Отлично, Юрий, рад помочь! Вот полный и детально продуманный план реализации проекта на основе проведенного анализа. Я постарался учесть все аспекты для улучшения качества и функциональности вашего приложения. План включает в себя все этапы модернизации с подробными шагами и рекомендациями. Можете спокойно заниматься своими делами, а когда будете готовы, начнем реализацию.

# План реализации улучшений для проекта MaxMontaj67

## Этап 1: Исправление текущих ошибок

### 1.1. Исправление ошибки с полем `customer\_name`

- \*\*Описание проблемы:\*\* При создании нового заказа возникает ошибка из-за отсутствия значения по умолчанию для поля `customer\_name` в таблице `orders`.

- \*\*Решение:\*\*

- Добавить логику заполнения поля `customer\_name` при создании заказа.

- Изменить структуру базы данных, чтобы поле могло принимать значение по умолчанию.

- Проверить все места, где создаются или обновляются заказы, и убедиться, что поле `customer\_name` корректно заполняется.

### 1.2. Исправление ошибки "Unsupported Media Type"

- \*\*Описание проблемы:\*\* При создании заказа форма отправляется с типом `application/x-www-form-urlencoded`, тогда как сервер ожидает `application/json`.

- \*\*Решение:\*\*

- Изменить обработчик маршрута `/api/orders` на прием данных из `request.form`.

- Убедиться, что формы HTML используют правильный метод отправки данных.

### 1.3. Динамическое обновление страниц заказов

- \*\*Проблема:\*\* Статические данные в шаблонах `my\_orders.html` и `installer\_dashboard.html`.

- \*\*Решение:\*\*

- Подключить базу данных и динамически отображать информацию о заказах, используя Jinja2 для генерации HTML.

- Добавить маршруты для получения данных из БД и передачи их в шаблоны.

## Этап 2: Улучшение кода и оптимизация логики

### 2.1. Использование ORM

- \*\*Задача:\*\* Перевести работу с базой данных на ORM (например, SQLAlchemy).

- \*\*Преимущества:\*\*

- Повышение читаемости кода.

- Упрощение работы с транзакциями и запросами.

- \*\*Шаги реализации:\*\*

- Установить и настроить SQLAlchemy в проекте.

- Переписать существующие запросы на ORM, протестировать их работу.

### 2.2. Разделение логики на модули

- \*\*Задача:\*\* Разделить бизнес-логику, маршруты и шаблоны на отдельные модули для повышения структуры проекта.

- \*\*Шаги реализации:\*\*

- Создать отдельные файлы для обработки логики, взаимодействия с базой данных и маршрутов.

- Настроить импорт и взаимодействие между модулями.

### 2.3. Валидация данных на сервере и клиенте

- \*\*Задача:\*\* Реализовать валидацию входных данных для предотвращения ошибок и некорректного ввода.

- \*\*Шаги реализации:\*\*

- Добавить валидацию в формах на стороне клиента (HTML/JavaScript).

- Реализовать валидацию на сервере с помощью библиотек, например, `WTForms`.

## Этап 3: Добавление функциональности и безопасности

### 3.1. Аутентификация и авторизация

- \*\*Задача:\*\* Реализовать механизм аутентификации и авторизации для разграничения доступа к функционалу.

- \*\*Шаги реализации:\*\*

- Добавить систему регистрации и входа пользователей с использованием Flask-Login.

- Разграничить доступ к страницам в зависимости от ролей (администратор, диспетчер, монтажник, клиент).

### 3.2. Логирование ошибок и действий

- \*\*Задача:\*\* Внедрить систему логирования для отслеживания ошибок и действий пользователей.

- \*\*Шаги реализации:\*\*

- Использовать библиотеку `logging` для записи ошибок и действий в файл логов.

- Настроить уровень логирования (информация, предупреждения, ошибки).

### 3.3. Реализация тестирования

- \*\*Задача:\*\* Написать тесты для основных компонентов приложения.

- \*\*Шаги реализации:\*\*

- Использовать `unittest` или `pytest` для создания тестов.

- Написать тесты для маршрутов, проверки работы с базой данных и ключевой бизнес-логики.

## Этап 4: Оптимизация пользовательского интерфейса (UI/UX)

### 4.1. Улучшение интерфейса на основе отзывов

- \*\*Задача:\*\* Собрать отзывы от пользователей и улучшить интерфейс на их основе.

- \*\*Шаги реализации:\*\*

- Провести опрос среди пользователей (диспетчеров и монтажников) для выявления слабых мест.

- Внести изменения в UI на основе полученных отзывов.

### 4.2. Доступ к функционалу в зависимости от ролей

- \*\*Задача:\*\* Ограничить доступ к определенным страницам для разных пользователей.

- \*\*Шаги реализации:\*\*

- Скрыть ссылки на страницы, доступные только администраторам или диспетчерам, от других пользователей.

- Реализовать проверку ролей на сервере при доступе к страницам.

## Этап 5: Документирование и поддержка

### 5.1. Документация API и структуры проекта

- \*\*Задача:\*\* Создать документацию по API и общей структуре проекта.

- \*\*Шаги реализации:\*\*

- Использовать инструменты, такие как Swagger, для документирования API.

- Описать структуру проекта, назначение модулей и зависимости.

### 5.2. Руководство для разработчиков

- \*\*Задача:\*\* Создать руководство по настройке и развертыванию проекта.

- \*\*Шаги реализации:\*\*

- Подготовить инструкции по установке зависимостей, настройке базы данных и запуску сервера.

- Добавить информацию о структуре проекта и назначении каждого модуля.

\*\*Итог:\*\* После выполнения всех этапов проект станет более стабильным, безопасным и масштабируемым. Пользовательский интерфейс улучшится, а функционал будет более удобным для пользователей с разными ролями. Также внедрение логирования и тестирования обеспечит упрощение поддержки и снижения количества ошибок в будущем.

Когда будете готовы, можем приступить к любому из этапов, обсуждая его детально и шаг за шагом.