Проект представляет собой веб-приложение «We Watch the Bees» – помощника для пчеловодов, объединяющего функции удалённого мониторинга состояния ульев и базы знаний по пчеловодству. Система позволяет пользователям регистрироваться, добавлять ульи с привязанными виртуальными датчиками (температура, влажность, вес), получать и визуализировать данные с этих датчиков в виде интерактивных графиков, а также пользоваться энциклопедией статей, разделённых по категориям, и оставлять комментарии к материалам.

Общая технологическая база (для всех вариантов)

- **Python 3.10+** – язык, на котором будет реализован весь бэкенд, а также скрипты эмуляции датчиков.
- **Django 4.x** – веб-фреймворк. Он берёт на себя:
  - Маршрутизацию (определение соответствия URL и представлений).
  - Работу с базой данных через ORM (объектно-реляционное отображение) – взаимодействие с таблицами как с классами Python.
  - Аутентификацию (регистрация, вход, выход) – практически готовые решения.
  - Административный интерфейс – встроенная админка для управления данными (пользователи, статьи, комментарии).
  - Формы – для обработки пользовательского ввода.
  - Шаблоны – для генерации HTML-страниц.
- **SQLite** – база данных, хранящаяся в одном файле. Не требует отдельной установки и настройки, идеальна для учебного проекта.
- **Bootstrap 5** – CSS-фреймворк. Подключается через CDN и предоставляет готовые стили для кнопок, карточек, сетки, что обеспечивает аккуратный и адаптивный дизайн.
- **HTML + Django Templates** – для вёрстки страниц. Шаблонизатор Django позволяет вставлять переменные, циклы и условия непосредственно в HTML.
- **JavaScript (ES6)** – для реализации интерактивных элементов:
  - **Fetch API** – встроенное средство браузера для асинхронных запросов к серверу без перезагрузки страницы.
  - **Chart.js** – библиотека для построения графиков, подключается через CDN.
- **Дополнительные пакеты Python (устанавливаются через pip)**:
  - `django-bootstrap5` – упрощает рендеринг форм в стиле Bootstrap (необязательно, но удобно).
  - `python-decouple` – для хранения секретных ключей и настроек в отдельном файле (повышает безопасность).
- **Эмулятор датчиков** – отдельный скрипт (или набор скриптов), написанный на Python, который имитирует работу реальных IoT-устройств. Он периодически генерирует случайные значения (температура, влажность, вес) и отправляет их на REST API сервера с помощью HTTP-запросов. Для этого используются стандартные библиотеки: `random`, `time`, `requests` (или `aiohttp` для асинхронности). Эмулятор не является частью основного Django-приложения, а работает параллельно, что позволяет тестировать систему в условиях, приближенных к реальным.

Вариант 1. Полноценный, но упрощённый (соответствует макету)

Этот вариант включает **и мониторинг пасеки, и базу знаний**, полностью покрывая страницы, представленные в макете Figma. Мониторинг реализован с эмуляцией данных через внешний скрипт-эмулятор, а визуализация осуществляется с помощью графиков.

Функционал

- **Регистрация, вход, выход** – стандартные формы Django.
- **Профиль пользователя** – просмотр и редактирование имени, email, пароля.
- **Управление пасекой**:
  - Каждый пользователь имеет **одну пасеку** (упрощение для снижения сложности).
  - Возможность добавления ульев: название/номер, порода пчёл, возраст матки (опционально). При создании улья автоматически создаются три виртуальных датчика (температура, влажность, вес) с уникальными идентификаторами.
  - На главной странице отображаются плитки всех ульев с последними показаниями и цветовой индикацией (зелёный – норма, жёлтый – предупреждение, красный – тревога).
  - **Карточка улья**:
    - Отображает текущие показания по каждому датчику.
    - Содержит графики за последние сутки и неделю (переключение с помощью кнопок), построенные с использованием Chart.js.
    - Данные для графиков запрашиваются через Fetch API и отображаются динамически.
- **Эмуляция данных**:
  - После добавления улья запускается (вручную или автоматически) скрипт-эмулятор, который периодически отправляет POST-запросы на эндпоинт `/api/v1/sensors/data` с JSON-пакетом, содержащим идентификатор датчика, значение и временную метку.
  - Частота отправки настраивается (например, раз в 15 минут). Скрипт может работать на локальной машине или на том же сервере, не требуя сложной инфраструктуры.
  - Сервер принимает данные, проверяет подлинность (по API-ключу или идентификатору датчика) и сохраняет их в БД.
- **База знаний**:
  - Категории статей создаются администратором через встроенную админку Django.
  - Просмотр статей: список по категориям.
  - Детальная страница статьи с текстом и комментариями.
  - Добавление статьи: любой авторизованный пользователь может создать статью через форму (кнопка «+ Добавить статью»). Статья публикуется сразу (без модерации).
  - Комментарии: плоские (без вложенности), авторизованные пользователи могут оставлять комментарии под статьями.
- **Навигация**: шапка сайта содержит вкладки «Моя пасека», «База знаний», «Профиль» – в соответствии с макетом.

Стек технологий (с пояснениями)

- **Django** – обеспечивает всю серверную логику, включая API-эндпоинты для приёма данных от датчиков.
- **SQLite** – база данных для хранения информации о пользователях, ульях, показаниях, статьях и комментариях.
- **Bootstrap 5** – для вёрстки всех страниц.
- **Chart.js** – для построения графиков в карточке улья; данные подгружаются асинхронно.
- **Fetch API (JavaScript)** – для получения данных по выбранному периоду без перезагрузки страницы.
- **django-bootstrap5** – для стилизации форм.
- **python-decouple** – для управления настройками.
- **Скрипт-эмулятор датчиков** – отдельное Python-приложение, использующее библиотеки `requests`, `random`, `time`. Оно может быть запущено как фоновый процесс и не требует интеграции с Django.

Преимущества

- Полное соответствие макету Figma.
- Охвачены обе ключевые идеи проекта: мониторинг пасеки и база знаний.
- Эмуляция датчиков приближена к реальному сценарию (внешние устройства отправляют данные по HTTP).
- Графики и асинхронные запросы делают интерфейс современным и интерактивным.

Недостатки

- Требуется разработка отдельного скрипта-эмулятора.
- Необходимо продумать механизм аутентификации датчиков (например, использование секретных ключей).
- Больший объём работ по сравнению с вариантами, не включающими эмуляцию.

Вариант 2. Фокус на мониторинг (без базы знаний)

Этот вариант концентрируется исключительно на IoT-части, реализуя расширенный функционал мониторинга: несколько пасек, выбор типов датчиков, генерация оповещений. База знаний не реализуется (соответствующие страницы макета можно заменить заглушками).

Функционал

- **Регистрация, вход, выход** – аналогично варианту 1.
- **Профиль пользователя** – редактирование личных данных.
- **Пасеки**: пользователь может создавать несколько пасек (название, местоположение).
- **Ульи**:
    - При добавлении улья указывается название и выбирается, какие датчики установлены (флажки: вес, температура, влажность). Каждому датчику присваивается уникальный идентификатор.
    - Для каждого выбранного датчика в БД создаётся соответствующая запись.
- **Эмуляция данных**:
    - Для каждого улья запускается отдельный экземпляр скрипта-эмулятора (или один многопоточный), который отправляет данные от имени всех датчиков этого улья.
    - Скрипт использует API сервера (`POST /api/v1/sensors/data`) и может работать с заданной периодичностью.
    - Альтернативно можно использовать один универсальный эмулятор, который перебирает все активные датчики в системе (получая их список через API) и отправляет показания.
- **Главная страница**: отображаются все ульи пользователя (с группировкой по пасекам) в виде плиток с последними показаниями и цветовой индикацией.
- **Карточка улья**:
    - Текущие показания по каждому датчику.
    - Графики за день, неделю, месяц (Chart.js, данные через Fetch API).
    - Список оповещений, относящихся к данному улью.
- **Оповещения**:
    - При обработке новых показаний сервер проверяет пороговые значения (например, температура > 38°C, резкое падение веса). При обнаружении аномалии создаётся запись в таблице `Alert`.
    - На главной странице отображается счётчик непрочитанных оповещений.
    - Доступна отдельная страница со списком всех оповещений, где можно фильтровать их по статусу и помечать как прочитанные.
- **Страницы из макета, не относящиеся к мониторингу**, могут быть оставлены заглушками с сообщением «В разработке».

Стек технологий

- Полностью повторяет стек варианта 1 (Django, SQLite, Bootstrap, Chart.js, Fetch API, django-bootstrap5, python-decouple).
- Дополнительно: эмулятор датчиков (Python + requests) может быть более сложным, поддерживающим многопоточность или асинхронность.

Преимущества

- Глубокая проработка мониторинга – основной ценности проекта.
- Поддержка нескольких пасек и гибкой конфигурации датчиков.
- Система оповещений добавляет практическую полезность.
- Эмуляция через внешний скрипт максимально приближена к реальной интеграции с IoT-устройствами.

Недостатки

- Отсутствует база знаний, что делает макет не полностью реализованным.
- Требуется реализация логики проверки аномалий и цветовой индикации.
- Необходимо обеспечить надёжную работу эмулятора (управление несколькими датчиками).

Вариант 3. Фокус на базу знаний (без мониторинга)

Данный вариант реализует энциклопедию и сообщество, полностью опираясь на макет, но исключая мониторинг пасеки. Это наиболее простой в реализации вариант, так как не требует работы с датчиками, графиками и асинхронными запросами.

Функционал

- **Регистрация, вход, выход**.
- **Роли**: обычный пользователь и администратор (используется стандартный флаг `is_staff` в Django).
- **Категории статей** – создаются администратором через админку.
- **Статьи**:
  - Любой авторизованный пользователь может создать статью через форму (кнопка «+ Добавить статью»). Поля: категория, заголовок, текст.
  - Новая статья получает статус «черновик» (`is_published=False`).
  - Администратор в админке публикует статью (меняет флаг) или удаляет её.
  - На главной странице отображаются только опубликованные статьи, сгруппированные по категориям.
  - Реализован простой поиск по заголовку и содержанию (с использованием `icontains` в ORM).
- **Комментарии**:
  - Под каждой опубликованной статьёй находится форма для добавления комментария (только текст).
  - Комментарии видны сразу после добавления (без предварительной модерации). Администратор имеет возможность удалить неуместный комментарий через админку.
  - Комментарии являются плоскими (без вложенности), что упрощает модель и шаблоны.
- **Профиль пользователя**: отображается список его статей и комментариев.
- **Навигация**: шапка сайта содержит вкладки «Главная» (список статей) и «Профиль». Вкладка «Моя пасека» из макета не реализуется или ведёт на заглушку.

Стек технологий

- **Django** – со стандартными компонентами, без необходимости в Chart.js и Fetch API.
- **SQLite** – база данных.
- **Bootstrap 5** – для оформления.
- **django-bootstrap5** – для форм.
- **python-decouple** – для настроек.
- **JavaScript** – практически не требуется; минимальное использование возможно для подтверждения удаления, но это опционально.

Преимущества

- Минимальный порог вхождения: требуется освоить только Django и Bootstrap.
- Быстрая реализация: основное время уходит на модели, шаблоны и логику модерации.
- Полностью соответствует макету, за исключением страницы карточки улья (заменяется заглушкой).
- Не требует эмуляции датчиков и работы с графиками.

Недостатки

- Отсутствует мониторинг пасеки – ключевая функция, заявленная в видении проекта.
- Меньше возможностей для демонстрации навыков работы с JavaScript и асинхронными запросами.