessage':'Invalid Request',
                'button':'../',
                'buttonText':'Back',
            }
            return render(request,'repair/errorPage.html', message)

    def updateRepairDetails(request):
        if(request.method=='POST'):
            receiptID = request.POST.get('receiptID','')
            componentsUsed = request.POST.get('componentsUsed','')
            repairCharge = request.POST.get('repairCharge',0)
            otherCharge = request.POST.get('otherCharge',0)

            if repairCharge == '':
                repairCharge = 0

            if otherCharge == '':
                otherCharge = 0

            # Changing Status
            enquiryObj = Enquiry.objects.get(receiptID=receiptID)
            enquiryObj.status = 'RE'
            enquiryObj.save()
```

```
            if (componentsUsed != ''):
                componentsUsedBackup = str(componentsUsed)
                componentsUsed = componentsUsed.split('~')
                componentsUsed = [i.split(':') for i in componentsUsed]
                del(componentsUsed[0])
                componentsUsedPrice = [int(i[1]) for i in componentsUsed]

                totalPrice = sum(componentsUsedPrice) + int(repairCharge) + int(otherCharge)

                componentsUsed = str(componentsUsedBackup)
            else:
                totalPrice = int(repairCharge) + int(otherCharge)

            # Updating Repair Details
            repairDetailsObj = RepairDetail(Enquiry = enquiryObj, componentsUsed = componentsUsed, repairCharge = repairCharge, otherCharge = otherCharge, totalPrice = totalPrice)
            repairDetailsObj.save()

            message = {
                'errorMessage':'Repair Details Updated Successfully',
                'button': '../finalReceipt?&receiptID='+ receiptID,
                'buttonText':'Generate Final Bill',
            }
            return render(request,'repair/errorPage.html', message)
        else:
            message = {
                'errorMessage':'Invalid Request',
                'button':'../',
                'buttonText':'Back',
            }
            return render(request,'repair/errorPage.html', message)

    def finalReceipt(request):
        if(request.method=='GET'):
            receiptID = request.GET.get('receiptID','')

            # Changing Status
```

```
            enquiryObj = Enquiry.objects.get(receiptID=receiptID)
            enquiryObj.status = 'CO'
            enquiryObj.save()

            data = {
                'receiptID' : receiptID,
            }
            return render(request, 'repair/finalReceipt.html', data)
        else:
            message = {
                'errorMessage':'Invalid Request',
            }
            return render(request,'repair/errorPage.html', message)
    def finalReceiptFrameContent(request):
        if(request.method=='GET'):
            receiptID = request.GET.get('receiptID','')
            customerDetails = Enquiry.objects.filter(receiptID = receiptID)
            enquiryInstance = Enquiry.objects.get(receiptID = receiptID)

            testDetails = TestDetail.objects.filter(Enquiry = enquiryInstance)
            repairDetails = RepairDetail.objects.filter(Enquiry = enquiryInstance)

            componentsUsed = str(repairDetails[0].componentsUsed)

            componentsUsedBackup = str(componentsUsed)
            componentsUsed = componentsUsed.split('~')
            componentsUsed = [i.split(':') for i in componentsUsed]
            del(componentsUsed[0])
            componentsUsedPriceTotal    =    str(sum([int(i[1])    for    i    in
componentsUsed]))

            balance          =          int(repairDetails[0].totalPrice)          -
int(customerDetails[0].advance)

            data = {
                'customerDetails' : customerDetails,
                'testDetails' : testDetails,
                'repairDetails' : repairDetails,
                'componentsUsed' : componentsUsed,
                'componentsUsedPriceTotal' : componentsUsedPriceTotal,
```

```
                'balance' : balance,
            }
            return render(request, 'repair/finalReceiptFrameContent.html', data)
        else:
            message = {
                'errorMessage':'Invalid Request',
            }
            return render(request,'repair/errorPage.html', message)


    def pendingRepairRequest(request):
        today = datetime.date.today()
        weekAgo = today - datetime.timedelta(days=7)
        monthAgo = today - datetime.timedelta(days=30)
        today, weekAgo, monthAgo = str(today), str(weekAgo), str(monthAgo)
        dates = {
            'today' : today,
            'weekAgo' : weekAgo,
            'monthAgo' : monthAgo,
        }
        return render(request,'repair/pendingRepairRequest.html', dates)

    def pendingRepairList(request):
        if(request.method=='POST'):
            dateFrom = request.POST.get('dateFrom','')
            dateTo = request.POST.get('dateTo','')
            #Sample.objects.filter(date__year='2011', date__month='01')
            pendingRepairs = Enquiry.objects.filter(enquiryDate__range = [dateFrom,
dateTo], status = 'EN')
            return        render(request,        'repair/pendingRepairList.html',
{'pendingRepairs' : pendingRepairs})
        else:
            message = {
                'errorMessage':'Invalid Request',
            }
            return render(request,'repair/errorPage.html', message)

    def checkStatusRequest(request):
        return render(request,'repair/checkStatusRequest.html')
```

```
    def checkStatusShow(request):
        if(request.method=='POST'):
            if(request.POST['requestType']=='receiptID'):
                receiptID = request.POST.get('receiptID','')

                customerDetails = Enquiry.objects.filter(receiptID = receiptID)

                try:
                    enquiryInstance = Enquiry.objects.get(receiptID = receiptID)
                except ObjectDoesNotExist:
                    message = {
                    'errorMessage':'No Record Exists for Receipt ID/No : ' +
receiptID,
                    }
                    return render(request,'repair/errorPage.html', message)
            elif(request.POST.get('requestType','')=='serialNo'):
                serialNo = request.POST.get('serialNo','')

                customerDetails = Enquiry.objects.filter(serialNo = serialNo)

                try :
                    enquiryInstance = Enquiry.objects.get(serialNo = serialNo)
                except ObjectDoesNotExist:
                    message = {
                    'errorMessage':'No Record Exists for Device Serial No : ' +
serialNo,
                    }
                    return render(request,'repair/errorPage.html', message)

            elif(request.POST.get('requestType','')=='personalDetails'):
                customerName = request.POST.get('customerName','')
                contactNo = request.POST.get('contactNo','')

                customerDetails = Enquiry.objects.filter(contactNo =
contactNo).filter(customerName = customerName)

                try:
                    enquiryInstance = Enquiry.objects.get(contactNo = contactNo,
customerName = customerName)
                except ObjectDoesNotExist:
```

```
                    message = {
                    'errorMessage':'No Record Exists for Customer : ' + customerName
+ " with Contact No " + contactNo,
                    }
                    return render(request,'repair/errorPage.html', message)

            else:
                message = {
                    'errorMessage':'Invalid Request Type',
                }
                return render(request,'repair/errorPage.html', message)

            # Check Status
            if customerDetails[0].status == 'EN':
                data = {
                    'customerDetails' : customerDetails,
                }
                return render(request, 'repair/checkStatusShow.html', data)
            elif customerDetails[0].status == 'CH':
                testDetails = TestDetail.objects.filter(Enquiry = enquiryInstance)
                data = {
                    'customerDetails' : customerDetails,
                    'testDetails' : testDetails,
                }
                return render(request, 'repair/checkStatusShow.html', data)
            elif customerDetails[0].status == 'RE' or customerDetails[0].status ==
'CO':
                testDetails = TestDetail.objects.filter(Enquiry = enquiryInstance)
                repairDetails     =     RepairDetail.objects.filter(Enquiry     =
enquiryInstance)
                if repairDetails:
                    componentsUsed = str(repairDetails[0].componentsUsed)

                    componentsUsedBackup = str(componentsUsed)
                    componentsUsed = componentsUsed.split('~')
                    componentsUsed = [i.split(':') for i in componentsUsed]
                    del(componentsUsed[0])
                    componentsUsedPriceTotal   =   str(sum([int(i[1])   for   i   in
componentsUsed]))
```

```
                else:
                    componentsUsed = None

                balance         =         int(repairDetails[0].totalPrice)         -
int(customerDetails[0].advance)

                data = {
                    'customerDetails' : customerDetails,
                    'testDetails' : testDetails,
                    'repairDetails' : repairDetails,
                    'componentsUsed' : componentsUsed,
                    'componentsUsedPriceTotal' : componentsUsedPriceTotal,
                    'balance' : balance,
                }
                return render(request, 'repair/checkStatusShow.html', data)
            elif customerDetails[0].status == 'RJ':
                message = {
                    'errorMessage':'This Record is Rejected',
                }
                return render(request,'repair/errorPage.html', message)
            else:
                message = {
                    'errorMessage':'Your Database contains some invalid values for
Enquiry.Status',
                }
                return render(request,'repair/errorPage.html', message)
        else:
            message = {
                'errorMessage':'Invalid Request',
            }
            return render(request,'repair/errorPage.html', message)
```

ДОДАТОК Б

Лістинг вебзастосунку (файл urls)

```
from django.urls import path
from .views import (
    dashboard, enquiryForm, enquiryReceipt, enquiryReceiptFrameContent,
    updateRequest, updateForm, updateTestDetails, updateRepairDetails,
    finalReceipt, finalReceiptFrameContent, pendingRepairRequest,
    pendingRepairList, checkStatusRequest, checkStatusShow, login_view,
logout_view,
)

urlpatterns = [
    path('', dashboard, name='dashboard'),
    path("login", login_view, name="login"),
    path("logout", logout_view, name="logout"),
    path('enquiryForm/', enquiryForm, name='enquiryForm'),
    path('enquiryReceipt/', enquiryReceipt, name='enquiryReceipt'),
    path('enquiryReceiptFrameContent/', enquiryReceiptFrameContent,
name='enquiryReceiptFrameContent'),
    path('updateRequest/', updateRequest, name='updateRequest'),
    path('updateForm/', updateForm, name='updateForm'),
    path('updateTestDetails/', updateTestDetails, name='updateTestDetails'),
    path('updateRepairDetails/', updateRepairDetails,
name='updateRepairDetails'),
    path('finalReceipt/', finalReceipt, name='finalReceipt'),
    path('finalReceiptFrameContent/', finalReceiptFrameContent,
name='finalReceiptFrameContent'),
    path('pendingRepairRequest/', pendingRepairRequest,
name='pendingRepairRequest'),
    path('pendingRepairList/', pendingRepairList, name='pendingRepairList'),
    path('checkStatusRequest/', checkStatusRequest,
name='checkStatusRequest'),
    path('checkStatusShow/', checkStatusShow, name='checkStatusShow'),
]
```

ДОДАТОК В

Лістинг вебзастосунку (файл models)

```
from django.db import models

# Create your models here.
class Enquiry(models.Model):
    PROBLEM_CATEGORY_CHOICES = (
        ('HW','Hardware'),
        ('SW','Software'),
    )

    STATUS_CHOICES = (
        ('EN','Enquired'),
        ('CH','Checked'),       # This is amazing
        ('RE','Repaired'),      # Changed from Update Form when Components are added and Repair Charge is added
        ('CO','Completed'),     # Changed to when Final Receipt is Generated
        ('RJ','Rejected'),
    )

    receiptID = models.AutoField(primary_key=True)
    enquiryDate = models.DateField()
    customerName = models.CharField(max_length = 50)
    contactNo = models.CharField(max_length = 20)
    email = models.CharField(max_length = 50, blank=True)
    address = models.TextField(blank=True)
    deviceType = models.CharField(max_length = 50)
    brand = models.CharField(max_length = 50, blank=True)
    deviceModel = models.CharField(max_length = 50)
    serialNo = models.CharField(max_length = 50, blank=True)
    problemCategory = models.CharField(max_length = 3, choices = PROBLEM_CATEGORY_CHOICES)
    problem = models.CharField(max_length = 50)
    problemDescription = models.TextField(blank=True)
    deviceCondition = models.TextField(blank=True)
    estimatedCost = models.IntegerField()
    advance = models.IntegerField(default=0)
    status = models.CharField(max_length=3, choices = STATUS_CHOICES, default='EN')
```

```
        def __str__(self):
            return  str(self.receiptID)  +  "  :  "    +  self.status  +  "  :  "    +
self.customerName + " : " + self.brand + " " + self.deviceModel

    class TestDetail(models.Model):
        Enquiry = models.ForeignKey(Enquiry, on_delete=models.CASCADE)
        actualProblem = models.CharField(max_length = 50)
        actualProblemDescription = models.TextField(blank=True)

        def __str__(self):
            return str(self.Enquiry.receiptID) + " : " + str(self.Enquiry.status)
+ " : " + self.Enquiry.customerName + " : " + self.actualProblem

    class RepairDetail(models.Model):
        Enquiry = models.ForeignKey(Enquiry, on_delete=models.CASCADE)
        componentsUsed = models.TextField(blank=True)
        repairCharge = models.IntegerField(blank=True)
        otherCharge = models.IntegerField(blank=True)
        totalPrice = models.IntegerField(blank=True)

        def __str__(self):
            return str(self.Enquiry.receiptID) + " : " + str(self.Enquiry.status)
+ " : " + self.Enquiry.customerName + " : " + str(self.totalPrice)

    class trialPeriod(models.Model):
        ID = models.AutoField(primary_key=True)
        counter = models.IntegerField()
        date = models.DateField()
```

ДОДАТОК Г

Лістинг вебзастосунку (файл settings)

```
"""
Django settings for Bucks project.

Generated by 'django-admin startproject' using Django 1.11.7.

For more information on this file, see
https://docs.djangoproject.com/en/1.11/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/1.11/ref/settings/
"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/1.11/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = ')5az3&z10_k&ijl20f=ansf_k8b(_s9aw4@))^&fp7)3a9jysu'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.admindocs',
    'django.contrib.auth',
    'django.contrib.contenttypes',
```

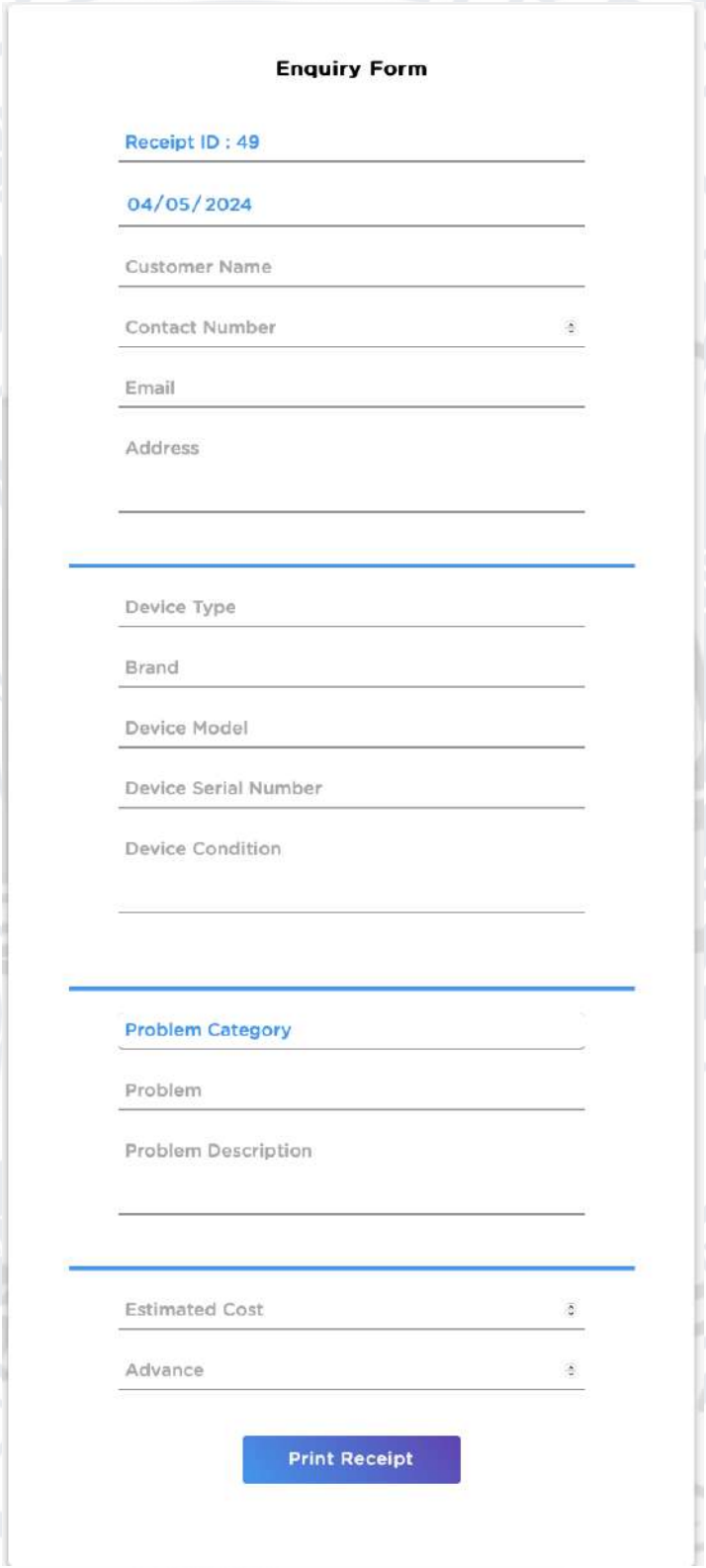
```
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'repair.apps.RepairConfig'
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'Bucks.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'Bucks.wsgi.application'


# Database
# https://docs.djangoproject.com/en/1.11/ref/settings/#databases
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}


# Password validation
#         https://docs.djangoproject.com/en/1.11/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/1.11/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'Europe/Kiev'

USE_I18N = True
```

```
USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.11/howto/static-files/

STATIC_URL = '/static/'

#STATIC_ROOT = os.path.join(BASE_DIR, 'static')
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

X_FRAME_OPTIONS = 'SAMEORIGIN'
```

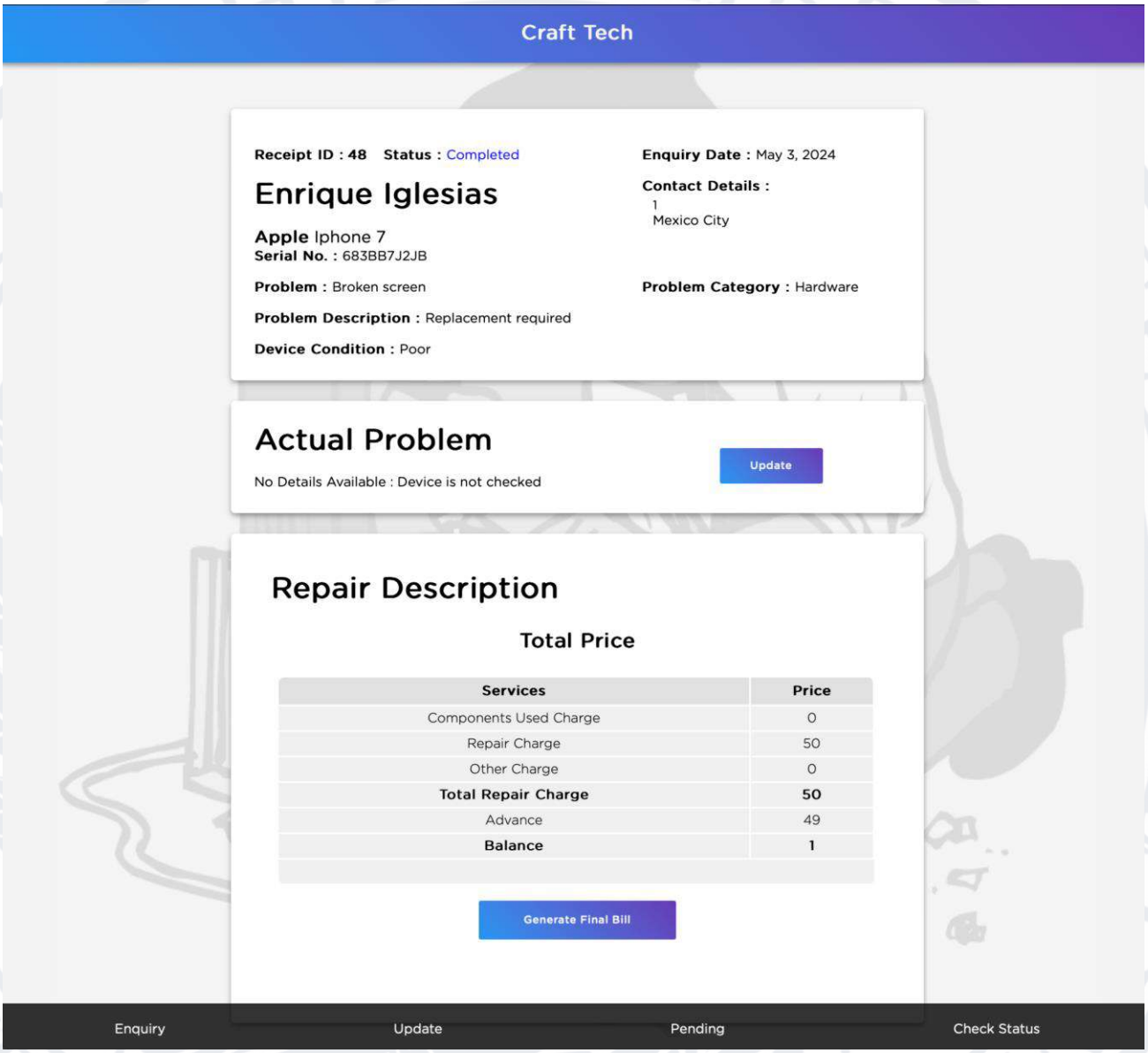
ДОДАТОК Д

Приклад інтерфейсу користувача (форма для створення нового замовлення)



ДОДАТОК Е

Приклад інтерфейсу користувача (вигляд замовлення на платформі)



ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)