Архитектура проекта “Нахождение аномалий в веб-приложениях методом анализа временных рядов”

1.**Вид архитектуры**

Монолитная архитектура — это подход к разработке программного обеспечения, при котором все компоненты приложения объединены в единое целое. В таком подходе все функции, модули и интерфейсы находятся в одном кодовом базе и развертываются как единый артефакт.

Основные характеристики монолитной архитектуры:

* Единая кодовая база: Все компоненты приложения находятся в одном репозитории, что упрощает управление кодом.
* Централизованное развертывание: Приложение разворачивается как единое целое, что упрощает процесс развертывания.
* Простота разработки: Для небольших команд и проектов монолитная архитектура может быть проще в реализации и управлении. Это основная причина выбора данной архитектуры для нашего приложения по анализу временных рядов
* Высокая производительность: Взаимодействие между компонентами происходит внутри одного процесса, что может обеспечить более высокую производительность по сравнению с распределенными системами.

Причины выбрать монолитную архитектуру:

* Простота разработки и развертывания: Для небольших проектов или стартапов монолитная архитектура позволяет быстро начать разработку и развертывание приложения без необходимости управлять сложной инфраструктурой.
* Легкость в тестировании: Тестирование монолитного приложения может быть проще, так как все компоненты находятся в одном месте, и можно использовать интеграционные тесты для проверки всего приложения.
* Упрощенное управление зависимостями: В монолите все зависимости находятся в одном проекте, что упрощает управление библиотеками и версиями.
* Снижение накладных расходов: Монолитные приложения могут иметь меньшие накладные расходы на взаимодействие между компонентами, так как все они работают в одном процессе.
* Быстрая разработка MVP: Для создания минимально жизнеспособного продукта (MVP) монолитная архитектура позволяет быстро реализовать идеи и протестировать их на рынке.

Альтернатива:

Микросервисная архитектура — это подход к разработке программного обеспечения, при котором приложение разбивается на небольшие, независимые сервисы, каждый из которых выполняет определенную бизнес-функцию. Эти сервисы взаимодействуют друг с другом через API и могут быть развернуты и масштабированы независимо. Вот несколько причин, по которым стоит рассмотреть выбор микросервисной архитектуры:

* Масштабируемость. Микросервисы позволяют масштабировать отдельные компоненты приложения в зависимости от нагрузки. Это означает, что можно выделить больше ресурсов для наиболее загруженных сервисов, не затрагивая остальные.
* Гибкость в выборе технологий. Каждый микросервис может быть разработан с использованием различных технологий и языков программирования, что позволяет командам выбирать наиболее подходящие инструменты для конкретной задачи.
* Упрощенное развертывание. Микросервисы могут быть развернуты независимо друг от друга. Это позволяет командам вносить изменения и обновления в один сервис без необходимости перезапуска всего приложения, что снижает время простоя.
* Устойчивость к сбоям. Если один микросервис выходит из строя, это не обязательно приводит к сбою всего приложения. Другие сервисы могут продолжать работать, что повышает общую надежность системы.
* Улучшенная командная работа. Микросервисная архитектура позволяет разделить приложение на небольшие, управляемые команды, каждая из которых отвечает за определенный сервис. Это способствует более эффективной работе команд и улучшает коммуникацию.
* Легкость в тестировании. Каждый микросервис можно тестировать независимо, что упрощает процесс тестирования и позволяет быстрее находить и исправлять ошибки.

2. **Структура базы данных**

Программа создает и управляет следующими таблицами в базе данных base.db:

activities:

* id (INTEGER, PRIMARY KEY)
* date (INTEGER, PRIMARY KEY): время, в которое был сделан запрос.
* content (INTEGER): характер запроса.

anomalies:

* id (INTEGER, PRIMARY KEY): уникальный идентификатор аномалии.
* level (BOOLEAN): тип аномалии (временная или хакерская).
* date (INTEGER): время, в которое была зафиксирована аномалия.

Users:

* id (INTEGER, PRIMARY KEY)
* login (STRING)
* password (STRING)

3. **Функции программы**

* get\_user(username : str = None,  id : str = None, \_session = Session) -> User | None – возвращает пользователя по никнейму, None в случае, если пользователь не зарегистрирован
* add\_user( username : str, password : str ) -> User – добавление нового пользователя
* get\_logs() – получение логов
* add\_log() – добавление лога

Основная функция программы, которая:

* Запрашивает URL-адрес.
* Подготавливает базу данных.
* Получает данные о запросах.
* Находит аномалии.
* Вставляет данные в базу данных.
* Проверяет наличие аномалий и выводит соответствующие сообщения.

4**. Поток выполнения**

Пользователь вводит URL-адрес.

Вызывается функция prepare\_database() для создания необходимых таблиц.

Данные о запросах получаются с помощью get\_acivities\_from\_dataset().

Аномалии определяются с помощью find\_amonalies().

Данные о запросах и аномалиях вставляются в соответствующие таблицы.

Программа проверяет наличие аномалий и выводит результаты.

5. **Зависимости**

sqlalchemy: библиотека Python для работы с SQLite.  
flask: библиотека Python для развертывания сервера.