МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ФАХОВИЙ КОЛЕДЖ ІНЖЕНЕРІЇ ТА УПРАВЛІННЯ
НАЦІОНАЛЬНОГО АВІАЦІЙНОГО УНІВЕРСИТЕТУ»

ДОПУСТИТИ ДО ЗАХИСТУ

Заступник директора з НР

_____О.В. Родіонова

«____»_____2021 р.

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
«БАКАЛАВР»

Тема: «Веб-додаток для ефективної комунікації робітників СТО»

Автор: О. О. Михалочкін

Керівник проєкту: В. В. Котелянець

Нормконтролер: В. М. Кругляк

Київ 2021

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ФАХОВИЙ КОЛЕДЖ ІНЖЕНЕРІЇ ТА
УПРАВЛІННЯ НАЦІОНАЛЬНОГО АВІАЦІЙНОГО
УНІВЕРСИТЕТУ»

Циклова комісія: Інженері програмного забезпечення

Освітнього ступеня: «Бакалавр»

Спеціальність: 123 Комп'ютерна інженерія

Освітньо-професійна програма: «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Голова циклової комісії

_____Н.А. Рябчук

«____»_____2021 р.

ЗАВДАННЯ

на виконання дипломного проєкту

Михалочкін Олександр Олександрович

Тема роботи: «Веб-додаток для ефективної комунікації робітників СТО» затверджена наказом від «15» березня 2021 року № 28-Ст.

Термін виконання: з 12.04.2021р. по 20.06.2021р.

Вихідні дані: Розробити веб-додаток з інтерфейсами для трьох ролей, які між собою передають інформацію про замовлення.

Зміст пояснювальної записки:

У 1 розділі проаналізувати завдання на проєкт та основні методи його розв'язання.

У 2 розділі наведений опис основних етапів розробки.

У 3 розділі описати основні вимоги охорони праці.

КАЛЕНДАРНИЙ ПЛАН

виконання дипломної роботи

№ п/п	Етапи виконання кваліфікаційної роботи	Термін виконання етапів	Примітка
1.	Уточнення постановки задачі	12.04.2021	Виконано
2.	Аналіз літературних джерел	15.04.2021	Виконано
3.	Обґрунтування рішення	21.04.2021	Виконано
4.	Збір інформації	01.05.2021	Виконано
5.	Аналіз існуючих методів. Обгрунтування вибору мови програмування	06.05.2021	Виконано
6.	Виконання проєкту	08.06.2021	Виконано
7.	Оформлення і друк пояснювальної записки	15.06.2021	Виконано
8.	Оформлення презентації	17.06.2021	Виконано
9.	Отримання рецензій	20.06.2021	Виконано
10.	Захист проєкту		

Дипломник Михалочкін О. О.

(підпис, дата) (П.І.Б.)

Дипломний керівник Котелянець В. В.

(підпис, дата) (П.І.Б.)

Консультант з охорони праці Пешков І.В.

(підпис, дата) (П.І.Б.)

Зміст

Вступ

З розвитком сфери обслуговування, з’являться стандарти які кожен бізнес має впроваджувати для підтримки високого рівня послуг що надаються клієнту та розвиток компанії в цілому. Такими стандартами можна назвати наприклад дотримання чіткого алгоритму прийняття замовлення, виконання та оплати.

Другим прикладом може бути швидкість виконання замовлення, яка впливає на настрій замовника та кількість виконаних замовлень за день.

Мета моєї кваліфікаційної роботи розробити веб додаток для ефективної комунікації між працівниками СТО, для швидкого введення замовлення та чіткого контролю стадій замовлення, що зменшить вплив людського фактору за попередить конфлікти на роботі.

Компанії СТО не звикли розвивати сучасний підхід для оптимізації однотипних процесів, що забирають багато робочого часу та спричиняють помилки.

Працівники сто записують необхідні на паперових листках, відразу після діагностики, коли їх руки в мазуті. Також дуже часто не вистачає спеціальних листків для написання запчастин. Тоді майстри використовують перший папір який попав під руку. Ці листки можуть бути навіть пошкоджені.

Потім ці листки відносять до менеджера який спілкуються з клієнтами. Йому потрібно розібрати почерк спеціаліста та визначити які запчастини потрібно додати до замовлення. Далі цей лист відносять до менеджера запчастин. Він також розбирає що там написано та прописує вартість кожної запчастини.

Ми вже бачимо які проблеми виникають при такому ставленні до замовлення. Дуже часто ці папірці можна загубити, та приходиться переробляти все заново.

Моя мета розробити програмне забезпечення з трьома інтерфейсами: менеджер, майстер та менеджер запчастин - саме ці три основні відділи потребують ефективного рішення.

– У майстра має бути змога максимально зручно та швидко записувати запчастини та послуги до замолвення.

– У менеджера мають бути всі необхідні функції для контролю замовлення, ніщо не повинно робитися без відома менеджера який спілкується з клієнтами, так як він відповідальний за правильне виконання робіть та репутацію перед клієнтом.

– У менеджера запчастин мають бути функції для записування можливих варіантів запчастин, від 1 до 3 варіантів на кожну запчастину.

Майстер має змогу редагувати вже існуюче замовлення тільки, якщо менеджер надасть доступ до цього замовлення.

На даний момент існує дуже мало рішення цієї проблеми. Найкраще рішення це CRM система, але всі необхідні функції має тільки одна така система і вона російська. Інші системи зараз на стадії розробки та вони для більш широких сфер бізнесу.

Звісно моя система не буде мати всіх функцій crm системи. Але вона буде мати функції яких не вистачає саме в сфері СТО.

1 Загальна частина

1.1 Постановка задачі

Перш ніж розробляти програму, потрібно визначити задачі та цілі. Постановка задач – це описання всіх вимог до розроблюваного програмного забезпечення. Правильно поставлені та описані задачі, це найкращий шлях для досягнення головних цілей, так як оптимальне рішення можна буде легше знайти.

Вимоги до реалізації проекту:

- візуальний інтерфейс користувача;
- реалізація ведення бази графічних об’єктів;
- реалізація групування об’єктів за призначенням;
- реалізація дублювання потрібних об’єктів;
- передбачити додаткові сервіси (можливість збереження створених схем).

В першу чергу, програма розробляється для користувача. Тому важливим завдання є розробка інтерфейсу користувача.

Інтерфейс користувача — засіб зручної взаємодії користувача з інформаційною системою. Сукупність засобів для обробки та відбиття інформації, якнайбільше пристосованих для зручності користувача. У графічних системах інтерфейс користувача, втілюється багатовіконним режимом, змінами кольору, розміру, видимості (прозорість, напівпрозорість, невидимість) вікон, їхнім розташуванням, сортуванням елементів вікон, гнучкими налаштуваннями як самих вікон, так і окремих їх елементів (файли, теки, шрифти тощо).

Інтерфейс користувача комп’ютерного додатку включає:

– засоби відображення інформації, відображувану інформацію, формати ікоди;

– командні режими, мову «користувач–інтерфейс»;

– пристрої та технології введення- виведення;

– діалоги, взаємодію та транзакції між користувачем та комп’ютером, зворотній зв’язок з користувачем;

– підтримку прийняття рішень в конкретній предметній області;

– порядок використання програми і документації на неї.

Користувацький інтерфейс часто розуміють лише як зовнішній вигляд програми. Однак насправді користувач сприймає через нього всю програму в цілому, тобто таке розуміння є надто вузьким.

В дійсності, користувацький інтерфейс об’єднує в собі всі елементи і компоненти програми, які здатні впливати на взаємодію користувача з програмним забезпеченням. До цих елементів належать:

– набір задач користувача, які він розв’язує за допомогою системи;

– використовувана системою метафора (наприклад, робочий стіл Windows);

– елементи управління системою;

– навігація між блоками системи;

– візуальний (і не тільки) дизайн екранів програми;

– засоби відображення інформації, відображувана інформація і формати;

– пристрої та технології введення даних;

– діалоги, взаємодія і транзакції між користувачем і комп’ютером;

– зворотній зв’язок з користувачем;

– підтримка прийняття рішення в конкретній предметній області;

– порядок використання програми і документація на неї.

Види інтерфейсу:

Командний інтерфейс. Пакетна технологія, історично цей вид технології з'явився перший. Ідея її проста: на вхід комп'ютера подається послідовність символів, в яких за певними правилами вказується послідовність запущених на виконання програм. Після виконання чергової програми

запускається наступна і т.д. Машина за певними правилами знаходить для себе команди і дані. Людина тут має малий вплив на роботу машини - користувач може лише призупинити роботу машини, змінити програму і знову запустити.

З появою алфавітно-цифрових дисплеїв по-справжньому почалася ера користувача технології - технологія командного рядка. При цій технології, єдиний спосіб введення інформації від людини до комп'ютера слугує клавіатура, а комп'ютер виводить інформацію людині за допомогою алфавітно-цифрового дисплея (монітора).

Графічний інтерфейс. Графічний інтерфейс користувача - тип інтерфейсу, який дозволяє користувачам взаємодіяти з електронними пристроями через графічні зображення та візуальні вказівки, на відміну від текстових інтерфейсів, заснованих на використанні тексту, текстовому наборі команд та текстовій навігації.

Можна виділити такі види графічного інтерфейсу:

– простий: типові екранні форми та стандартні елементи інтерфейсу, що забезпечуються самою підсистемою GUI;

– істинно-графічний, двовимірний: нестандартні елементи інтерфейсу та оригінальні метафори, що реалізовані власними засобами програми або сторонньою бібліотекою;

– тривимірний.

– WIMP інтерфейс, переводиться як - Windows, Icons, Menus, Pointers (вікно, символи, меню, вказівники).

– Другим етапом у розвитку графічного інтерфейсу став "чистий" інтерфейс WIMP, цей підвид інтерфейсу характеризується наступними особливостями:

– вся робота з програмами, файлами і документами відбувається у вікнах - певних окреслених рамкою частинах екрану;

– усі програми, файли, документи, пристрої та інші об'єкти представляються у вигляді символів - іконок. При відкритті, іконки перетворюються у вікна;

– всі дії з об'єктами здійснюються за допомогою меню. Хоча меню з'явилося на першому етапі становлення графічного інтерфейсу, воно не мало в ньому головного значення, а служило лише доповненням до командного рядка. У чистому WIMP інтерфейсі меню стає основним елементом управління;

– широке використання маніпуляторів для вказівки на об'єкти.

Маніпулятор перестає бути просто іграшкою - доповненням до клавіатури, а стає основним елементом управління. За допомогою маніпулятора користувач вказує на будь-яку область екрану, вікна або іконки і таким чином виділяє її, а вже потім через меню або з використанням інших технологій здійснюють управління ними.

Слід зазначити, що WIMP вимагає для своєї реалізації кольоровий растровий дисплей з високим дозволом і маніпулятор. Також програми, орієнтовані на цей вид інтерфейсу, потребують підвищені вимоги до продуктивності комп'ютера, обсяг його пам'яті, пропускної здатності шини і т.п. Однак цей вид інтерфейсу найбільш простий у засвоєнні та інтуїтивно зрозумілий. Тому зараз WIMP інтерфейс став стандартом для будь-якого програмного забезпечення.

Семантичний інтерфейс. Цей вид інтерфейсу виник з розвитком штучного інтелекту. Його важко назвати самостійним видом інтерфейсу - він включає в себе і інтерфейс командного рядка, і графічний, і мовної, і мімічний інтерфейс. Основна його відмінна риса - це відсутність команд при спілкуванні з комп'ютером. Запит формується на природній мові, у вигляді пов'язаного тексту і образів. За своєю суттю це важко називати інтерфейсом - це вже моделювання "спілкування" людини з комп'ютером.

При розробці інтерфейсу, потрібно слідувати принципам проектування інтерфейсів користувача. Розробники інтерфейсів завжди повинні враховувати фізичні і розумові здібності людей, які працюватимуть з програмним забезпеченням. Люди на короткий час можуть запам'ятати вельми обмежений об'єм інформації і робити помилки при введенні великих об'ємів даних або при роботі у стресових умовах. Фізичні можливості людей можуть істотно

різнитися. Основні принципи проектування інтерфейсів користувача:

врахування знань користувача — у інтерфейсі необхідно використовувати терміни й поняття, зрозумілі користувачам. Інакше кажучи, інтерфейс має бути настільки зручним, щоб користувач звикає до нього без особливих зусиль. В інтерфейсі потрібно використовувати терміни, зрозумілі користувачеві, а об'єкти, керовані системою, мають бути безпосередньо пов'язані з робочим середовищем користувача;

– узгодженість — однотипні (хоча й різні) операції виконують одним і тим самим способом;

– мінімум несподіванок — поведінка системи має бути прогнозованою;

– здатність до відновлення — інтерфейс повинен мати засоби відновлення даних після вчинення помилкових дій;

– підтримка користувача — засоби підтримки користувача потрібно вбудовати в інтерфейс. Вони мають забезпечувати різні рівні допомоги і довідкової інформації на кількох рівнях: від основ для початківців до повного опису можливостей системи. Інтерфейс має надавати необхідну інформацію у разі помилок користувача й підтримувати засоби контекстно-залежної довідки;

– підтвердження деструктивної дії — при виборі потенційно деструктивної дії користувач повинен ще раз підтвердити свій намір;

– можливість відміни дії — повернення системи у той стан, в якому вона перебувала до їх виконання. Бажана підтримка багаторівневої відміни дій, бо користувач не завжди відразу розуміє, що зприпустився помилки;

– врахування різнорідності користувачів — інтерфейс має містити засоби зручної взаємодії з користувачами, що мають різний рівень кваліфікації таі різні можливості.

– І нарешті необхідно вирішити дві головні проблеми: яким чином користувач вводитиме дані в систему і як дані буде представлено користувачеві.

– Після архітектури програми, потрібно визначити структуру. Програми, які використовуються для роботи з комп’ютером, можна поділити на три категорії:

– системні програми, які виконують різноманітні допоміжні функції, наприклад, використовуються для створення копій наявної інформації, для перевірки роботоздатності пристроїв комп’ютера і т. ін.;

– прикладні програми, що безпосередньо забезпечують виконання необхідних користувачу робіт: редагування текстів, створення малюнків, обробка інформаційних масивів тощо;

– інструментальні системи (системи програмування), які забезпечують створення нових програм для комп’ютера.

Так як в моєму кваліфікаційному проекті користувач має створювати та редагувати об’єкти, то структурою програмного модуля буде саме прикладна програма.

Наступне крок перед написанням коду програми, це архітектура. Архітектура програмного забезпечення — спосіб структурування програмної системи, абстракція елементів системи на певній фазі її роботи. Система може складатись з кількох рівнів абстракції і мати багато фаз роботи, кожна з яких може мати окрему архітектуру.

Проектування архітектури ПЗ — це процес розроблення, що виконується після етапу аналізу і формулювання вимог. Задача такого проектування — перетворення вимог до системи у вимоги до ПЗ і побудова на їхній основі архітектури системи. Побудова архітектури системи здійснюється шляхом визначення цілей системи, її вхідних і вихідних даних, декомпозиції системи на підсистеми, компоненти або модулі та розроблення її загальної структури. Проектування архітектури системи може проводитися різними методами (стандартизованим, об’єктно-орієнтованим, компонентним і ін.), кожний з яких пропонує свій шлях побудови архітектури, а саме, визначення концептуальної, об’єктної та інших моделей за допомогою відповідних конструктивних елементів (блок-схем, графів, структурних діаграм тощо).

Життє вий цикл програ много забезпе чення — сукупність окремих етапів робіт, що проводяться у заданому порядку протягом періоду часу, який починається з вирішення питання про розроблення програмного забезпечення і

закінчується припиненням використання програмного забезпечення.

Є три моделі життєвого циклу:

– Каскадна модель;
– Спіральна модель;
– Еволюційна модель.

В даному проекті я використав еволюційну модель - система послідовно розробляється з блоків конструкцій. В еволюційній моделі вимоги встановлюються частково і уточнюються в кожному наступному проміжному блоці структури системи.

Технічні характеристики проекту:

– операційна система Windows;
– мова програмування функції системи C#, інтерфейс XAML;
– середовище розробки Visual Studio 2019;
– одно-користувацький.

1.2 Теоретичні відомості

Проектування програмного забезпечення - це процес вирішення задач та планування для створення програмного рішення. Після того як мета і специфікація програми описані, розробник створить дизайн проекту, або найме дизайнера для розробки плану рішення. В дизайн включаються як описи низькорівневих компонентів, алгоритмів, так і огляд архітектури.

Протягом кількох десятиліть стоїть завдання пошуку повторюваного, передбачуваного процесу або методології, яка б поліпшила продуктивність, якість і надійність розробки. Одні намагалися систематизувати та формалізувати цей, мабуть, малопередбачуваний процес. Інші застосовували до нього методи управління проектами та методи програмної інженерії. Треті вважали, що без постійного контролю з боку замовника розробка ПЗ виходить з-під контролю, з'їдаючи зайвий час і кошти.

Досвід управління розробкою програм відбивається у відповідних

посібниках, звичаях і стандартах. Якщо при розробці використовується декілька стандартів і нормативних документів, то має сенс скласти профіль. Інформатика як наукова дисципліна пропонує і використовує на базі методів структурного програмування технологію надійної розробки програмного забезпечення, використовуючи тестування програм та їх верифікацію на основі методів доказового програмування для систематичного аналізу правильності алгоритмів і розробки програм без алгоритмічних помилок.

Дана методологія спрямована на вирішення завдань на електронних обчислювальних машинах, аналогічної технології розробки алгоритмів і програм, використовуваної на олімпіадах з програмування вітчизняними студентами та програмістами з використанням тестування і структурного псевдокоду для документування програм в корпорації IBM з 70-х років.

Методологія структурного проектування програмного забезпечення може використовуватися з застосуванням самих різних мов і засобів програмування для розробки надійних програм самого різного призначення. Одним з таких проектів була розробка бортового програмного забезпечення для космічного корабля «Буран», в якому вперше використовувався бортовий комп'ютер для автоматичного управління апарату, яка виконала успішний старт і посадку космічного корабля.

При виборі методології розробки програмного забезпечення слід керуватися тим, що складність методології порівнянна з складністю структури програмного продукту, і невиправдана для продукту даної складності складність методології тільки невиправдано збільшить вартість розробки. Прикладом сучасної методології проектування може бути проблемно-орієнтоване проектування.

Не всякий програмний модуль сприяє спрощенню програми. Виділити хороший з цієї точки зору модуль є серйозним творчим завданням. Для оцінки прийнятності виділеного модуля використовуються деякі критерії. Так, Хольт запропонував такі два загальні критерії:

– хороший модуль зовні простіший, ніж усередині;

– хороший модуль простіше використовувати, ніж побудувати.

Як я згадував раніше, я використовував еволюційну модель життєвого циклу розробки програми. Використання еволюційної моделі припускає проведення дослідження предметної області для вивчення потреб її замовника і аналізу можливості застосування цієї моделі для реалізації. Модель використовується для розробки нескладних і некритичних систем, де головною вимогою є реалізація функцій системи. При цьому вимоги не можуть бути визначені відразу і повністю. Тому розробка системи здійснюється ітераційним шляхом її еволюційного розвитку з отриманням деякого варіанта системи– прототипу, на якому перевіряється реалізація вимог. Іншими словами, такий процес за своєю суттю є ітераційним, з етапами розробки, що повторюються, починаючи від змінених вимог і закінчуючи отриманням готового продукту. В деякому розумінні до цього типу моделі можна віднести спіральну модель.

У даному проекті я використав технологію WIMP інтерфейсу. Тобто графічний інтерфейс керування яким відбувається комп’ютерною мишкою та клавіатурою. Інтерфейс передбачуваний та зрозумілий. Я використав кольорове виділення блоків для кращої орієнтації під час створення блок-схеми. Лінію автоматично вирівнюються під час їх створення, що значно полегшує з’єднання блоків.

1.3 Обґрунтування вибору мови програмування

Для виконання кваліфікаційної роботи я використав мову програмування Python, а саме Django-фреймворк для створення веб-додатків.

Чому саме Python?

Почнемо відразу з гучних, але не голослівних заяв:

– Instagram і Pinterest працюють на Python;

– Python входить в десятку найпопулярніших мов програмування в світі, поступово просуваючись все вище і вище;

– Зараз цей проект активно інвестує Google.

Ця високорівнева мова програмування, яка по праву належить до одних з найбільш простих і зрозумілих, завдяки синтаксису і читання. Вважається, що він відмінно підходить новачкам, тому що програмувати на Python виходить набагато швидше і простіше, ніж, наприклад, на Java Script. Вивчити синтаксис Python можна за допомогою звичайного туторіал на офіційному сайті. Він досить добре написаний.

Якщо ви початківець розробник, то універсальний Python стане кращим вибором.

Пайтон - мова інтерпретується і до запуску є просто текстовим файлом. Етап компіляції в ньому відсутній. Він логічний, непогано спроектований і підходить для роботи практично на всіх платформах.

Крім того, Python хороший для більшості проектів, таких як:

– Веб-розробки.

– Додатки.

– Розробки для мобільних пристроїв.

– Вбудовані системи і багато іншого.

Якщо говорити про те, де ця мова використовується найчастіше, то лідером, звичайно, будуть веб-розробки. Що стосується мобільних додатків, то на Пайтон, як правило, пишуть серверну частину, а для пристроїв на Android і IOS використовують інші мови.

Часто пайтон виступає базою для відтворення вбудованих систем, як, наприклад, в Raspberry Pi.

Мова використовується для створення засобів регулювання різних показників, таких, як температура, тиск, витрата різних ресурсів. Саме тому Python так люблять розробники популярних сьогодні систем типу «розумний дім».

У списку шанувальників цієї зрозумілої і лаконічної мови ціла армія «зоряних» компаній: IROBOT, Maya, Pixar, Intel, IBM, ESRI і навіть гентство національної безпеки США. Теоретично будь-який Python-фахівець може

спробувати і відправити резюме.

Django вважається кращим веб-фреймворком, написаним на Python. Цей інструмент зручно використовувати для розробки сайтів, що працюють з базами даних. З цієї статті ви дізнаєтеся про Django і зрозумієте, чому це ключова ланка векосистемі Python.

Django створили розробники видання Lawrence-Journal World. Цій газеті знадобився сайт, щоб публікувати новини в інтернеті. Програмісти Едріан Головатий і Саймон Віллісон створили веб-додаток і зробили його публічним.

Навколо Django швидко сформувалася активна спільнота. Фреймворк став стрімко розвиватися зусиллями волонтерів. Значну роль в успіху Django зіграли кілька відомих сайтів, які використовували цей фреймворк. В їх число входять Pinterest, Dropbox, Spotify, сайт The Washington Post. В даний час співтовариство Django включає більше 11 тис. Розробників з 166 країн світу.

Довідка: російськомовні розробники і користувачі часто використовують такий варіант написання: «Джанго». Будьте готові до цього, якщо зустрінетеся з цим варіантом в обговореннях або професійному листуванні. До речі, у нас є короткий словник професійного сленгу програмістів.

В Django реалізований принцип DRY (do not repeat yourself). Завдяки цьому скорочується час створення сайтів. Тобто при використанні Django вам не потрібно кілька разів переписувати один і той же код. Фреймворк дозволяє створювати сайт з компонентів. Це можна порівняти з будівництвом фортеці за допомогою Lego.

Django підходить для розробки високонавантажених веб-додатків. Це можливо завдяки архітектурі фреймворку, про яку піде мова нижче.

Фреймворк Django написаний на мові програмування Python, тому його структура відповідає особливостям мови. Творці реалізували в Django патерн MVC, і він застосовується в поточній версії фреймворку.

Архітектура MVC дозволяє розробнику працювати з візуальним представленням і бізнес-логікою додатка окремо. До речі, при роботі з Django фахівці частіше використовують термін MVT - Model-View-Template або

модель- уявлення-шаблон. Компоненти MVT можна використовувати незалежно один від одного.

Документація Django визначає модель (model) як «джерело інформації про дані, в яких містяться ключові поля і поведінка даних». Зазвичай одна модель вказує на одну таблицю в базі даних. Django підтримує бази даних PostgreSQL, MySQL, SQLite і Oracle.

Моделі містять інформацію про дані. Ці дані представлені атрибутами або полями. Оскільки модель являє собою простий клас, вона нічого не знає на інших рівнях Django. Взаємодія між рівнями відбувається через API.

Модель відповідає за бізнес-логіку, методи, властивості і інші елементи, пов'язані з маніпуляцією даними. Також моделі дозволяють розробникам створювати, читати, оновлювати та видаляти об'єкти в базі даних.

Подання (view) вирішує три завдання: приймає HTTP-запити, реалізує бізнес-логіку методами і властивостями, відправляє HTTP-відповідь у відповідь на запити. Тобто уявлення отримує дані від моделі і надає шаблонами (templates) доступ до цих даних або попередньо обробляє дані і потім надає до них доступ шаблонами.

В Django реалізований потужний движок шаблонів і власна мова розмітки. Шаблони являють собою файли з HTML-кодом, за допомогою якого відображаються дані. Вміст файлів може бути статичним або динамічним. Шаблони не містять бізнес-логіки. Тому вони тільки відображають дані.

Така архітектура дозволяє Django успішно вирішувати різні завдання, про які нижче написано.

Які завдання можна вирішувати за допомогою Django: движки для сайтів, CRM, machine learning.

Недосвідчені фахівці вважають Django однією з багатьох систем управління контентом (CMS). Насправді це програмний інструмент, за допомогою якого можна створювати і запускати веб-додатки.

Довідка: назва фреймворка підкреслює його багатогранність. Він названий на честь відомого джазового гітариста Джанго Рейнхардта. Цей

музикант віртуозно грав на гітарі, хоча два пальця на його лівій руці не функціонували після травми, отриманої під час пожежі. Тобто музикантові доводилося братиакорди трьома пальцями.

Фреймворк Django справляється з великою кількістю завдань і підвищеними навантаженнями. Його застосовують для створення:

– CRM-систем.
– CMS.
– Комунікаційних платформ.
– Сервісів бронювання номерів.
– Платформ управління документообігом.

Також Django підходить для створення алгоритмічних генераторів, платформ для електронних розсилок, систем верифікації, систем фільтрації з динамічними правилами і складними параметрами, платформ для аналізу даних і складних обчислень, машинного навчання.

Тисячі сайтів в різних країнах світу створені на Django. Цей фреймворк відмінно підходить для розробки веб-додатків. Давайте подивимося, через що розробники люблять даний інструмент.

Чому Django - відмінний фреймворк для веб-розробки: екосистема, SEO, бібліотеки.

Якщо ви запитаєте у кількох розробників, чому вони вибрали Django, відповіді будуть переважно однаковими. Нижче описані основні переваги фреймворка, завдяки яким він став популярним.

Досвідчені розробники рекомендують сприймати Django як систему. Це означає, що фреймворк зазвичай використовується з великою кількістю сторонніх додатків. Їх можна вибирати в залежності від потреб конкретного проекту.

Щоб краще зрозуміти цей принцип, уявіть конструктор Lego. У ньому є багато типових блоків. В Django теж є типові блоки. Наприклад, блок авторизації або блок підписки на розсилку застосовується практично в кожному

проекті. Створені за допомогою фреймворка веб-додатки складаються з таких незалежних блоків.

Django був представлений в 2005 році. За 14 років існування він сильно змінився і удосконалився. У фреймворку постійно з'являються нові можливості, а старі удосконалюються.

Важливий момент: коли ви розбираєтеся з Django і шукаєте відповідь на конкретне запитання, найчастіше це не викликає ускладнень. Тисячі фахівців вже вирішували такі ж проблеми до вас і ділилися своїм досвідом в інтернеті. Так працює спільнота Django.

Адміністративна панель Django автоматично генерується при створенні програми. Це позбавляє розробника від необхідності створювати адмінку вручну.

За допомогою сторонніх додатків дефолтну консоль управління Django можна вдосконалити і адаптувати під потреби свого проекту. Крім того, фреймворк дозволяє налаштовувати інтерфейс дефолтной адміністративної панелі.

Написаний на Python код виходить чітким і зрозумілим навіть непідготовленим людям. Це один з факторів, завдяки яким веб-додатки на Python вважаються SEO-дружніми. Django генерує семантичні URL. Їх також називають людино-зрозумілими URL або ЛЗУ. У додатках на Django легко реалізуються іншіфункції, необхідні для пошукової оптимізації.

Функціональність Django розширюється за допомогою плагінів. Це програмні модулі, які дозволяють швидко додати на сайт потрібну функцію. В офіційному каталозі є сотні плагінів, які дозволяють легко реалізувати на сайті sitemap.xml, управляти доступами, підключити платіжну систему Stripe і так далі. При необхідності ви можете відключати або замінювати плагіни, щоб пристосувати її до поточних потреб проекту.

У популярних мовах програмування є бібліотеки, за допомогою яких зручно вирішувати спеціальні завдання. В бібліотеках можна знайти готові рішення: функції, класи, конфігурації і так далі. Завдяки таким рішенням

розширюються можливості мови, а також спрощується створення додатків.

Django підтримує використання бібліотек при розробці веб-додатків. У число популярних бібліотек входять:

– Django REST Framework, який спрощує роботу з API.

– Django CMS - зручний інструмент для управління контентом.

– Django-allauth - з його допомогою реалізуються функції реєстрації, авторизації, управління обліковими записами.

– ORM.

В Django реалізовано об'єктно-реляційне відображення (ORM), яке забезпечує взаємодію додатки з базами даних (БД). ORM автоматично передає дані з БД, наприклад, PostgreeSQL або MySQL, в об'єкти, які використовуються в коді програми.

ORM прискорює розробку прототипів і готових веб-додатків на Django. Розробнику навіть не потрібно знати мову, який використовується для взаємодії з базами даних.

Також ORM дозволяє швидко перемикатися між базами даних з мінімальними змінами коду. Наприклад, ви можете використовувати SQLite на локальному сервері, а потім переключитися на MySQL на production-сервері. Однак для мінімізації помилок краще використовувати одну базу даних під час розробки і в продакшені.

Недоліки Django: не для маленьких проектів, не завжди передбачувана поведінка.

У Django є недоліки, як у будь-якого інструменту. Основні недоліки фреймворка перераховані нижче.

Можливості Django можуть бути надлишковими для невеликого проекту. Однак в екосистемі Python є інші фреймворки, які можна використовувати в таких випадках. Наприклад, якщо ви хочете зробити простий чат, краще використовувати Flask.

2 Спеціальна частина

2.1 Опис алгоритму створення програмного засобу

Створення програмного засобу складається з трьох етапів.

Системний аналіз. У рамках цього етапу здійснюється аналізвимог, що пред'являються до програмної системи. Він проводиться на основі первинного дослідження всіх потоків інформації при традиційному проведенні робіт і здійснюється в наступній послідовності:

– уточнення видів і послідовності всіх робіт;

– визначення цілей, які повинні бути досягнуті програмою, що розробляється;

– виявлення аналогів, що забезпечують досягнення подібних цілей, їх переваг та недоліків.

– Зовнішнє специфікування. Полягає у визначенні зовнішніх специфікацій, тобто описів вхідної та вихідної інформації, форм її подання і способів обробки інформації. Реалізується у такій послідовності:

– постановка завдання на розробку нової програми;

– оцінка цілей розроблюваного програмного продукту.

Далі, при необхідності, етапи 1-2 можуть бути повторені до досягнення задовільного вигляду програмної системи з описом виконуваних нею функцій і деякої ясністю реалізації її функціонування.

Проектування програми. На цьому етапі проводиться комплекс робіт із формування опису програми. Вихідними даними для цієї фази є вимоги, викладені у специфікації, розробленої на попередньому етапі. Приймаються рішення, що стосуються способів задоволення вимогам специфікації. Цю фазу розробки програми поділяють на два етапи:

– архітектурне проектування;

– робоче проектування.

– кодування і тестування. Ці види діяльності здійснюються для окремих модулів і сукупності готових модулів до отримання готової програми.

– комплексне тестування.

– розробка експлуатаційної документації.

– прийомо-здавальні та інші види випробувань.

– коригування програм. Проводиться за результатами попередніх випробувань.

У загальному випадку існують три різні стилі кодування, які можуть застосовуватися при створенні програми WPF.

Тільки код. Це традиційний підхід, який використовується в Visual Studio для додатків Windows Forms. Інтерфейс в ньому генерується операторами коду.

Код і не скомпільована розмітка (XAML). Це спеціалізований підхід, який має сенс в певних сценаріях, коли потрібні виключно динамічні інтерфейси. При цьому частина користувацького інтерфейсу завантажується з файлу XAML під час виконання за допомогою класу XamlReader з простору імен System.Windows.Markup.

Код і скомпільована розмітка (BAML). Це кращий підхід для WPF, підтримуваний в Visual Studio. Для кожного вікна створюється шаблон XAML, і цей код XAML компілюється в BAML, після чого вбудовується в кінцеву збірку. Під час виконання скомпільований BAML витягується і використовується для регенерації призначеного для користувача інтерфейсу.

Розробка на основі тільки коду - найменш поширений (але повністю підтримуваний) шлях створення програмного забезпечення WPF без застосування будь-якого XAML-коду. Очевидним недоліком розробки на основі тільки коду є те, що цей варіант потенційно надзвичайно стомлює. Елементи управління WPF не включають параметризованих конструкторів, тому навіть додавання простої кнопки в вікно вимагає декількох рядків коду. Однією потенційною перевагою розробки на основі тільки коду є обмежений простір для налаштування.

Наприклад, можна згенерувати форму, заповнену елементами управління

введенням, на основі інформації із запису бази даних, або ж можна на основі якоїсь умови прийняти рішення, додавати або підставляти елементи управління в залежності від поточного користувача. Все, що для цього буде потрібно - це логіка перевірки умови і розгалуження. Навпаки, коли використовуються документи XAML вбудовуються в збірку як фіксовані, незмінні ресурси.

Програмний продукт — це програмне забезпечення, розроблене для вирішення задачі масового попиту та призначене для постачання користувачам. Програмний продукт відрізняється від просто програмного забезпечення максимально узагальненим набором вхідних даних, ретельним тестуванням, наявністю документації, гарантії та технічної підтримки. На відміну від програмного забезпечення, яке надається як послуга , програмні продукти зазвичай встановлюють на обладнанні користувача (власному чи орендованому). Термін ліцензії на продукт зазвичай не обмежується (хоча трапляються випадки, коли виробники обмежують час використання продукту).

При створенні й розвитку ПЗ рекомендується застосовувати такі загальносистемні принципи:

– принцип включення, який передбачає, що вимоги до створення, функціонування та розвитку ПЗ визначаються з боку більш складної системи, що включає його в себе;

– принцип системної єдності, який полягає в тому, що на всіх стадіях створення, функціонування та розвитку ПЗ його цілісність буде забезпечуватися зв'язками між підсистемами, а також функціонуванням підсистеми управління;

– принцип розвитку, який передбачає в ПЗ можливість його нарощування та вдосконалення компонентів і зв'язків між ними;

– принцип комплексності, який полягає в тому, що ПЗ забезпечує зв'язаність обробки інформації, як окремих елементів, так і для всього обсягу даних в цілому на всіх стадіях обробки;

– принцип інформаційної єдності, тобто у всіх підсистемах, засобах забезпечення і компонентах ПЗ використовуються єдині терміни, символи, умовні

позначення і способи подання;

– принцип сумісності полягає в тому, що мова, символи, коди та засоби програмного забезпечення узгоджені, забезпечують спільне функціонування всіх підсистем і зберігають відкритою структуру системи в цілому;

– принцип інваріантності визначає інваріантність підсистем і компонентів ПЗ до оброблюваної інформації, тобто вони є універсальними або типовими.

– Проектування програми. На цьому етапі проводиться комплекс робіт із формування опису програми. Вихідними даними для цієї фази є вимоги, викладені у специфікації, розробленої на попередньому етапі. Приймаються рішення, що стосуються способів задоволення вимогам специфікації. Цю фазу розробки програми поділяють на два етапи:

– архітектурне проектування. Являє собою розробку опису програми у найзагальнішому вигляді. Цей опис містить відомості про можливі варіанти структурної побудови програмного продукту (або у вигляді кількох програм, або у вигляді кількох частин однієї програми), а також про основні алгоритми, і структури даних. Результатом цієї роботи є остаточний варіант архітектури програмної системи, вимоги до структури окремих програмних компонентів і організації файлів для міжпрограмного обміну даними;

– робоче проектування. На цьому етапі архітектурний опис програми деталізується до такого рівня, який робить можливими роботи з її реалізації (кодування і збірці). Для цього здійснюється складання і перевірка специфікацій модулів, складання описів логіки модулів, складання остаточного плану реалізації програми.

Кодування і тестування. Ці види діяльності здійснюються для окремих модулів і сукупності готових модулів до отримання готової програми.

Тестування програмного забезпечення - процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, здійснюваний шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином.

Може оцінюватись:

– відповідність вимогам, якими керувалися проектувальники та розробники;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з програмним забезпеченням та операційними системами;

– відповідність задачам замовника.

Оскільки число можливих тестів навіть для нескладних програмних компонент практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення (ПЗ) тестується стандартним виконанням програми з метою виявлення баґів (помилок або інших дефектів).

Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якістьПЗ, ризики відмови, як для користувачів, так і для замовників.

Тестування може проводитись, як тільки створено виконуваний код (навіть частково завершений). Процес розробки зазвичай передбачає, коли та як буде відбуватися тестування. Наприклад, при поетапному процесі, більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються в тестових програмах. На противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно.

2.2 Опис засобів реалізації

Структура проекту зображена на рисунку 1.1.

Рисунок 1.1

– settings.py містить в собі всі налаштування проекту. Тут я реєструю додатки, задаю розміщення статичних файлів, налаштування бази даних і так далі.

– urls.py задає асоціації url адрес з уявленнями. Незважаючи на те, що цей файл може містити всі налаштування url, зазвичай його ділять на частини, по одній на додаток.

– wsgi.py використовується для налагодження зв'язку між Django додатком і веб-сервером.

– views.py проміжний файл між шаблоном сторінки та сервером, в ньому налаштовуються данні які доступні сторінці, шаблон в якому будуть ці данні застосовуватись, та обробка даних, тобто логіка додатку.

– templates папка в якій зберігаються шаблони сторінок.

– static папка в якій зберігаються статичні файли, наприклад файли стилів, фото, документи та скрипти.

Для забезпечення автоматизації та гнучкості програмного засобу потрібно зберегти велику кількість даних в базі даних. А також потрібен функціонал для ефективного звернення до бази даних та можливість редагувати таблиці.

Тому кожна таблиця бази даних має свій клас для взаємодії з нею та зберігається в файлі models.py.

Кожний стовпчик таблиці має свою змінну для звернення, а також таблиці пов’язані між собою, тому з однієї таблиці можна звернутись до пов’язаної з нею таблицею, щоб отримати необхідні данні. Також клас надає можливість сортувати данні та фільтрувати для оптимізації отримання даних, що економить час відповіді серверу.

Є один головний шаблон для головної сторінки, та один шаблон для сторінки замовлення, головні шаблони підключаються в допоміжних для того щоб не вводити однаковий код в багатьох файлах. Це дуже полегшує читання коду.

Сторінки постійно відправляють запит на оновлення, але оновлення відбувається тільки якщо відбулись зміни на сторінці, наприклад коли з’явилося нове замовлення, або статус замовлення було змінено. Це полегшує роботу з програмою.

В програмі три інтерфейси для трьох відділів компанії. Майстри, Менеджери та Менеджери запчастин. В усіх ролей різні доступи, функції та цілі.

2.3 Порівняльний аналіз реалізованого програмного засобу та програм аналогів

На даний момент, немає програмного забезпечення, за допомогою якого можна ефективно керувати процесами в такій сфері як СТО.

Проблема полягає в тому що програми та додатки створюють програмісти, а не майстри СТО. А це значить що вони не знають як зробити програмне забезпечення таким чином, що їм легше було б скористатись, ніж виконувати роботу без програми.

Один з найкращих прикладів CRM системи яку використовують в сфері СТО є Ремонлайн. Цей веб-додаток був створений спочатку тільки для сферу ремонту комп'ютерної техніки і зі своєю задачею він справляється ідеально. Потім функції Ремонлайн розширювали для застосування в інших сферах. В них це досить добре виходить, але є і проблеми. В ньому немає функцій для написання списків послуг та запчастин для СТО, а також немає функцій для прорахування вартості запчастин. А згодом Ремонлайн припинила підтримку сфери СТО, тому на покращення ситуації найближчим часом можна не очікувати. Другим прикладом є 1С. Це програмне забезпечення дуже популярне і дуже широко застосовується.

Але в ньому не вистачає інтерактивності, та немає можливості зробити зрозумілий та простий інтерфейс, а це означає що оптимізувати роботу в компанії буде не можливо. Тому ті хто використовують 1С втрачають можливість розширення кількості замовлень, та компанії в цілому відповідно.

Наступний аналог це Carbook. Ця система створюється саму для сфери ремонту автомобілів. Вони втілили можливість написання та прорахування запчастин для замовлення, але вони ще в стадії розробки, та в них не вистачає функцій для ретельнішого контролю замовлення.

Найкращим додатком для ремонту авто є СТО CRM, він мають майже всі необхідні функції для цієї сфери, окрім агрегатного ремонту. Головними мінусами

цієї системи є те, що це російська програма, працюють вони тільки з російським ринком, а також в них слабка технічна підтримка.

Є проблеми які стосуються всіх подібних програм. Наприклад, агрегатний ремонт жодна програма не підтримує, немає автоматизації при заповненні списку запчастин. Програма має бути максимально простою в користуванні та ефективною. Тому що майстрам немає часу вчитись користуватись складними програмами, їм потрібно робити діагностику і ремонт, а написання списків має бути максимально оптимізовано.

Подібних проблем немає тільки м монобрендових СТО, тому що їм програмне забезпеченні надає компанія яка виготовляє авто, так як вони в цьому зацікавлені в першу чергу.

Переваги мого веб-додатку в тому, що компанія в будь який момент може до мене звернутись за допрацюванням додатку. Не потрібно залежати від інших компаній та чекати коли вони покращать свою систему. Інтуїтивно зрозумілий та простий процес написання списку запчастин та прорахування вартості замовлення.

2.4 Інтсрукція роботи користувача

Зайшовши на по адресі головної сторінки користувач має авторизуватись, інакше він нічого не зможе робити. Потім додаток визначає до якої групи належить користувач та виводить відповідний інтерфейс в якому видно всі замовлення доступні конкретній групі.

Якщо користувач належить до групи майстрів, він має змови обрати доступне замовлення над яким працює та додати послуги, або запчастини до замовлення. Також користувач має змогу додати нове замовлення та заповнити список робіт і запчастин.

Якщо користувач належить до групи менеджери запчастин, він має змогу прораховувати запчастини до замовлень в яких написаний список запчастин,

відмічати які запчастини замовлені, та доставлені якщо в замовлення статус замовлення запчастин, або очікуються запчастини.

Якщо ж користувач належить до групи менеджерів, то він має можливості попередніх двох груп, а також повний контроль етапів замовлення, надсилання замовлення на допрацювання, або на прорахування, на замовлення запчастин, та контролювати все. Менеджер може навіть замінити одного з працівників.

Суперкористувач має адміністративну панель, в якій може керувати всіма таблицями бази даних, користувачами, групами та доступом.

2.5 Тестування роботи програмного модуля

Тесту веб-додатку відбувається в компанії СТО Турбо Дизель Сервіс. В адміністративній панелі було створено 5 користувачів, три менеджери, один менеджер запчастин, один акаунт на всіх майстів, так як в них три комп'ютери на 10 працівників.

Тестування проводиться на реальних замовлення. Інтерфейс написання списку показава себе дуже добре, неочікуваних проблем не виникло. В перші дні було вирішено додати три додаткові поля для замовлення, номер форсунки, номер турбіни, номер насосу. Це необхідно для того, щоб точно підібрати запчастини. Серед інших порад майстрів було лише додавання нових запчастин, послуг та вз'язків між ними. Тому я надав доступ до адмін панелі керівництву компанії, щоб вони змогли самі додавати такі дані.

Тестування інтерфейсу менеджерів теж виявилось позитивним. Прорахування вартості робіт та запчастин працює чудово, навіть якщо потрібно змінити ціни, або кількість полсуг це можна відразу зробити з автоматичним збереженням.

Важливим допрацюванням стало поліпшення переходім між статусами, щоб менеджер мав змогу відправляти замовлення у потрібний момент іншій групі користувачів.

В інтерфейсі менеджера запчастин була виявленя потребність залишити

обов’язковим тільки поле ціна запчастини, а також я додав ціну за замовчуванням, так як є запчастини на складі компанії ціни яких точно відомі.

З важливих допрацювань було зазначено, можливість ставити коментарі які не можна змінювати до замовлення. Промто список коментарів до замовлення від різних працівників які працювали над конкретним замовленням.

Також важливим пунком є заборона видаляти запчатсини які були погоджені. Щоб в майбутньому менеджери не могли шахраювати з замовленням.

Найближчим часом буде введено ще одне важливе оновлення. Це створення технічного завдання майстру після того як менеджер погодить роботи. Покищо керівництво Турбо Дизель Сервіс не знає як саме має виглядати це технічне завдання, тому цей пункт на обміркуванні.

Загалом програма продемонструвала хорошу та надійну роботу в реальних умовах. Від друкування листів вже відмовились. А менеджер запчастин припинив вести блокнот запчастин, тепер в нього все в комп’ютері.

3 Охорона праці

3.1 Вимоги безпеки при користуванні ПК

До самостійної роботи на комп’ютерах допускаються особи, які пройшли медичний огляд, навчання по професії, вступний інструктаж з охорони праці та первинний інструктаж з охорони праці на робочому місці. В подальшому вони проходять повторні інструктажі з охорони праці на робочому місці один раз на півріччя, періодичні медичні огляди один раз на два роки.

Основним обладнанням робочого місця користувача комп’ютера є монітор, системний блок та клавіатура.

Робочі місця повинні бути розташовані на відстані не менше 1,5 м від стіни з вікнами, від інших стін на відстані 1м, між собою на відстані не менше 1,5 м. Відносно вікон робоче місце доцільно розташовувати таким чином, щоб природнесвітло падало на нього збоку, переважно зліва.

Робочі місця необхіднорошташовувати так, щоб уникнути попадання в очі прямого світла. Джерела освітлення рекомендується розташовувати з обох боків екрану паралельно напрямку погляду. Для уникнення світлових відблисків екрану, клавіатури в напрямку очей користувача, від світильників загального освітлення або сонячних променів, необхідно використовувати антибликові сітки, спеціальні фільтри для екранів, захисні козирки, на вікнах – жалюзі.

Фільтри з металевої або нейлонової сітки використовувати не рекомендується, тому що сітка спотворює зображення через інтерференцію світла.

Найкращу якість зображення забезпечують скляні поляризаційні фільтри. Вони усувають практично всі відблиски, роблять зображення чітким і контрастним.

При роботі з текстовою інформацією (в режимі введення даних та редагування тексту, читання з екрану) найбільш фізіологічним правильним є зображення чорних знаків на світлому (чорному) фоні.

Монітор повинен бути розташований на робочому місці так, щоб поверхня екрана знаходилася в центрі поля зору на відстані від 400 мм до 700 мм від очей користувача. Рекомендується розміщувати елементи робочого місця так, щоб витримувалася однакова відстань очей від екрана, клавіатури, тексту.

Зручна робоча поза при роботі з комп'ютером забезпечується регулюванням висоти робочого столу, крісла та підставки для ніг. Раціональною робочою позою може вважатися таке положення, при якому ступні працівника розташовані горизонтально на підлозі або підставці для ніг, стегна зорієнтовані у горизонтальній площині, верхні частини рук – вертикальні. Кут ліктєвого суглоба коливається в межах від 70° до 90°, зап'ястя зігнуті під кутом не більше ніж 20°, нахил голови від 15° до 20°.

Для нейтралізації зарядів статичної електрики в приміщенні, де виконується робота на комп'ютерах, в тому числі на лазерних та світлодіодних принтерах, рекомендується збільшувати вологість повітря за допомогою кімнатних зволожувачів. Не рекомендується носити одяг з синтетичних матеріалів.

Згідно статті 18 Закону України "Про охорону праці" користувач ПК зобов'язаний:

– знати і виконувати вимоги нормативних актів про охорону праці, правила поводження з устаткуванням та іншими засобами виробництва, користуватися засобами колективного та індивідуального захисту;

– дотримуватись зобов'язань щодо охорони праці, передбачених колективним договором та правилами внутрішнього трудового розпорядку підприємства;

– співробітничати з власником у справі організації безпечних і нешкідливих умов праці, особисто вживати посильних заходів щодо усунення

будь-якої виробничої ситуації, яка створює загрозу його життю чи здоров'ю, або людей, які його оточують, повідомляти про небезпеку свого безпосереднього керівника або іншу посадову особу.

Перед початком роботи користувачеві необхідно:

– увімкнути систему кондиціювання в приміщенні;

– перевірити надійність встановлення апаратури на робочому столі. Повернути монітор так, щоб було зручно дивитися на екран – під прямим кутом (а не збоку) і трохи зверху вниз, при цьому екран має бути трохи нахиленим, нижній його край ближче до оператора;

– перевірити загальний стан апаратури, перевірити справність електропроводки, з'єднувальних шнурів, штепсельних вилок, розеток, заземлення захисного екрана;

– відрегулювати освітленість робочого місця;

– відрегулювати та зафіксувати висоту крісла, зручний для користувача нахил його спинки;

– приєднати до системного блоку необхідну апаратуру. Усі кабелі, що з'єднують системний блок з іншими пристроями,необхідновставляти та виймати при вимкненому комп'ютері;

– ввімкнути апаратуру комп'ютера вимикачами на корпусах в послідовності: монітор, системний блок, принтер (якщо передбачається друкування);

– відрегулювати яскравість свічення монітора, мінімальний розмір світної точки, фокусування, контрастність. Ненеобхіднороботи зображення надто яскравим, щоб не втомлювати очей.

Під час виконання роботи:

– необхідно стійко розташовувати клавіатуру на робочому столі, не опускати її хитання. Під час роботи на клавіатурі сидіти прямо, не напружуватися;

– для забезпечення несприятливого впливу на користувача пристроїв типу "миша" належить забезпечувати вільну велику поверхню столу для

переміщення ”миші” і зручного упору ліктьового суглоба;

– не дозволяються сторонні розмови, подразнюючі шуми;

– періодично при вимкненому комп’ютері прибирати ледь змоченою мильним розчином бавовняною ганчіркою пил з поверхонь апаратури. Екран ВДТ та захисний екран протирають ганчіркою, змоченою у спирті. Не дозволяється використовувати рідинні або аерозольні засоби чищення поверхонь комп’ютера.

При користуванні ПК забороняється:

– самостійно ремонтувати апаратуру. Ремонт апаратури здійснюється спеціалістами з технічного обслуговування комп’ютера, які 1 раз на півроку повинні відкривати процесор і вилучати пилососом пил і бруд, що накопичилися;

– класти будь-які предмети на апаратуру комп'ютера;

– закривати будь-чим вентиляційні отвори апаратури, що може призвести до її перегрівання і виходу з ладу.

Для зняття статичної електики рекомендується час від часу доторкатися до металевих поверхонь.

Після закінчення роботи з ПК необхідно:

– закінчити та записати у пам'ять комп'ютера файл, що знаходиться в роботі;

– вимкнути принтер та інші периферійні пристрої. Штепсельні вилки витягнути з розеток. Накрити клавіатуру кришкою для запобігання попаданню в неї пилу;

– прибрати робоче місце;

– ретельно вимити руки теплою водою з милом;

– вимкнути кондиціонер, освітлення і загальне електроживлення;

– пройти в спеціально обладнаному приміщенні сеанс психофізіологічного розвантаження і зняття втоми з виконанням спеціальних вправ аутогенного тренування.

3.2 Заходи та засоби протипожежного захисту

Можливість виникнення пожежі на виробництві (в побуті) оцінюється її ймовірністю, яка залежить від можливості виникнення умов, необхідних і достатніх для виникнення загоряння.

Будь-яка пожежа не трапляється сама по собі, а викликана конкретними умовами. Одночасна поява у просторі трьох факторів – горючої речовини, окислювача та джерела запалювання — може при певних кількісних і якісних співвідношеннях спричинити виникнення та розвиток пожежі. Пожежа - це процес горіння, який не контролюється.

Складність та різноманітність завдань, пов'язаних із забезпеченням протипожежного захисту та попередження і гасінням пожеж, викликають необхідність безпосередньої участі в цьому процесі всіх державних, господарських, комерційних та громадських організацій. В залежності від призначення та функцій ці організації наділяються певними повноваженнями і є окремими елементами загальної системи протипожежного захисту держави.

За пожежну безпеку підприємств, установ та організацій відповідають їх власники й уповноважені ними органи (далі — власники), а також орендарі. Вони зобов'язані:

– розробляти комплексні заходи по забезпеченню пожежної безпеки, впроваджувати досягнення науки й техніки, позитивний досвід;

– відповідно до нормативних актів з пожежної безпеки розробляти й затверджувати положення, інструкції, інші нормативні акти, що діють у межах підприємства, установи та організації, здійснювати постійний контроль за їх дотриманням;

– забезпечити дотримання протипожежних вимог стандартів, норм, правил, а також виконання вимог, розпоряджень і постанов органів державного пожежного нагляду; організовувати навчання працівників правил пожежної безпеки і пропаганду заходів для їхнього забезпечення;

– за відсутності в нормативних актах вимог, необхідних для забезпечення пожежної безпеки, вживати відповідних заходів, погоджуючи їх з органами державного пожежного нагляду;

– тримати в справному стані засоби протипожежного захисту і зв'язку, пожежну техніку, устаткування та інвентар, не допускати їх використання не за призначенням;

– створювати, в разі потреби, відповідно до встановленого порядку підрозділи пожежної охорони і необхідну для їх функціонування матеріально-технічну базу;

– надавати на вимогу державної пожежної охорони відомості і документи про стан пожежної безпеки об'єктів і виробленої ними продукції;

– здійснювати заходи по впровадженню автоматичних засобів виявлення й гасіння пожеж і використанню з цією метою виробничої автоматики;

– вчасно інформувати пожежну охорону про несправність пожежної техніки, систем протипожежного захисту, водопостачання, а також про закриття доріг і проїздів на своїй території;

– проводити службове розслідування випадків пожеж.

На кожному підприємстві роботодавець повинен забезпечити пожежну безпеку, розробивши систему протипожежного захисту.

До системи протипожежного захисту належить:

– забезпечення необхідної вогнестійкості будівель та споруд;

– забезпечення своєчасної евакуації людей та відповідність нормативам шляхів евакуації;

– створення умов для ефективного гасіння пожежі та обмеження її негативних наслідків.

Для відвернення пожеж розробляють організаційно-технічні, режимні, пожежно-евакуаційні, тактико-профілактичні, будівельно-конструктивні та інші заходи режимів експлуатації машин і обладнання, за яких повністю виключається можливість виникнення іскор та полум'я чи контакт нагрітих деталей обладнанняз горючим середовищем.

Комплекс організаційно-технічних заходів повинен включати: організацію пожежної охорони; паспортизацію технологічних процесів, будівель та споруд об'єктів щодо забезпечення пожежної безпеки; організацію навчання працюючих правилам пожежної безпеки; розробку інструкцій про дотримання протипожежного режиму та порядок дій людей у разі пожежі; встановлення порядку зберігання пожежонебезпечних речовин; облік та аналіз даних про пожежі і збитки від них та аналіз витрат на забезпечення пожежної безпеки.

Обмеження поширення пожежі здійснюється шляхом улаштування: протипожежних розривів між будівлями і спорудами; протипожежних перешкод; аварійного відключення комунікацій; обмежувачів розливу і розтікання рідин; гранично допустимих за технічними нормами площ протипожежних відсіків, секцій, а також кількості поверхів будівель та споруд; локалізації пожеж відповідними вогнегасними речовинами.

На підприємствах для гасіння пожеж застосовуються засоби пожежогасіння: автоматичні газові і порошкові установки, установки автоматичної пожежної сигналізації, протипожежний водогін та пожежні крани на внутрішніх протипожежних водогонах. Будівлі, споруди, приміщення, технологічні установки також повинні бути забезпечені первинними засобами пожежогасіння: вогнегасниками, ящиками з піском, бочками з водою, покривалами з негорючого теплоізоляційного полотна, пожежними відрами, лопатами, пожежним інструментом (гаками, ломами, сокирами тощо), які використовуються для локалізації і ліквідації пожеж у їх початковій стадії розвитку за нормами.

3.3 Створення оптимальних умов освітлення робочого місця програміста

Правильно спроектоване і виконане виробниче освітлення покращує умови зорової роботи, знижує стомлюваність, сприяє підвищенню продуктивності праці, благотворно впливає на виробниче середовище, надаючи

позитивну психологічну дію на працюючого, підвищує безпеку праці і знижує травматизм.

Недостатність освітлення приводить до напруги зору, ослабляє увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає засліплення, роздратування і різь в очах. Неправильний напрямок світла на робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працюючого. Всі ці причини можуть призвести до нещасного випадку або профзахворювань, тому такий важливий правильний розрахунок освітленості.

Існує три види освітлення - природне, штучне і поєднане (природне і штучне разом).

Природне освітлення - освітлення приміщень денним світлом, що потрапляє через світлові прорізи в зовнішніх огороджуючих конструкціях приміщення.Природне освітлення характеризується тим, що змінюється в широких межах залежно від часу дня, пори року, характеру області і ряду інших чинників.

Штучне освітлення застосовується при роботі в темний час доби і вдень, коли не вдається забезпечити нормовані значення коефіцієнта природного освітлення (похмура погода, короткий світловий день). Освітлення, при якому недостатнє за нормами природне освітлення доповнюється штучним, називається змішаним освітленням.

Штучне освітлення підрозділяється на робоче, аварійне, евакуаційне, охоронне. Робоче освітлення, у свою чергу, може бути загальним або комбінованим. Загальне - освітлення, при якому світильники розміщуються у верхній зоні приміщення рівномірно, або, як розташоване устаткування. Комбіноване - освітлення, при якому до загального додається місцеве освітлення.

Згідно нормативним документам з охорони праці в приміщень обчислювальних центрів необхідно застосувати систему комбінованого освітлення.

При виконанні робіт категорії високої зорової точності (найменший

розмір об'єкту розрізнення від 0,3 до 0,5 мм) величина коефіцієнта природного освітлення (КЕО) повинна бути не нижче 1,5%, а при зоровій роботі середньої точності (найменший розмір об'єкту розрізнення від 0,5 до 1,0 мм) КЕО повинен бути не нижче 1,0%. В якості джерел штучного освітлення звичайно використовуються люмінесцентні лампи типа ЛБ, або ДРЛ, які попарно об'єднуються в світильники, які повинні розташовуватися рівномірно над робочими поверхнями.

Вимоги до освітленості в приміщеннях, де встановлені комп'ютери, наступні: при виконанні зорових робіт високої точності загальна освітленість повинна складати 300лк, а комбінована - 750лк; аналогічні вимоги при виконанні робіт середньої точності - 200 і 300лк відповідно.

Крім того все поле зору повинне бути освітлено достатньо рівномірно - ця основна гігієнічна вимога. Іншими словами, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими, оскільки яскраве світло в районі периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності.

Висновок

Розроблений програмний продукт збільшив швидкість виконання замовлення на СТО, за рахунок чого збільшив продуктивність компанії. Також додаток зменшив вплив людського фактору на ведення замовлення, так як майстри перестали забувати обов’язкові роботи та запчастини, а значить і середній чек збільшився.

На даний момент програмний засіб використовується в Турбо Дизель Сервісі, звідки я отримую інформацію про те як працює додаток та як його покращити. Майстри вже значно розширили список послуг, запчастин, покращили зв’язки між ними.

В перспективі мій додаток може стати повнофункціональною CRM системою, або модулем для вже існуючих систем.

Для збільшення функціоналу програми, вже намічені найближчі оновлення. Наприклад, парсер який сам шукає запчастини в постачальників, а менеджеру запчастин залишається тільки обрати кілька хороших варіантів.

Покращення функцій для написання списку, додаткові можливості для менеджера запчастин, обмеження прав для деяких статус замовлення.

– Збільшення автоматизації програми, для максимального спрощення роботи. Переваги мого програмного засобу в тому що він вузькоспеціалізований, а значить ідеально підходить для СТО. Інтерфейс програми дуже простий та зрозумілий, працівники СТО швидко навчились орієнтуватися в ньому.

Тепер немає необхідності розбирати почерк майстрів, та носити папірці. В кілька кліків можна зробити всю монотонну роботу.

Перелік використаних джерел

1. ГОСТ 19.701–90. ЄСПД. Схеми алгоритмів, програм, даних і систем [Текст]. - Замість ГОСТ 19.002–80, ГОСТ 19.003–80 ; введ. 1992-01-01. – М.: Государственный комитет СССР по управлению качеством продукции и стандартам.

2. ДСТУ ГОСТ 7.1:2006. Система стандартів з інформації, бібліотечної та видавничої справи. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання [Текст]. – Замість ГОСТ 7.1-84, ГОСТ 7.16-79, ГОСТ 7.18-79, ГОСТ 7.34-81, ГОСТ 7.40-82 ; введ. 2007-07-01. – К.: Держкомітет України з питань технічного регулювання і споживчої політики.

3. ДСТУ 3582–97. Інформація та документація. Скорочення слів в українській мові у бібліографічному описі. Загальні вимоги та правила [Текст]. – Замість РСТ УРСР 1743-82 ; введ. 1997-07-04. – К.: Держстандарт України.

4. ГОСТ 7.82-2001. Бібліографічний запис / Бібліографічний опис електронних ресурсів [Текст]. – Введ. 2001-05-22. – М.: Межгос. совет по стандартизации, метрологии и сертификации.

5. Безпека праці: ергономічні та естетичні основи: Навчальний посібник / С. Апостолюк, В.С. Джигирей. А.В . Апостолюк та ін. - К.: Знання, 2006. - 215 с.

6. Гогіташвілі Г.Г., Лапін В.М. Основи охорони праці: Навч. посіб. – 4-е вид., випр.. і доп. – К.: Знання, 2008. - 302с.

7. Гігієна праці та виробнича санітарія: (Навч.-метод. посібник) / І.М. Трахтенберг, М.М. Коршун, О.В. Чебанова; За ред. І.М. Трахтенберга. - К.; 1997. -464 с.

8. ГандзюкМ.П., Желібо Є.П., Халімовський М.О. Основи Охорони праці: Підруч. для студ. вищих навч. закл. За ред. М.П. Гандзюка. - К.: Каравела; Львів: Новий Світ-2000, 2003. - 408 с.

9. Желібо Є П., Заверуха Н. М., Зацарний В, В. Безпека життедіяльності /За ред. Є П. Желібо. - К.: Каравела, 2010. - 328 с.

Додаток А – Код файлу представлення

```
        from django.shortcuts import
render, redirectfrom django.views import
View
        from django.views.generic import DetailView
        from django.http import HttpResponse, HttpResponseRedirect,
JsonResponse,
        HttpResponseForbidden
        from django.contrib.auth.models import
Group, Userfrom .forms import AddOrdersForm
        from django.shortcuts import get_object_or_404


        # Create your views here.
        from .models import Options, Groups, Services, Products,
                Connections,ThroughProducts, KitsTable, BigKits,
        BigKitsTable, Orders, OrderStatuses,
        OrderService, OrderProduct, UsersMin, PropertiesOrderProduct


        def index(request):
            orders = Orders.objects.exclude(order_status__id=4)
            orders_manager
    =
Orders.objects.exclude(order_status__id=4).filter(order_status
```

```
    _____user_class=2) orders_products
Orders.objects.exclude(order_status__id=4).filter(order_status
    _____user_class=3) orders_meister
Orders.objects.exclude(order_status__id=4).filter(order_status
    _____user_class=1)manager_products =
    Group.objects.get(id=3).user_set.all()
    meisters =
    Group.objects.get(id=1).user_set.all()
    managers = User.objects.filter(groups
    _____id=2
    ) template = 'index.html'
            context = {'orders': orders, 'managers': managers, }
            if request.user.is_authenticated:

        if request.user in manager_products:
            template = 'desk-products-
            manager.html'
        elif request.user in meisters:
            template = 'index-
            meister.html'
                else:
            context = {'managers': managers, 'orders_manager':
                    orders_manager,'orders_products': orders_products,
'orders_meister': orders_meister,}
        return
            render(
            request
            ,
            templat
            e,
                    context = context,
```

```
            )
        else:
            return HttpResponseRedirect('/accounts/login/')


    def Archive(request):
        if request.user.is_authenticated:
            orders =
    Orders.objects.filter(order_status_____id=4)
    managers = User.objects.filter(groups__id=2)
    template = 'archive.html'
            context = {'managers': managers,
    'orders': orders,}return render(
                r
      eque
      st,
      temp
      late,
            context = context,
            )
        else:
            return HttpResponseRedirect('/accounts/login/')
    def add_order(request):
if not
    request.user.is_authenticated
    :return redirect('index')
        error = ''
```

```
        if request.method == 'POST':
            form =
    AddOrdersForm(request.POST)if
    form.is_valid():
                test =
        Orders.objects.filter(order_code=form.cleaned_data['order_code'])if
        not test:
                    firma = form.save()
            return redirect('edit-order',
        firma.pk)else:
                    error = 'Замовлення вже снує'
            else:
                error = 'Неправильно заповнена форма'


        form = AddOrdersForm()
        managers = User.objects.filter(groups___id=2)


return
    render(
    request
    ,
            'pages/add-order.html',
            context={'managers': managers, 'form': form, 'error': error},
        )
```

```
class
    GetOrderDetail(View)
    :@staticmethod
            def get(request, *args, **kwargs):
                impo
        rt requests
        import json
                order =
        str(request.GET.get('order'))data
        = {
                    'api_key': '03e69ae3a04c4b5f84af3862ae21e6af'
                }

                # Авторизация

                res = requests.post('https://api.remonline.ua/token/new',
        data=data)data = json.loads(res.text)
                token =
        data['token']
        params = (
                    ('token', token),
                    ('sort_dir', 'desc'),

                )

                sklad = requests.get('https://api.remonline.ua/order/',
        params=params)data = json.loads(sklad.text)
                #початок циклу отримання
        замовленьfrom multiprocessing
        import Pool
        page =
        1 while
        True:
```

```
        params = (
            ('token',
            token),
                    ('sort_
            dir', 'desc'),
            ('page', page)
                )
                goods = requests.get('https://api.remonline.ua/order/',
        params=params)data = json.loads(goods.text)
                for i in data['data']:
                    if i['id_label'] == order:

                        manager = i['manager_id']  # 58 sanya 59 dima
                if manager ==
                    73658:manager
                    = 3
                elif manager ==
                    73659:manager
                    = 5
                        else:
                            manager = 6

                        m_name =
                User.objects.get(id=manager)
                m_name = m_name.first_name
                        car = i['custom_fields']['f386134'] + ' ' +
                i['custom_fields']['f386133']vin = i['custom_fields']['f386138']
                        payment = i['order_type']['name']
                return JsonResponse({'manager': manager, 'm_name': m_name,
'car':car, 'vin': vin, 'payment': payment})
                page += 1
```

```
            if (len(i) <
                50):
                break
                  # кінець циклу отримання
        замовлень return JsonResponse({'data':
        'не знайдено'})


class
    UpdateOrder(View)
    :@staticmethod
    def get(request, *args, **kwargs):
        code =
        str(request.GET.get('order'))
                id_manager =
        request.GET.get('manager')car =
        request.GET.get('car')
                vin = request.GET.get('vin')
                payment =
        request.GET.get('payment')key =
        request.GET.get('key')
                value = request.GET.get('value')
                order = Orders.objects.get(order_code=code)



                if key != None:

            if value ==
                'non':value
                = ''
            if key == 'turbine_code':
                order.turbine_code =
```

```
            value
        elif key ==
            'pump_code':
            order.pump_code =
            value
        elif key ==
            'nozzle_code':
            order.nozzle_code =
            value



            if id_manager != None:
        manager =
        User.objects.get(id=id_manager)
        order.manager = manager

    if car != None:
        order.car =
        car
    if vin != None:
        order.vin_code =
        vin
    if payment != None:
        order.payment = payment
            order.save()
            return JsonResponse({'data': 'оновлено', 'value': value})


class
```

```
OrderDetail(DetailView)
:model = Orders
        template_name = 'pages/view-
order.html'context_object_name =
'order'
        def get_context_data(self, **kwargs):
            context = super(OrderDetail,
    self).get_context_data(**kwargs) context['services'] =
    OrderService.objects.filter(order=self.kwargs['pk'])
    context['products'] =
    OrderProduct.objects.filter(order=self.kwargs['pk'])order =
    Orders.objects.get(id=self.kwargs['pk'])
            context['btn_next'] = False
    if order.order_status.id != 1 and order.order_status.id != 5 andorde
                context['btn_next'] = True
            product_manager =
    Group.objects.get(id=3).user_set.all()if
    self.request.user in product_manager:
       context['product_manager'] =
    Trueelse:
       context['product_manager'] =

    Falsereturn context

        def get_template_names(self, **kwargs):
    meister =
    Group.objects.get(id=1).user_set.all()order
    = Orders.objects.get(id=self.kwargs['pk'])
```

```
                if self.request.user in meister:
            return ['pages/meister-view-
        order.html']else:
            if order.order_status.id >= 6:
                return ['pages/check-
                order.html']
                    else:
                        return ['pages/view-order.html']
    def get(self, request, *args,
        **kwargs):self.object =
        self.get_object()
        if not
            self.request.user.is_authenticated
            :return redirect('index')
                else:
                    context =
            self.get_context_data(object=self.object)return
            self.render_to_response(context)

class
    OrderEditDetail(DetailView)
    :model = Orders
            template_name = 'pages/edit-
    order.html'context_object_name =
    'order'

            def get_context_data(self, **kwargs):
                context = super(OrderEditDetail,
        self).get_context_data(**kwargs)context['big_kits'] =
```

```
        BigKits.objects.all()
                context['services'] = Services.objects.all()
        context['products'] = Products.objects.all()
        context['masters'] = UsersMin.objects.all()
                context['services_saved']


        =

          OrderService.objects.filter(order=self.kwargs['pk'])

                context['products_saved']


        =

OrderProduct.objects.filter(order=self.kwargs['p
        k'])managers = User.objects.filter(groups
        _____id
        =2) if self.request.user in managers:
            context['permis'] =
        Trueelse:
            context['permis'] =
        Falsereturn context
    def get(self, request, *args,
        **kwargs):self.object =
        self.get_object()
        if not
            self.request.user.is_authenticated
            :return redirect('index')
                else:
                    context =
            self.get_context_data(object=self.object)return
            self.render_to_response(context)
```

```
class
    PropetriesProdOrd(DetailView)
    :model = Orders
            template_name = 'pages/properties-order-
    product.html'context_object_name = 'order'

            def get_context_data(self, **kwargs):

                context = super(PropetriesProdOrd,
        self).get_context_data(**kwargs) context['products'] =
        OrderProduct.objects.filter(order=self.kwargs['pk'])
        context['products_properties']                                    =
PropertiesOrderProduct.objects.filter(product
        _____order=self.kwargs['pk'
        ])managers = User.objects.filter(groups_id=2)
        if self.request.user in
            managers:
            context['permis'] = True
                else:
            context['permis'] =
        Falsereturn context

            def get_template_names(self, **kwargs):

                order = Orders.objects.get(id=self.kwargs['pk'])


        if order.order_status.id == 9 or order.order_status.id
            == 7 :return ['pages/check-properties-order.html']
                else:
```

```
                return ['pages/properties-order-product.html']


    def get(self, request, *args,
        **kwargs):self.object =
        self.get_object()
        if not
            self.request.user.is_authenticated
            :return redirect('index')
                else:
                    context =
            self.get_context_data(object=self.object)return
            self.render_to_response(context)

class
    RemoveProperty(View)
    :@staticmethod
    def get(request, *args,
        **kwargs):id =
        request.GET.get('id')
                PropertiesOrderProduct.objects.get(id=i
        d).delete()return JsonResponse({'data':
        'Видалено', 'id': id})

class
    RemoveItemOrder(View)
    :@staticmethod
    def get(request, *args,
        **kwargs):id =
        request.GET.get('id')
```

```
            table =
    request.GET.get('table')if
    table == 'table-services':
       OrderService.objects.get(id=id).del
    ete()else:
                OrderProduct.objects.get(id=id).delete()

            return JsonResponse({'data': 'Видалено', 'id': id})


      class AddUpdateProperty(View):
         @staticmethod

def get(request, *args,
    **kwargs):id =
    request.GET.get('id')
            product =
    request.GET.get('product')sku =
    request.GET.get('sku')
            brand =
    request.GET.get('brand')
    price =
    request.GET.get('price') url
    =  request.GET.get('url')
    value =
    request.GET.get('value')
            comment =
    request.GET.get('comment')confirm
    = request.GET.get('confirm')
    ordered =
    request.GET.get('ordered') received
```

```
= request.GET.get('received') if id
!= 'non':
            row =
    PropertiesOrderProduct.objects.get(id=id)if
    sku:
        row.sku =
    skuif brand:
        row.brand =
    brandif price:
        row.price =
    priceif url is
    not None:
        row.url
    = urlif
    confirm:
        if confirm == 'true':
            row.select = True
                else:
            row.select =
    False if comment is
    not None:
                row.comment_product_manager = comment
             if ordered:
        if ordered == 'true':
            row.ordered = True
                else:
            row.ordered =
    Falseif received:
        if received == 'true':
            row.received = True
```

```
                else:
            row.received =
        Falserow.save()
                product =
        OrderProduct.objects.get(id=row.product.id)if
        value:
            product.value =
        valueproduct.save()
        return JsonResponse({'data': 'Оновлено', 'id':
    id})else:
                product_id =
        OrderProduct.objects.get(id=product)if price:
            if url ==
                None:url
                = ''
            if comment ==
                None:comment
                = ''
            if brand ==
                None:brand
                = ''
            if sku ==
                None:sku
                = ''
            table = PropertiesOrderProduct.objects.create(product=product_id,
sku=sku,                brand=brand,                price=price,
                        url=url,comment_product_manager=comment)
                if value:
            product_id.value =
```

```
    valueproduct_id.save()
    return JsonResponse({'data': 'Додано', 'id':
id})else:
            if value:
        product_id.value =
    valueproduct_id.save()
            return JsonResponse({'data': 'Оновлено продукт', 'id':
    product})
```