

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Направление: 02.03.02

«Фундаментальная информатика и информационные технологии»

Основная образовательная программа: СВ.5003.2020

«Программирование и информационные технологии»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

*«Разработка серверной части геоинформационного веб-приложения для
точного земледелия»*

Выполнил:

Учебная группа

Аминов Д. Т.

20.Б11-пу

Научный руководитель:

Должность:

Митрофанова О. А.

доцент

Санкт-Петербург
2024

СОДЕРЖАНИЕ

1	ВВЕДЕНИЕ	4
1.1	Актуальность работы	4
1.2	Практическая значимость работы	5
1.3	Цель работы.....	5
2	ОБЗОР ИССЛЕДОВАНИЙ И СУЩЕСТВУЮЩИХ РЕШЕНИЙ .	7
2.1	Обзор исследований	7
2.2	Обзор существующих решений.....	8
3	ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРИЛОЖЕНИЮ	12
3.1	Функциональные требования.....	12
3.2	Нефункциональные требования.....	13
4	ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ	14
4.1	Выбор паттерна проектирования серверной части приложения	14
4.2	Межсервисное общение	18
4.3	Аутентификация и авторизация в микросервисной архитектуре	21
4.4	Итоговая архитектура	27
5	ПРОГРАММНАЯ РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ	30
5.1	Выбор технологий	30
5.2	Микросервис аутентификации	31
5.3	Микросервис метаданных	32
5.4	Микросервис профилей	35
5.5	API Gateway.....	36
6	ТЕСТИРОВАНИЕ И РАЗВЁРТЫВАНИЕ НА УДАЛЁННОЙ МАШИНЕ	37
7	ЗАКЛЮЧЕНИЕ	38
7.1	Результаты работы	38

7.2	Дальнейшее развитие	39
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	41

1 ВВЕДЕНИЕ

1.1 Актуальность работы

За последние 50 лет сельскохозяйственная отрасль радикально изменилась. Достижения в области машиностроения увеличили масштабы, скорость и производительность сельскохозяйственной техники, что привело к более эффективной обработке большего количества земель. Семена, ирригация и удобрения также значительно улучшились, что помогло фермерам повысить урожайность. Сейчас сельское хозяйство находится на заре ещё одной революции, в основе которой лежат информационные технологии(ИТ).

Одним из способов использования ИТ в сельском хозяйстве является точное земледелие. Точное земледелие — комплексная высокотехнологичная система сельскохозяйственного менеджмента, цель которой - оптимизация затрат и получение максимальной прибыли и урожая. Сама концепция возникла ещё в 1980-х годах, когда в США стали делать первые карты по дифференцированному внесению удобрений на основании анализов почвы. [1] Но широкое распространение идеи точного земледелия получили в последние пять лет благодаря развитию мобильных технологий, высокоскоростному интернету и точным спутниковым данным. [2; 3]

Точное земледелие включает множество отдельных технологий, необходимость внедрения которых определяется на усмотрение собственников и руководителей агропредприятия. Примерами таких технологий являются: глобальное спутниковое позиционирование (GPS), оценка урожайности YMT (Yield Monitor Technologies), геоинформационные системы (GIS), дистанционное зондирование земли (ДЗЗ) и другие. Можно задействовать как все технологии, входящие в комплекс, или только некоторые, эффект от которых будет наиболее значимым для данного предприятия. [4; 5]

1.2 Практическая значимость работы

Серверная часть веб-приложения будет интегрирована с клиентской частью с целью получения готового продукта для конечного пользователя.

Пользователями данного веб-сервиса могут быть организации и фермеры, владеющие сельхозугодьями, агрономы, учёные и другие заинтересованные лица, являющиеся сотрудниками соответствующих организаций или фермеров. «Цифровой двойник сельскохозяйственного поля» позволит им провести цифровизацию различных процессов управления сельскохозяйственными полями.

Ключевыми же факторами для цифровизации данных процессов являются оптимизация использования ресурсов, таких как химические удобрения и пестициды, а также максимизация урожайности. Эти факторы могут быть достигнуты за счёт сбора и анализа метеоданных, данных о почве, аэрофотоснимков и т.д.

Так, метеоданные, такие как температура и влажность могут оптимизировать принятие решений по планированию будущих севооборотов на конкретных полях.

В то же время использование агрохимических данных может способствовать идентификации областей с высоким уровнем риска возникновения болезней, плесени и других проблем, которые могут негативно сказаться на урожае. Это позволяет предпринимать своевременные меры для предотвращения потерь урожая и сокращения расходов на лечение и защиту растений.

1.3 Цель работы

Данная работа посвящена разработке серверной части веб-приложения «Цифровой двойник сельскохозяйственного поля» - геоинформационной системы для точного земледелия. Процесс разработки можно разбить на следующие части:

1. Обзор исследований и существующих решений;
2. Формирование требований к приложению;
3. Проектирование архитектуры серверной части приложения;
4. Программная разработка серверной части приложения;
5. Тестирование и развертывание приложения на удалённом сервере;

2 ОБЗОР ИССЛЕДОВАНИЙ И СУЩЕСТВУЮЩИХ РЕШЕНИЙ

2.1 Обзор исследований

В статье [6] рассматривается «Soil daTA Retrival Tool (START)» - инструмент подготовки данных для имитационных моделей сельскохозяйственных культур с использованием WEB-ориентированных баз данных о почвах. На примере «START» утверждается, что создание и инициализация входных файлов для почвенных данных занимают около 0.33 % от времени, которое требуется на подготовку данных вручную. Эти результаты позволяют предположить, что использование специализированного ПО может обеспечить эффективный подход к подготовке файлов входных данных о почвах.

В работе [7] были выполнены исследования в рамках развития базовых технологий точного земледелия для улучшения производственных и экологических показателей растениеводства. Эти исследования были направлены на определение оптимальной урожайности на основе климатических и почвенных факторов, а также на разработку методов управления качеством и объёмом урожая. Был сделан вывод, что применение дифференцированных агротехнологий наиболее перспективно для эффективного управления производством растениеводческой продукции.

Исследования [8], [9] указывают на широкое применение аэрофотоснимков в точном земледелии с акцентом на детальном дифференцированном подходе к управлению системой "поле-посев". Новые методы, основанные на анализе аэрофотоснимков, представляют собой перспективную альтернативу традиционным методам полевого анализа уровня питательности растений и необходимости использования химических средств защиты. Эти методы показали значительные преимущества, включая высокую разрешающую способность снимков, быстрое и экономичное получение изображе-

ний и результатов, а также возможность наблюдения за посевами в различные временные интервалы.

В исследовании [10] авторы разрабатывают сервис-ориентированную архитектуру и интегрируют пространственные данные для агроконсультационных систем на основании экспериментов, проведённых в Северном Гуджарате. Результатом данной работы является архитектура с **RESTful** веб-сервисами, обрабатывающими запросы фермеров для манипуляции геопропространственными данными, хранящимися в PostgreSQL. В заключение авторы утверждают, что существует необходимость обобщить предложенную ими архитектуру, а также необходимо протестировать и реализовать дальнейшее бесшовное взаимодействие между источниками пространственных данных.

2.2 Обзор существующих решений

На сегодняшний день существует множество компьютерных программ для решения задач точного земледелия. Рассмотрим самые популярные из них.

1. *Агроаналитика-IoT* [11] - онлайн-сервис оказания консультационных услуг, в области сельскохозяйственного производства с целью снижения неопределённости и, соответственно, рисков в агропромышленном комплексе; создания благоприятного инвестиционного климата; увеличения рентабельности производства сельскохозяйственной продукции.

Плюсы:

- Широкий функционал: автоматизация и прозрачность бизнес-процессов агропредприятия, планирование агрономических и инженерных работ, кадастровый контроль и работа с земельным фондом и т.п.;
- Высокая производительность ИОТ сервера: получение треков техники по 1000 единиц за сутки занимает около 10 секунд, получение последнего местоположения - 5 секунд;

- Возможность интеграции с другими системами (например, 1С);
- Наличие чат-бота и мобильного приложения;

Минусы:

- Нет поддержки функционала для работы с аэрофотоснимками;
- Цена: от 30р в год за гектар сельхозугодий. Доступны разные тарифы;

2. *Cropwise Operations*[12] - это система спутникового мониторинга полей, специально разработанная для руководителей аграрных компаний и агрономов. Система контролирует состояние посевов в режиме реального времени, следит за вегетацией полей в различных регионах, выявляет проблемные участки и рассчитывает рекомендованную норму удобрений.

Плюсы:

- Широкий функционал: мониторинг и контроль посевных площадей, уточненный прогноз погоды для каждого поля, создание заданий по выполнению работ на поле и т.п.;
- Возможность интеграции с другими системами (например, 1С);
- Автоматические оповещения о превышении скорости, выезде за пределы геозоны, сливе топлива;
- Наличие мобильного приложения;

Минусы:

- Нет поддержки функционала для работы с аэрофотоснимками;
- Цена: от \$1 до \$5 за гектар в год, в зависимости от страны и размера полей, которые вы собираетесь контролировать;

3. *OneSoil* [13] - платформа, которая помогает эффективно управлять полями. Платформа объединяет мобильное приложение

OneSoil Scouting, веб-приложение OneSoil и инструменты PRO-версии для точного земледелия OneSoil Yield. Вместе они помогают отслеживать изменения на полях, планировать сельскохозяйственные работы, повышать урожайность поля и экономить ресурсы вашей фермы.

Плюсы:

- Широкий функционал: Создание полей, внесение севооборотов, возможность делиться полями и заметками, получение метеоданных и т.д.;
- Наличие мобильного приложения;
- В PRO-версии доступны инструменты для дифференцированного посева;
- В PRO-версии доступна интеграция с другими платформами;
- Наличие мобильного приложения;

Минусы:

- Нет поддержки функционала для работы с аэрофотоснимками;
- Некоторые функции доступны только в платной PRO-версии;

4. *QGIS*[14] - бесплатное ПО для геоинформационного анализа. Оно используется для создания и редактирования геоданных. *QGIS* широко используется географами, картографами, специалистами по геодезии и геоинформатике. Еще одним применением *QGIS* может быть мониторинг и анализ сельхоз полей. Важно отметить, что *QGIS* сам по себе не является готовым продуктом «из коробки» в сравнении с предыдущими решениями, а требует самостоятельного написания кода и настройки под свои задачи.

Плюсы:

- Гибкость в настройке: пользователи могут настраивать интерфейс, разрабатывать и настраивать собственные инструменты на языке программирования Python;

- Открытое API: как следствие, возможность интеграции почти с любыми системами;
- Цена: бесплатно;

Минусы:

- Не является готовым продуктом «из коробки». Требуется самостоятельного написания кода и настройки под свои задачи.
- Специалистам сельскохозяйственной сферы сложно использовать этот инструмент из-за отсутствия экспертных знаний.

В процессе исследования были рассмотрены и другие решения: АгроСигнал [15], АгроМон [16] и другие. Все исследуемые решения, за исключением QGIS имеют богатый функционал, отличную техническую поддержку и сопровождение. Однако за подключение к системе и обслуживание взимается плата.

QGIS, напротив, является бесплатным ПО с открытым исходным кодом, но требует самостоятельной настройки и написания кода. QGIS является единственным рассмотренным решением с поддержкой функционала для работы с аэрофотоснимками.

3 ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРИЛОЖЕНИЮ

3.1 Функциональные требования

1. Должна быть спроектирована система хранения и обработки информации о сельхозугодье (поле). Так, для каждого поля должны храниться:

- название, площадь, полигон и контур поля;
- данные о текущем посеве: дата начала, дата окончания, культура и т.д.

Посевы полей могут меняться со временем, но должна храниться история всех посевов для каждого поля - севооборот;

2. Должны быть разработаны механизмы аутентификации и авторизации, дающие защиту от несанкционированного доступа к данным и управления ресурсами. Так, пользователями приложения могут быть:

- организации: предприятия/фермеры являющиеся владельцами сельхозугодий, они могут предоставлять доступ к полям для своих сотрудников;
- сотрудники: наблюдатели, инженеры, трактористы, и т.д. Они обладают меньшими правами доступа по сравнению с организациями и фермерами;

Регистрацией корпоративного аккаунта сотрудника в системе занимается организация на которую работает данный сотрудник.

3. Пользователи должны иметь профили, содержащие их личные данные. Профиль организации состоит из следующих данных:

- название
- описание (опционально)
- город
- ИНН (опционально)
- номер телефона (опционально)
- веб-сайт (опционально)

Профиль сотрудника состоит из следующих данных:

- имя
- фамилия
- отчество (опционально)
- дата рождения (опционально)
- номер телефона (опционально)

4. Должна быть разработана ETL-система хранения, сбора, обработки метеоданных о полях, выбранных пользователем.

Каждый день в 5:00, 9:00, 13: 00 и 17:00 должны загружаться следующие данные, относящиеся к сельхозугодью, актуальные на момент их получения:

- температура
- влажность
- уровень осадков (мм)
- точка росы
- температура почвы на глубине/глубинах 0-18 см
- влажность почвы на глубине/глубинах от 0-27 см

Также раз в сутки должны загружаться следующие данные:

- максимальная и минимальная температуры
- время восхода солнца
- время заката
- сумма осадков (мм)

3.2 Нефункциональные требования

Как мы могли видеть из [2.2](#), коммерческие приложения для точного земледелия, в основном, имеют широкий функционал. Следовательно, приложение должно быть открыто к добавлению нового функционала, интеграции с другими системами.

4 ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ

4.1 Выбор паттерна проектирования серверной части приложения

Паттерны проектирования - это описания коммуникационных объектов и классов, которые настроены для решения общей проблемы проектирования в определенном контексте. [17]

На сегодняшний день существуют два популярных подхода к проектированию веб-приложений: монолитное приложение и микросервисы.

4.1.1 Монолитная архитектура

Монолитная архитектура — это классический подход, который используется при создании программных приложений. Являлся доминирующим подходом к разработке программного обеспечения на протяжении многих лет и до сих пор широко используется. Монолитная архитектура объединяет и запускает все компоненты приложения как единую унифицированную систему. Таким образом, любые изменения, внесенные в одну часть приложения, потенциально могут повлиять на всю систему. [18]

На рисунке 1 мы можем видеть, что слои пользовательского интерфейса, бизнес-логики и манипулирования данными объединены в одно приложение.

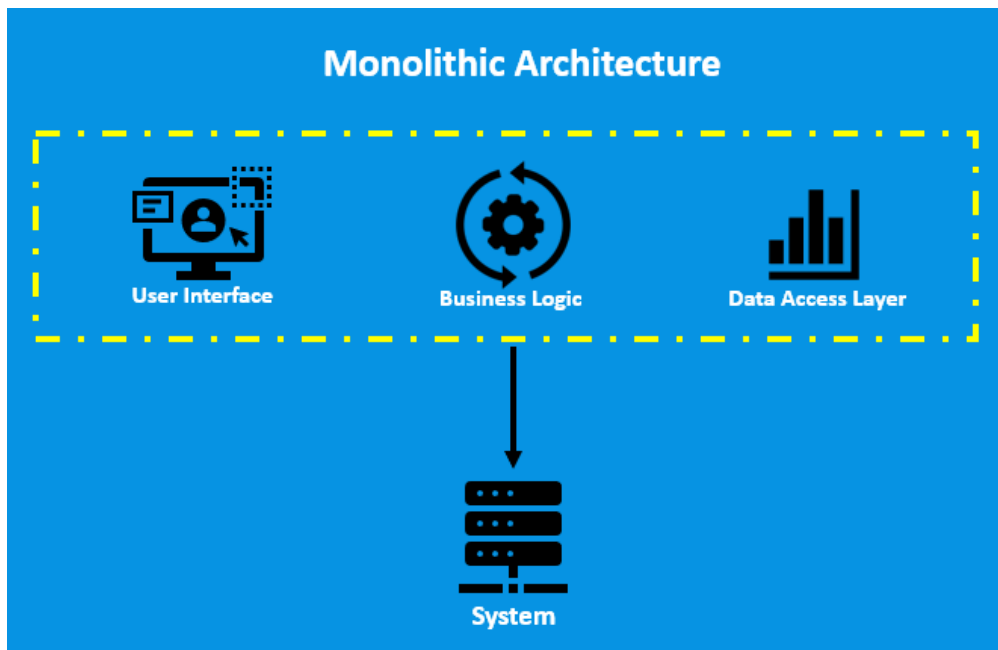


Рисунок 1 — Монолитная архитектура. Источник: [18]

Выделим ключевые преимущества монолитной архитектуры:

- **Упрощенная разработка, тестирование и развертывание:** Команды создают и тестируют монолитные приложения в согласованной инфраструктуре с использованием аналогичных инструментов в виде единой базы кода, а затем развертывают эти приложения как один контейнер в рабочей среде.
- **Быстрое решение проблем:** Несмотря на то, что компоненты приложения в монолитной архитектуре более сложны, чем в микросервисах, запросы рабочих процессов легче отслеживать, что позволяет разработчикам обнаруживать проблемы по мере их возникновения. [19]

Ключевые недостатки:

- **Трудности масштабирования:** монолитные приложения имеют тесно связанные программные компоненты, работа которых зависит друг от друга. Эффективное масштабирование практически невозможно.

- **Сильная связность:** когда один компонент системы замедляется или выходит из строя, все приложение может перестать работать должным образом.
- **Трудности поддержки:** по мере роста кодовой базы становится сложнее поддерживать приложение, тратится много времени на ускорение работы ресурсов и устранение неполадок. [18; 19]

Монолитная архитектура является оптимальным решением когда известно, что размер и основной функционал приложения не будет расширяться. Разработчики могут работать над одним и тем же кодом. Для разработки монолитного приложения требуется меньше разработчиков, чем для разработки микросервисного.

4.1.2 Микросервисная архитектура

Микросервисная архитектура представляет собой другую методологию разработки программных приложений по сравнению с монолитной. Так, приложение разбивается на небольшие независимые сервисы, которые взаимодействуют друг с другом с помощью четко определенных API(application programming interface). Более того, каждый микросервис отвечает за определенную функциональность приложения. Каждый сервис можно разрабатывать, развертывать и масштабировать независимо от других сервисов. Таким образом, обеспечивается большая гибкость при разработке и поддержке приложения. [18; 19]

Пример микросервисной архитектуры, можно увидеть на рисунке 2.

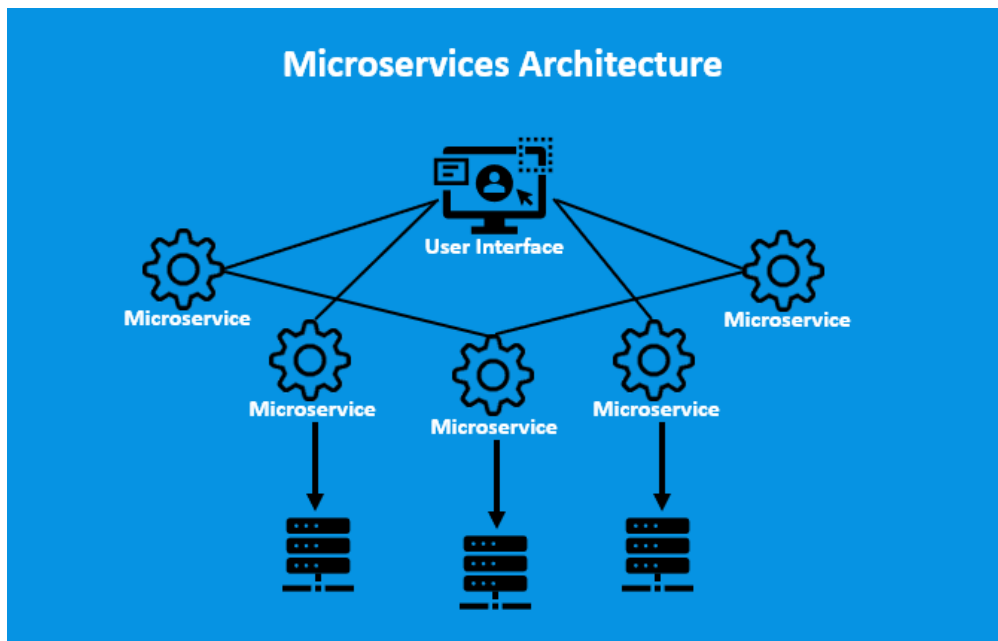


Рисунок 2 — Микросервисная архитектура. Источник: [18]

Выделим ключевые преимущества микросервисной архитектуры:

- **Масштабируемость:** с помощью микросервисов команды могут разрабатывать независимые сервисы, ориентированные на конкретную проблему. Как правило, кодовая база микросервисов не сильно разрастается, что делает поддержку микросервисов более простым процессом.
- **Эффективное использование ресурсов и изолированность:** поскольку микросервисы независимы, организации могут масштабировать отдельные микросервисы в удобном для них темпе без необходимости масштабирования всего приложения. Команды также имеют свободу выбирать собственные технологические стеки для создания микросервисов.
- **Отказоустойчивость:** Если какой-либо из микросервисов перестанет работать, то это не повлечет за собой полную остановку работы всей системы. Достаточно будет устранить неполадку в неработающем микросервисе. [19]

Ключевые недостатки:

- **Сложность развертывания:** чем больше микросервисов, тем сложнее процесс их развертывания на сервере.
- **Необходимо больше ресурсов:** для достижения успеха отдельным командам необходимы собственные системы, программное обеспечение, инфраструктура и операционные процессы, что может привести к более высоким затратам как на оборудование, так и на зарплаты разработчикам.
- **Долгий старт:** Для разработки MVP (минимального жизнеспособного продукта) требуется разработать больше компонент, чем требовалось бы для монолитного приложения.

Микросервисная архитектура предоставляет свободу выбора инструментов для разработки: команды могут использовать те инструменты, которые им нужны для добавления нового функционала. Также, повышается надежность и отказоустойчивость системы, так как микросервисы представляют собой независимые приложения, и в случае возникновения неисправности у одного из них, система не перестанет функционировать. [19]

Так как основное нефункциональное требование к серверной части приложения - возможность масштабирования 3.2, то микросервисная архитектура будет оптимальным выбором для данной задачи.

4.2 Межсервисное общение

После того как мы определились с выбором микросервисной архитектуры, надо определиться с тем, каким образом микросервисы будут взаимодействовать в системе. Общение клиента с микросервисами в микросервисной архитектуре обычно осуществляется через сетевые протоколы и API. Далее, рассмотрим некоторые распространённые способы общения клиента с микросервисами.

4.2.1 Протокол HTTP

Будем считать протоколом общепринятый формат общения систем между собой. На сегодняшний день, большинство веб-протоколов работают

поверх HTTP. Он относится к прикладным протоколам и работает поверх других протоколов сетевого стека. HTTP — это простой текстовый протокол для передачи любого контента. Изначально разработан для передачи HTML-файлов. Как правило, в веб-разработке используются надстройки поверх протокола. HTTP использует TCP в качестве транспортного протокола, который управляет передачей данных между компьютерами. TCP обеспечивает доставку данных с подтверждением от «получателя». Если «отправитель» не получает подтверждения, он повторно отправляет пакет. [20]

Для передачи данных используются HTTP-запросы и HTTP-ответы. На рисунке 3 приведена структура HTTP-запроса:

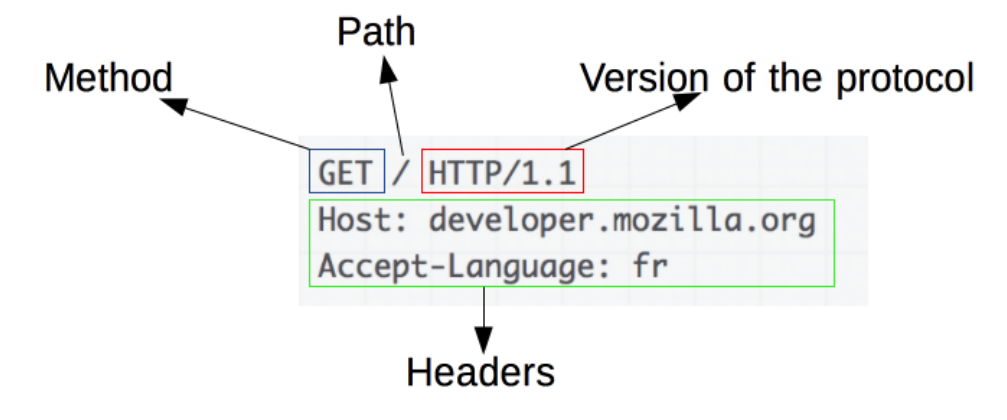


Рисунок 3 — Пример HTTP-запроса. Источник: [20]

Здесь,

- HTTP-метод (method) — это глагол, который определяет, какую операцию мы хотим выполнить (GET, POST, PUT и т.д.);
- Путь (path) — URL до необходимого нам ресурса;
- Версия протокола HTTP (version of the protocol);
- Заголовки (headers) — дополнительные параметры запроса, которые нужны серверу;
- Опционально: тело запроса — данные, которые мы хотим передать.

Структура HTTP-ответа показана на рисунке 4. Из нового, здесь: код состояния (status code), указывающий, был ли запрос успешным или нет и почему, а также сообщение о состоянии (status message).

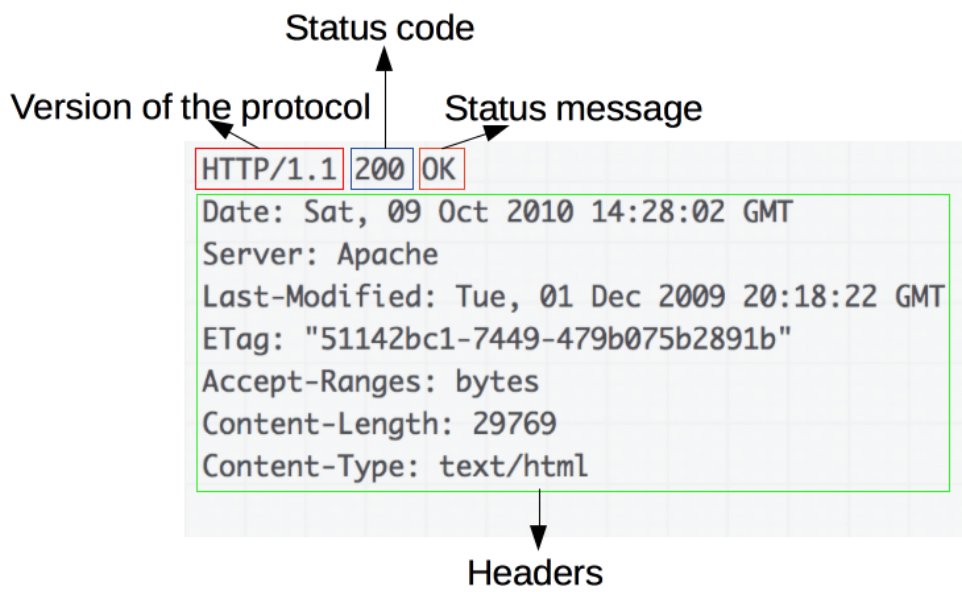


Рисунок 4 — Пример HTTP-ответа. Источник: [20]

Теперь, имея базовое представление о HTTP, мы можем перейти к рассмотрению REST, как стиля межсервисного взаимодействия.

4.2.2 REST

REST (Representational State Transfer, передача репрезентативного состояния) – архитектурный стиль построения распределенной системы, в основе которого лежит понятие ресурса и его состояния. Ресурс имеет свой универсальный идентификатор (URI, unified resource identifier), используя который, над данным ресурсом можно совершать различные действия, например, CRUD (create, read, update, delete). [21]

Важно отметить, что используя REST, нужно думать о приложении с точки зрения ресурсов. Для управления ресурсами, REST использует HTTP-методы. Таким образом, REST – это не столько конкретная конструкция или реализация, сколько архитектурный стиль, позволяющий наилучшим образом использовать функции, предоставляемые HTTP.

Существуют и другие подходы к микросервисному общению, такие как семейство протоколов RPC, WebSocket и т.д. Однако из-за своей популярности и удобства, REST-подход остается наиболее распространенным в веб-разработке. В дальнейшем, будет предполагаться использования REST и протокола HTTP для проектирования архитектуры серверной части «Цифрового двойника сельскохозяйственного поля».

4.3 Аутентификация и авторизация в микросервисной архитектуре

4.3.1 Введение

Введём определения, необходимые для понимания дальнейшего текста:

- *Аутентификация* - процесс предоставления пользователю доступа к информационной системе. [22]
- *Авторизация* - предоставление определённому лицу или группе лиц прав на выполнение определённых действий. [22]

Выбор способа аутентификации и авторизации является важной задачей разработчиков, поскольку неправильно реализованная система может стать уязвимостью для хакерских атак и угрозой для конфиденциальности пользовательских данных. В микросервисной архитектуре каждый сервис обычно имеет свою собственную базу данных и собственный сетевой интерфейс, что делает эту задачу особенно сложной. Кроме того, важно обеспечить защищенное соединение между всеми микросервисами, чтобы предотвратить возможность перехвата данных во время передачи.

Далее будет рассмотрено несколько способов аутентификации в микросервисах.

4.3.2 Аутентификация в каждом микросервисе

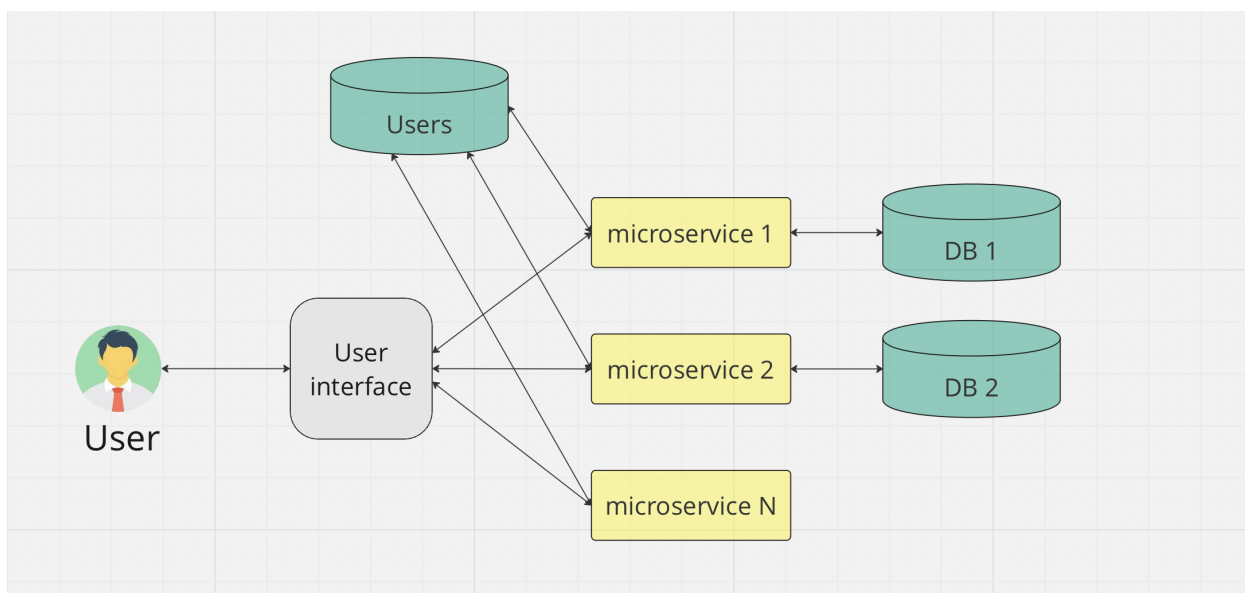


Рисунок 5 — Вариант с аутентификацией в каждом микросервисе

На рисунке 5 показан вариант микросервисной архитектуры с аутентификацией в каждом микросервисе. Так, пользователь, используя клиент-серверный подход, отправляет HTTP-запросы на микросервисы. Для всех микросервисов существует общая база данных (далее БД) «Users», необходимая для получения пользовательских данных для дальнейшей аутентификации и авторизации пользователей.

Основной проблемой данного решения является нарушения принципа слабой связанности в микросервисной архитектуре, т.к. все сервисы имеют доступ к одной БД «Users».

4.3.3 Работа с пользователями в отдельном микросервисе.

Реализация 1

В данной реализации, существует микросервис аутентификации, на который каждый из микросервисов делает запрос (см. рисунок 6).

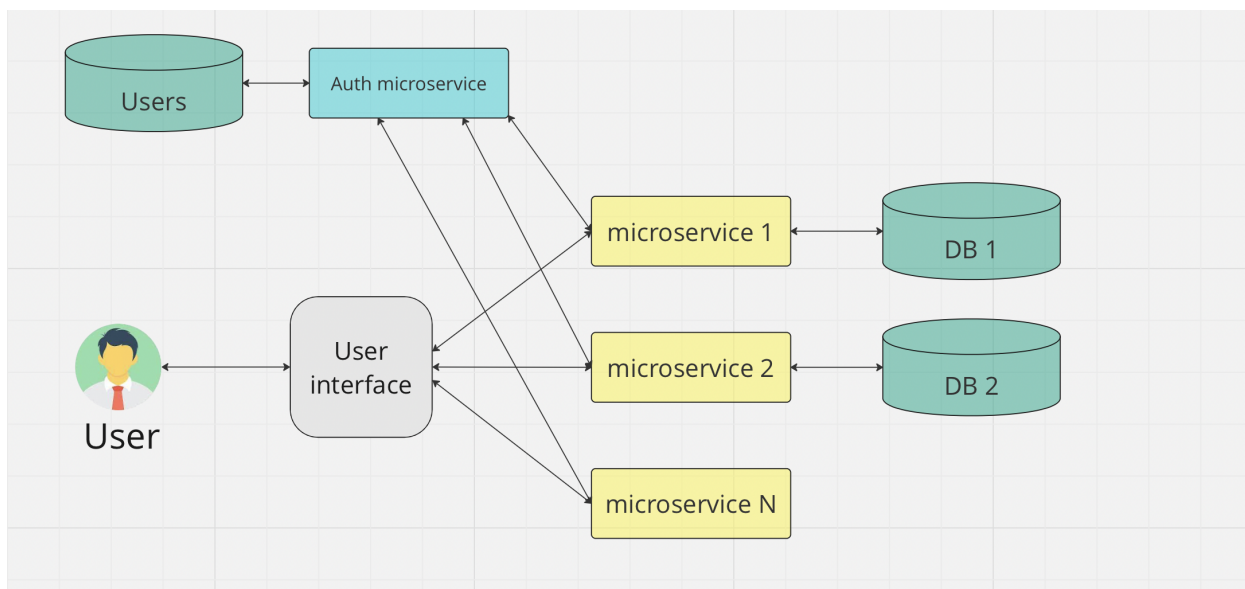


Рисунок 6 — Работа с пользователями в отдельном микросервисе. Реализация 1

Недостатком такого решения можно считать реализацию логики обращения к микросервису аутентификации для каждого микросервиса. Также стоит отметить, что при такой реализации, микросервисы не знают, какие методы требуют аутентификации. Следовательно, на каждый запрос, отправленный в *microservice i*, *microservice i* будет отправлять запрос в микросервис аутентификации для проверки наличия доступа.

4.3.4 Работа с пользователями в отдельном микросервисе.

Реализация 2

Удобство данного метода заключается в том, чтобы клиентской части приложения уже не нужно знать о том, к какому из микросервисов надо обращаться, так как все запросы проходят через микросервис

«Auth» (см. рисунок 7). Также решены проблемы предыдущей реализации.

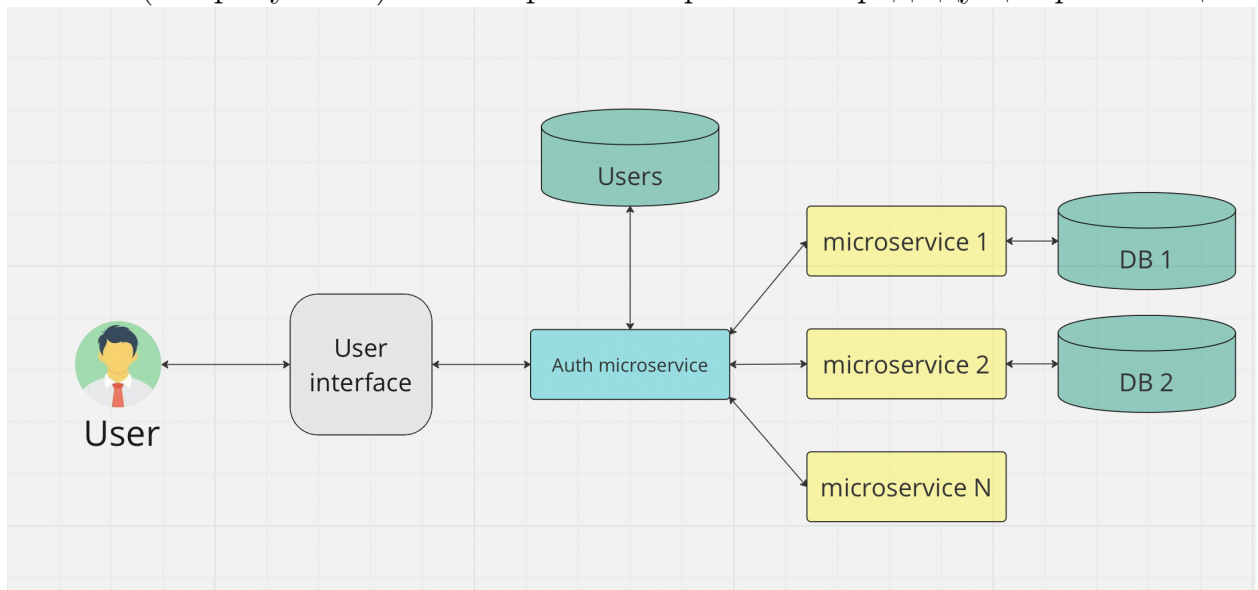


Рисунок 7 — Работа с пользователями в отдельном микросервисе. Реализация 2

Хотя на первый взгляд это решение может показаться оптимальным, оно также обладает недостатками. Во-первых, теперь все запросы проходят через микросервис «Auth», даже те, что не требуют аутентификации. Как следствие, некоторые запросы проходят «неоптимальный маршрут». Во-вторых, микросервис аутентификации имеет свою базу данных, тяжеловесные компоненты (фреймворк, утилиты и т.д.), а значит при каждом запросе, даже не требующем аутентификации, происходит инициализация ядра сервиса, что замедляет время работы всего приложения.

4.3.5 Паттерн API Gateway

API Gateway - это прокси-сервер, который предоставляет единую точку входа для клиентов и обеспечивает управление и контроль доступа к различным микросервисам и их API. API Gateway должен иметь возможность определять, к какому микросервису направить запрос клиента на основе заданных правил и путей. В этом случае, реализация серверной части приложения будет скрыта от клиента, так, обеспечиваются принцип разделения обязанностей (Separation of Concerns) и дополнительная безопасность. Важно отметить, что API Gateway должен быть максимально быстрым, так как все запросы идут через него. [23]

Вариант с API Gateway отображен на рисунке 8:

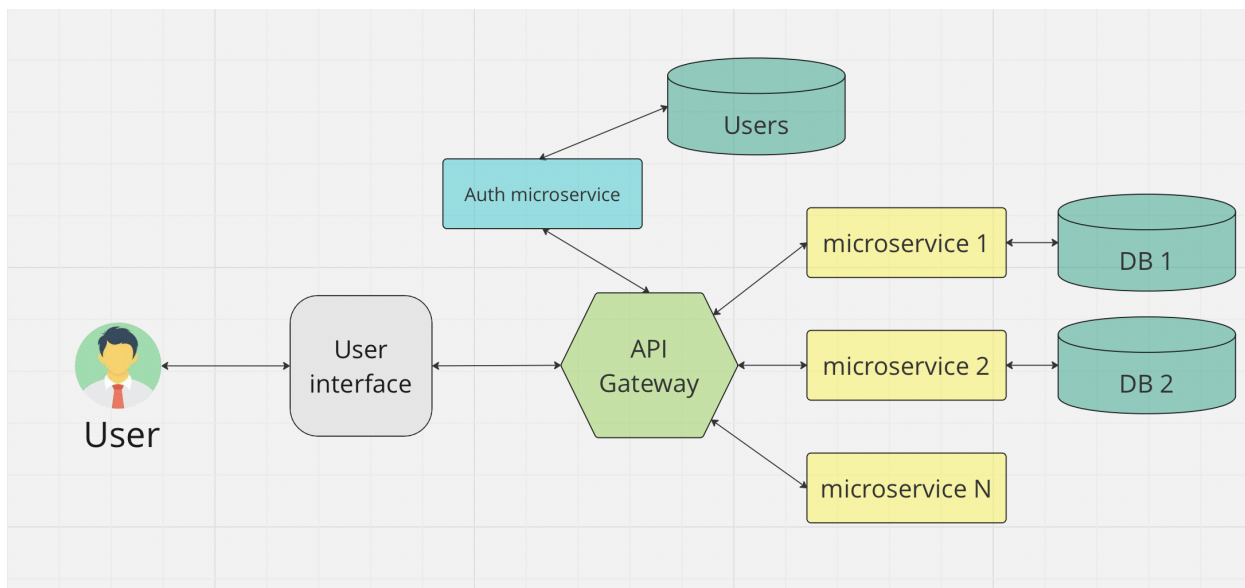


Рисунок 8 — API Gateway

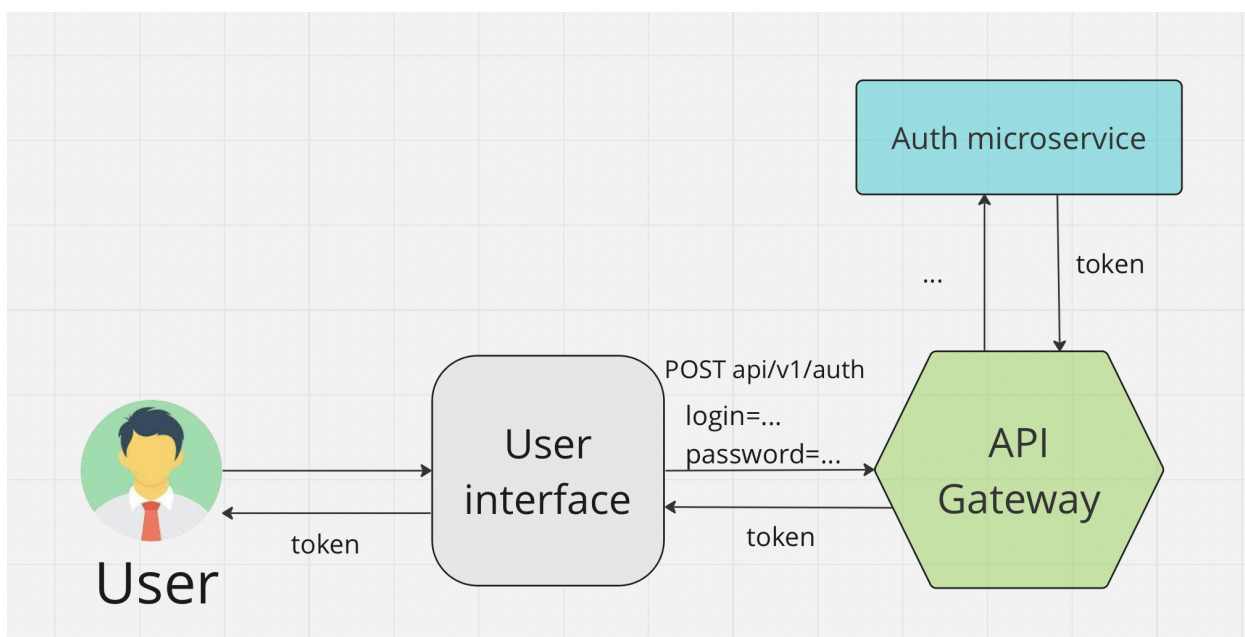


Рисунок 9 — Аутентификация с API Gateway

В таком случае процесс аутентификации можно описать так (см. рисунок 9):

1. Пользователь вводит логин и пароль;
2. Информация приходит на API Gateway;

3. API Gateway видит путь *api/v1/auth* и отправляет пришедшие ему данные на микросервис «Auth»;
4. Микросервис «Auth» проверяет пользователя, выдает токен и передает его в API Gateway;
5. API Gateway передает токен пользователю;

4.3.6 Авторизация и JWT

Для пользования нашим приложением, клиенты должны аутентифицировать свои учетные данные. Однако данный механизм никак не разделяет аутентифицированных пользователей по правам доступа к ресурсам. Для этого необходимо реализовать механизм авторизации.

JSON Web Token (JWT) — это открытый стандарт, определяющий компактный и автономный способ безопасной передачи информации между сторонами в виде объекта JSON. Безопасность подтверждается цифровой подписью. [24]

В основе JWT лежат:

- Заголовок (header) — информация про алгоритм шифрования;
- Полезная нагрузка (payload) — данные, которые хотим передать от одного микросервиса к другому;
- Сигнатура (signature) — то, за счет чего обеспечивается безопасность. Информация шифруется по выбранному алгоритму хеширования через секретный ключ, заранее известному всем микросервисам, которым нужна авторизация.

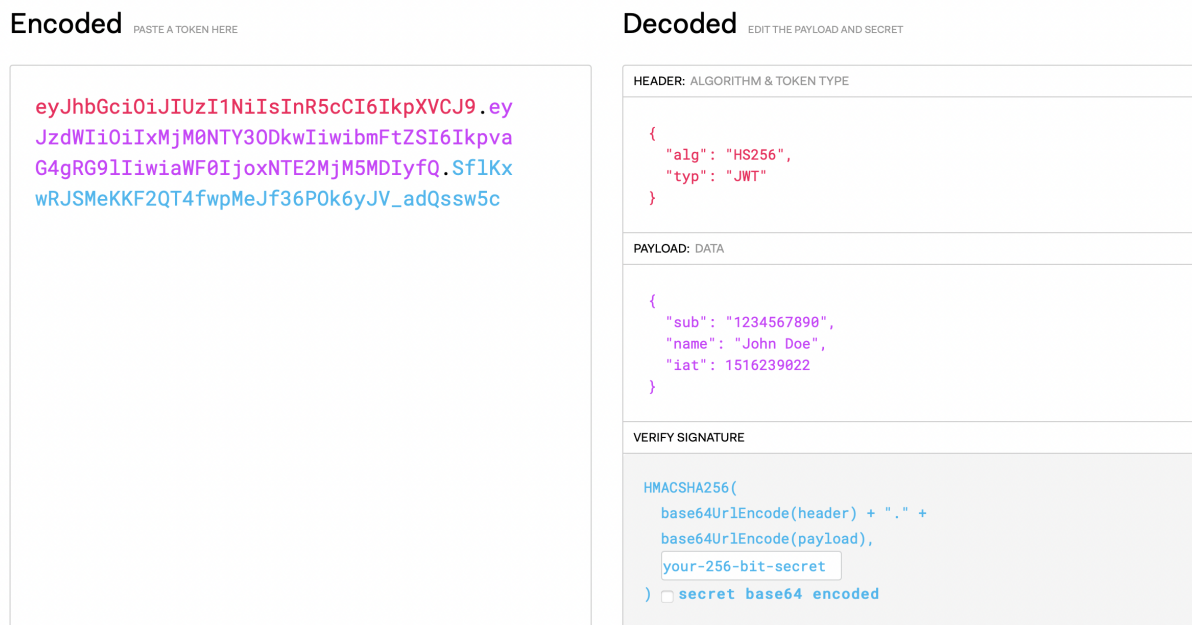


Рисунок 10 — Пример JWT в закодированном и декодированном видах. Источник: [24]

JWT передается в заголовках HTTP-запросов в зашифрованном (encoded) виде - в виде обычной строки (см. рисунок 10). «Получатель» сможет декодировать (decode) JWT для получения полезной информации при наличии секретного ключа.

4.4 Итоговая архитектура

Для создания архитектуры, нам осталось только определить необходимые сервисы (результат главы 4.1). Так как они являются REST API (результат главы 4.2), то каждый микросервис должен представлять собой ресурс. Из функциональных требований (глава 3.1) и выбранного архитектурного паттерна (4.3.5), формируем следующий список компонентов:

1. API Gateway;
2. Микросервис полей;
3. Микросервис аутентификации;
4. Микросервис профилей;
5. Микросервис метаданных, являющийся также ETL-системой;

Получившаяся архитектура отображена на рисунке 11:

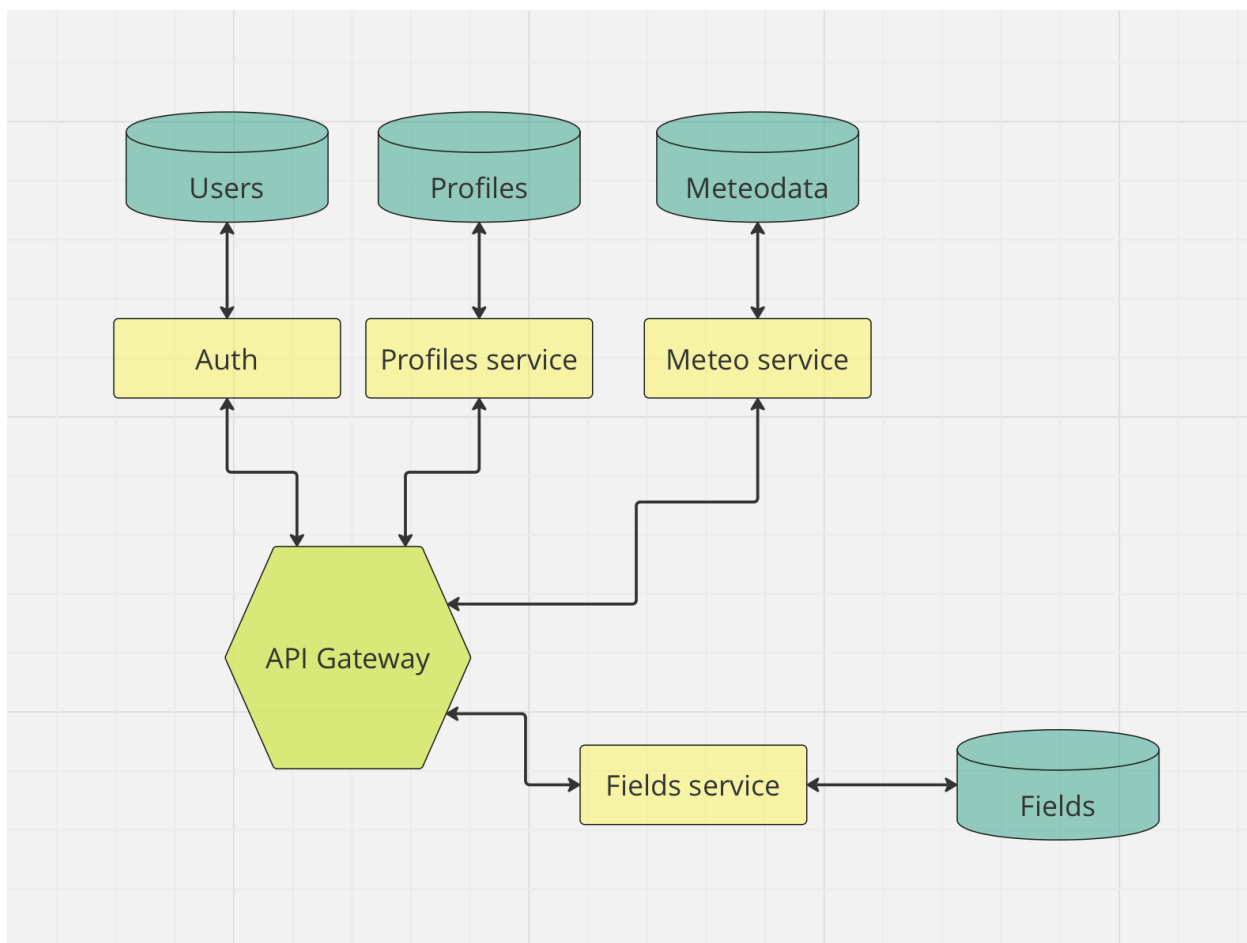


Рисунок 11 — Архитектура серверной части веб-приложения «Цифровой двойник сельскохозяйственного поля».

ER-диаграмма распределенной базы данных отображена на рисунке 12:

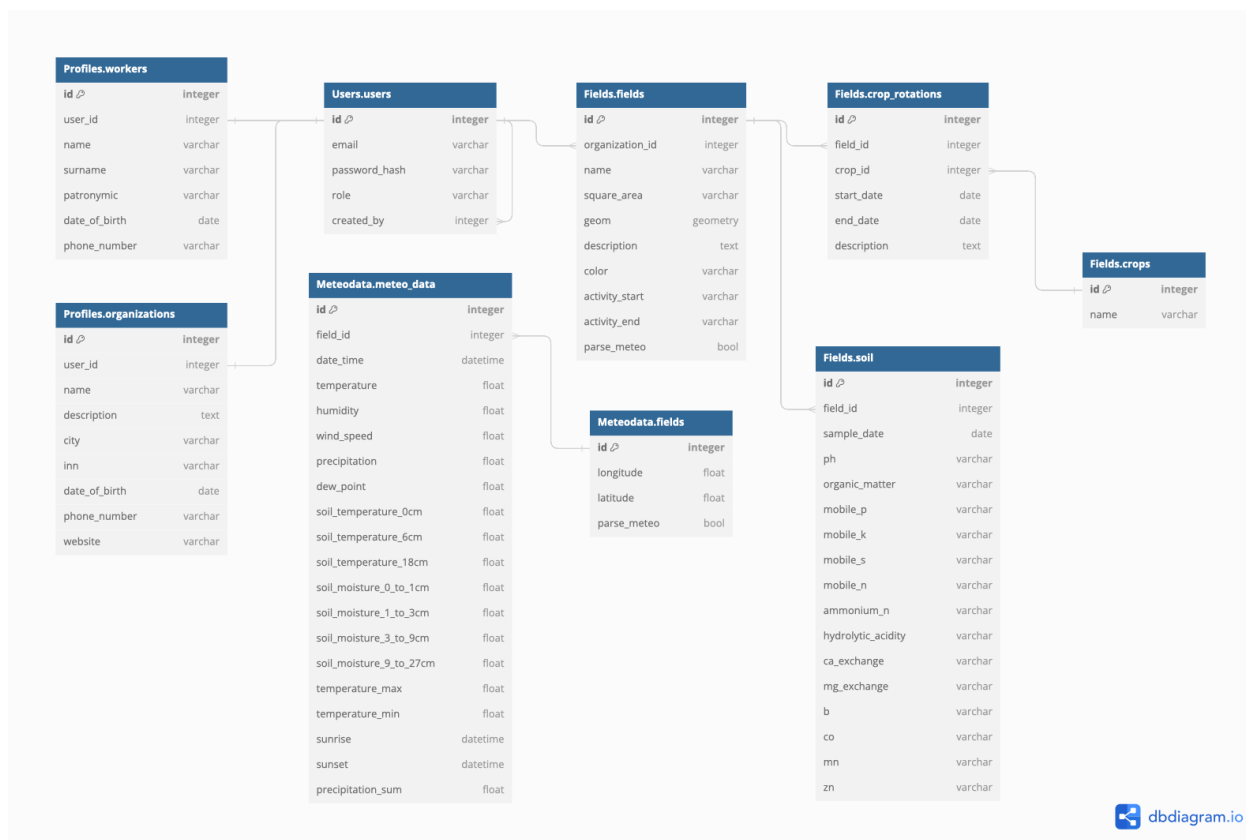


Рисунок 12 — ER-диаграмма распределенной базы данных веб-приложения «Цифровой двойник сельскохозяйственного поля».

Следующим шагом будет выбор как общих технологий, так и конкретных для реализации соответствующих сервисов.

5 ПРОГРАММНАЯ РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ

5.1 Выбор технологий

В качестве основного языка программирования для разработки микросервисов был выбран Python. Python является одним из наиболее популярных языков программирования для веб-разработки. Есть несколько причин, почему Python является оптимальным вариантом:

1. Простота и удобство использования: Python имеет простой и понятный синтаксис, который легко читать и писать. Это делает его идеальным языком для написания микросервисов, основной нагрузкой в которых являются I/O-bound задачи.
2. Большое количество библиотек и фреймворков: Python имеет огромное сообщество разработчиков, которые создали множество библиотек и фреймворков, которые упрощают и ускоряют процесс разработки веб-приложений. Некоторые из наиболее популярных фреймворков для веб-разработки на Python: Django, FastAPI, Flask.
3. Большое количество инструментов для тестирования: Python имеет множество инструментов для тестирования, которые помогают разработчикам создавать качественное и надежное программное обеспечение.

В качестве фреймворка был выбран FastAPI [25] – веб-фреймворк для реализации REST API. Одним из его преимуществ является поддерживаемая асинхронность “из коробки”, которая так важна для быстрого выполнения I/O-bound задач, которыми в свою очередь являются основные функции сервисов аутентификации и профилей пользователей. Также важно упомянуть о поддерживаемой валидации данных в FastAPI, достигаемой использованием библиотеки Pydantic. [26]

В качестве СУБД был выбран PostgreSQL.

5.2 Микросервис аутентификации

Одной из основных задач при разработке данного сервиса являлось составление структуры JWT:

```
class TokenPayloadSchema(BaseModel):  
    iat: datetime  
    exp: datetime  
    sub: str  
    role: str  
    email: str  
    org: int
```

Так, помимо передачи стандартных полей, таких как *iat*, *exp* и *sub*, мы дополнительно передаем роль пользователя - для авторизации в каждом сервисе, адрес его электронной почты - для корректной работы email-сервиса, а также id его организации (для организации значения *sub* и *org* будут совпадать).

Для обеспечения безопасности пользовательских данных, пароли хранятся в зашифрованном с помощью алгоритма хэширования bcrypt [27] формате. Таким образом, даже при получении злоумышленниками доступа к базе данных с пользовательскими данными, пароли не смогут быть расшифрованы.

Сервис был разработан на базе фреймворка FastAPI с использованием СУБД PostgreSQL. В последствии сервис был обернут в docker-контейнер.

На рисунке 13 представлен API-контракт разработанного сервиса:

auth ^			
POST	/api/auth/register	Register	✓
POST	/api/auth/token	Login	✓
POST	/api/auth/introspect	Introspect Token	🔒 ✓
users ^			
POST	/api/auth/users	Create User	🔒 ✓
GET	/api/auth/users/me	Get User	🔒 ✓
PUT	/api/auth/users/me	Update User	🔒 ✓
DELETE	/api/auth/users/me	Delete User	🔒 ✓
GET	/api/auth/users/workers	Get Created Users	🔒 ✓
DELETE	/api/auth/users/workers/{id}	Delete Worker	🔒 ✓

Рисунок 13 — API-контракт сервиса аутентификации, сгенерированный инструментом Swagger.

Ссылка на исходный код: <https://github.com/AgroScience-Team/auth-service>.

5.3 Микросервис метеоданных

5.3.1 Выбор внешней системы для получения данных

Из требований 3.1 имеем следующий список необходимых метеоданных для получения из внешних систем:

- Текущая температура;
- Текущая влажность;
- Текущий уровень осадков;
- Текущая точка росы;
- Текущая скорость ветра;
- Текущая температура почвы на глубине/глубинах 0-18 см;
- Текущая влажность почвы на глубине/глубинах от 0-27 см;
- Время восхода солнца;
- Время заката;
- Сумма осадков за сутки;

На сегодняшний день существует множество внешних систем, предоставляющих API для получения метеоданных. Рассмотрим популярные варианты.

1. *OpenWeather* [28] - онлайн сервис, который предоставляет платный (есть функционально ограниченная бесплатная версия) API для доступа к данным о текущей погоде, прогнозам и историческим данным.

Плюсы:

- Широкий функционал в платной версии сервиса;
- Возможность получения данных в разных форматах;

Минусы:

- API не работает на территории РФ;
- Ограниченное число запросов к API в бесплатной версии: 1000 в день. Доступны разные тарифы;
- Нет данных о влажности и температуре почве в бесплатной версии;

2. *API Яндекс Погоды* [29] - API, предоставляющее погодные и климатические данные. Передаёт текущие данные и прогноз по 150+ погодным параметрам: от базовых (температура, осадки) до нишевых, которые важны для конкретного бизнеса.

Плюсы:

- Широкий функционал сервиса;
- Возможность получения данных в разных форматах;
- Быстрое время ответа: до 500мс в 95 процентах от всех запросов;

Минусы:

- Отсутствие бесплатной версии: от 54.000 рублей за месяц использования. Доступны разные тарифы;

3. *Open-Meteo* [30] - API погоды с бесплатным доступом для некоммерческого использования.

Плюсы:

- Широкий функционал и гибкая настройка: есть возможность получить все необходимые метеоданные;
- Цена: бесплатно;
- Нет ограничения на количество запросов;
- Возможность получения данных в разных форматах;

Минусы:

- Для решения поставленной задачи недостатков выявлено не было;

Таким образом, Open-Meteo был выбран источником для загрузки метеоданных.

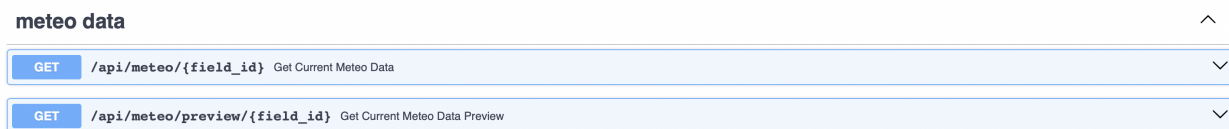
5.3.2 Реализация сервиса

Сервис был разработан на базе фреймворка FastAPI с использованием СУБД PostgreSQL. В последствии сервис был поднят в docker-контейнере.

Для парсинга метеоданных использовалась библиотека HTTPX [31] - полнофункциональный HTTP-клиент для Python 3, который предоставляет синхронные и асинхронные API, а также поддерживает протоколы HTTP/1.1 и HTTP/2.

В качестве библиотеки для реализации планировщика задач был выбран APScheduler [32]. Данной библиотекой легко пользоваться, а также она не требует подключения дополнительных компонентов, таких как очередь задач и результирующее хранилище.

По итогу сервис представляет из себя одновременно REST API и ETL-систему. На рисунке 14 представлен API-контракт разработанного сервиса:



The image shows a Swagger API contract for 'meteo data'. It contains two endpoints:

meteo data	
GET	/api/meteo/{field_id} Get Current Meteo Data
GET	/api/meteo/preview/{field_id} Get Current Meteo Data Preview

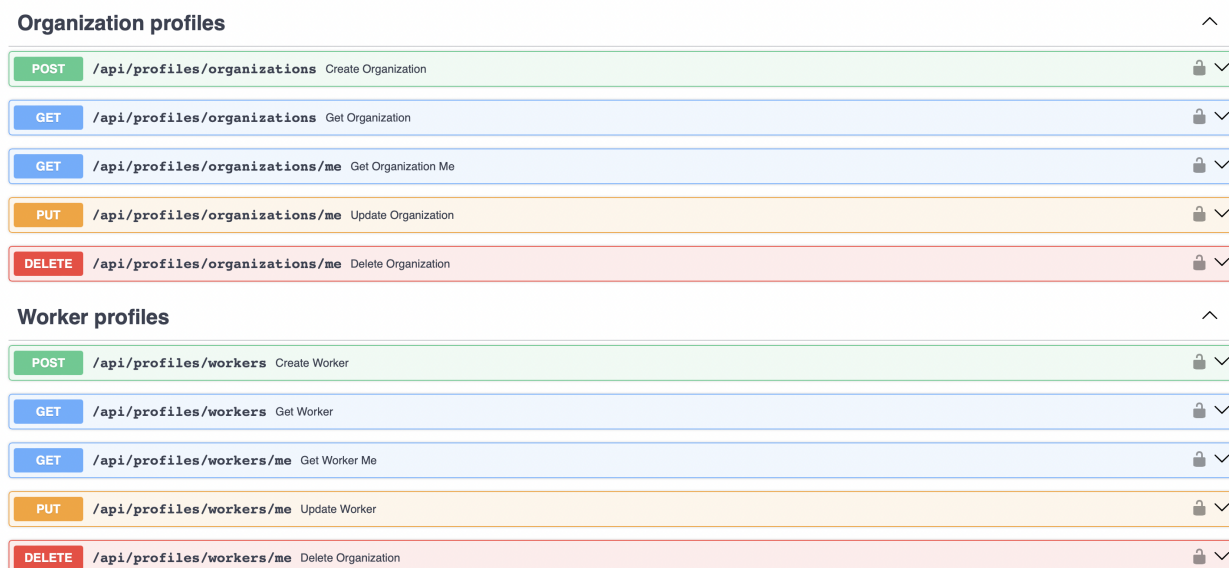
Рисунок 14 — API-контракт сервиса метео данных, сгенерированный инструментом Swagger.

Ссылка на исходный код: <https://github.com/AgroScience-Team/meteo-service>.

5.4 Микросервис профилей

Сервис был разработан на базе фреймворка FastAPI с использованием СУБД PostgreSQL. В последствии сервис был поднят в docker-контейнере.

На рисунке 15 представлен API-контракт разработанного сервиса:



The image shows a Swagger API contract for 'Organization profiles' and 'Worker profiles'. It contains ten endpoints:

Organization profiles	
POST	/api/profiles/organizations Create Organization
GET	/api/profiles/organizations Get Organization
GET	/api/profiles/organizations/me Get Organization Me
PUT	/api/profiles/organizations/me Update Organization
DELETE	/api/profiles/organizations/me Delete Organization

Worker profiles	
POST	/api/profiles/workers Create Worker
GET	/api/profiles/workers Get Worker
GET	/api/profiles/workers/me Get Worker Me
PUT	/api/profiles/workers/me Update Worker
DELETE	/api/profiles/workers/me Delete Organization

Рисунок 15 — API-контракт сервиса аутентификации, сгенерированный инструментом Swagger.

Ссылка на исходный код: <https://github.com/AgroScience-Team/profiles-service>.

5.5 API Gateway

Для реализации микросервисного паттерна API-Gateway был выбран Nginx. Главной причиной для выбора Nginx являлась простота его настройки в качестве обратного прокси-сервера [33].

При каждом пользовательском запросе, подразумевающим обращение к сервисам `fields-service`, `profiles-service` и `meteo-service`, nginx отправлял подзапрос `api/auth/introspect` в сервис аутентификации для валидации отправленного токена в заголовке первоначального запроса. Если отправленный токен проходил валидацию успешно, то дальше отправлялся первоначальный пользовательский запрос в соответствующий микросервис.

Для поднятия `api-gateway` необходимо запустить все `docker`-контейнеры остальных сервисов в одной `docker`-сети. При этом каждый сервис должен использовать уникальный порт.

Ссылка на исходный код: <https://github.com/AgroScience-Team/api-gateway>.

6 ТЕСТИРОВАНИЕ И РАЗВЁРТЫВАНИЕ НА УДАЛЁННОЙ МАШИНЕ

Во время разработки проекта писались unit-тесты с помощью библиотеки Pytest [34]. Это помогло выявить и устранить ошибки на ранних этапах разработки. Кроме того, выполнялось ручное тестирование с помощью инструментов Swagger [35] и Postman [36]. Это позволило проверить правильность работы API, а также ручные сценарии использования приложения. Благодаря комбинации автоматизированных и ручных тестов было значительно повышено качество приложения и появилась уверенность в его надёжности перед развёртыванием на сервере.

Для развёртывания приложения на удаленной машине использовался протокол SSH для безопасного соединения и управления сервером. После установки необходимых зависимостей и настройки конфигурации был произведен запуск приложения.

Сейчас доступ к приложению можно получить по ссылке <http://www.agromelio.ru>. Пользователи могут обращаться к приложению просто перейдя по данной ссылке и пользоваться его функционалом без необходимости устанавливать какие-либо дополнительные программы.

7 ЗАКЛЮЧЕНИЕ

7.1 Результаты работы

В современном мире происходит активная интеграция информационных технологий в сельское хозяйство. Широкое применение находят приложения для точного земледелия, с помощью которых ведется хранение различной информации о сельхозугодьях в понятном для человека виде.

В ходе выполнения поставленных задач из 1.3 были выполнены следующие подзадачи:

1. Обзор исследований и существующих решений:
 - Проведено исследование современного состояние проблемы использования информационных технологий в сельском хозяйстве;
 - Проведен анализ современных программных решений для точного земледелия;
2. Формирование требований к приложению:
 - Были сформированы функциональные и нефункциональные требования к приложению;
3. Проектирование архитектуры серверной части приложения:
 - Изучены и выбраны оптимальные паттерны проектирования серверной части приложения. Выбор микросервисного паттерна позволил выполнить нефункциональное требование 3.2 о расширяемости системы.
 - Изучены различные способы межсервисного взаимодействия, выбран оптимальный вариант для данной задачи - REST API;

- Были исследованы различные способы аутентификации и авторизации в микросервисных приложениях, был выбран оптимальный вариант - проксирование запросов при помощи API Gateway с авторизацией в каждом сервисе;
- Разработана схема итоговой архитектуры серверной части приложения;
- Разработана ER-диаграмма распределенной базы данных;

4. Программная разработка серверной части приложения:

- Выбраны общие технологии для разработки бекенда приложения, а также технологии для реализации точечных подзадач;
- Разработаны микросервисы аутентификации, профилей, метеоданных и API Gateway;
- Для микросервиса аутентификации был выбран алгоритм шифрования пользовательских паролей для безопасного хранения. Также была разработана схема JWT.
- Для микросервиса метеоданных проведено исследование внешних систем для сбора данных. Выбран оптимальный вариант.

5. Тестирование и развертывание приложения на удалённом сервере:

- Написаны unit-тесты для отдельных функций сервисов;
- Проведено ручное тестирование с помощью инструментов Swagger и Postman;
- Все микросервисы были подняты в docker-контейнерах в рамках одной docker-сети и объединены сервисом api-gateway.
- Приложение было развернуто на удалённой машине.

7.2 Дальнейшее развитие

Благодаря выбранному архитектурному решению, дальнейшее развитие сервиса может ограничиваться фантазией заказчика или разработчика.

Можно рассмотреть добавление функционала по хранению и обработке ортофотопланов. Ортофотопланы представляют собой изображения, полученные в результате аэрофотосъемки и прошедшие специальную обработку для исключения искажений. Эти данные могут быть ценными для сельскохозяйственных предприятий, позволяя им более эффективно планировать работу на полях, определять места повреждения растений или оценивать объем урожая.

Следующим шагом может быть внедрение сервиса рассылки сообщений, который позволит агрономам и фермерам оперативно уведомляться о важных событиях и новостях, касающихся их деятельности. Такие уведомления могут касаться изменения погодных условий или напоминаний о скором конце срока текущего посева.

Кроме того, для улучшения аналитики и прогнозирования процессов на полях, можно добавить функцию получения метеоданных за выбранный период. Это позволит агрономам анализировать погодные условия в разные периоды времени, выявлять тенденции и коррелировать их с результатами урожайности. Автоматическое построение графиков сделает этот процесс более удобным и эффективным, освобождая время специалистов для других задач и принятия важных решений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Amber Anderson*. Introduction to Soil Science. — Iowa State University Digital Press, 2023. — P. 165–170.
2. *Якушев В. В.* ТОЧНОЕ ЗЕМЛЕДЕЛИЕ: ТЕОРИЯ И ПРАКТИКА. — СПб.:ФГБНУ АФИ, 2016. — С. 364.
3. *Генин В.* Что такое точное земледелие и как начать его использовать. [Электронный ресурс]. — 2023. — URL: <https://blog.onesoil.ai/ru/what-is-precision-farming>.
4. ТОЧНОЕ ЗЕМЛЕДЕЛИЕ КАК ОДИН ИЗ АСПЕКТОВ ЦИФРОВИЗАЦИИ СЕЛЬСКОГО ХОЗЯЙСТВА. [Электронный ресурс] // Вестник Курской государственной сельскохозяйственной академии. — 2022. — URL: <https://cyberleninka.ru/article/n/tochnoe-zemledelie-kak-odin-iz-aspektov-tsifrovizatsii-selskogo-hozyaystva>.
5. *Статья компании "Б-Истокское РТПС"*. Точное земледелие в современном сельском хозяйстве. Плюсы и минусы технологии. [Электронный ресурс]. — 2021. — URL: <https://istokrtps.ru/stati/tochnoe-zemledelie>.
6. *Kwang Soo Kima, Byung Hyun Yooa, Vakhtang Sheliac, Cheryl H. Porterc, Gerrit Hoogenboomc*. START: A data preparation tool for crop simulation models using web-based T soil databases // Computers and Electronics in Agriculture. — 2018. — URL: <https://www.sciencedirect.com/science/article/abs/pii/S0168169917313807?via%3Dihub>.
7. *В.А. Семёнов, В.И. Мирный*. Принципы адаптации технологий возделывания сельскохозяйственных культур // Программирование урожаев сельскохозяйственных культур на Северо-Западе РСФСР. Л. — 1988. — С. 4–9.

8. Митрофанов Е. П., Петрушин А. Ф., Митрофанова О. А. Использование данных аэрофотосъемки для обоснования прецизионных агроприемов применения агрохимикатов // Применение средств дистанционного зондирования Земли в сельском хозяйстве. — 2018. — С. 212—217.
9. Шнаар Д., Захаренко А., Якушев В. П., Арефьев Н. В., Ауернхаммер Х., Брунш Р., Вагнер П., Вартенберг Г., Венкель К-О., Вернер А., Войтюк Д., Герхардс Р., Даммер К., Домен Б., Каленская С., Кауфманн О., Клочков А., Кохан С., Ляйтхольд П., Лысов А. К., Гончаров Н., Мазиров М. А., Михайленко И., Нехай А., Неш Э., Нордмайер Х., Реклебен И., Хербст Р., Шеин Е., Шуманн П., Элерт Д., Эльмер Ф. Точное сельское хозяйство // Санкт-Петербург: предприятие «Павел». — 2009. — С. 397.
10. S. Chaudhary, V. Kumar. Service Oriented Architecture and Spatial Data Integration for Agro Advisory Systems // Geospatial Infrastructure, Applications and Technologies: India Case Studies. — 2018. — P. 185–199. — URL: https://link.springer.com/chapter/10.1007/978-981-13-2330-0_15.
11. Сайт SmartAGRO [Электронный ресурс]. — URL: <https://smartagro.ru>.
12. Сайт Cropwise Operations [Электронный ресурс]. — URL: <https://operations.cropwise.com>.
13. Сайт OneSoil [Электронный ресурс]. — URL: <https://onesoil.ai/ru>.
14. Сайт QGIS [Электронный ресурс]. — URL: <https://www.qgis.org/ru/site/>.
15. Сайт АгроСигнал [Электронный ресурс]. — URL: <https://agrosignal.com>.
16. Сайт АгроМон [Электронный ресурс]. — URL: <https://agromon.ru>.
17. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. — Addison-Wesley, 1994. — P. 16.

18. Microservices vs. Monolithic Architectures. [Электронный ресурс]. — 2023. — URL: <https://www.baeldung.com/cs/microservices-vs-monolithic-architectures>.
19. *Eric Schabell*. Monolith vs. Microservice Architecture for Software Delivery. [Электронный ресурс]. — 2023. — URL: <https://thenewstack.io/monolith-vs-microservice-architecture-for-software-delivery/>.
20. *Mozilla Developer Network contributors*. An overview of HTTP [Электронный ресурс]. — 2023. — URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>.
21. *Дергачев А.М., Кореньков Ю.Д., Логинов И.П., Сафронов А.Г.* Технологии веб-сервисов. — СПб: Университет ИТМО, 2021. — С. 46—65.
22. *S.Rajarajeswari, Ms.A.Maria Stella*. A REVIEW OF AUTHENTICATION AND AUTHORIZATION METHODS [Электронный ресурс] // International Journal of Computer Science and Information Technology Research. — 2019. — С. 78. — URL: <https://www.researchpublish.com/upload/book/A%20REVIEW%20OF%20AUTHENTICATION-7836.pdf>.
23. *Казаков О.* Аутентификация и авторизация в проекте с микросервисной архитектурой: стратегии, практический пример. [Электронный ресурс]. — 2023. — URL: <https://habr.com/ru/companies/spectr/articles/715290/>.
24. Introduction to JSON Web Tokens [Электронный ресурс]. — URL: <https://jwt.io/introduction>.
25. FastAPI documenation. [Электронный ресурс]. — URL: <https://fastapi.tiangolo.com>.
26. Pydantic documentation. [Электронный ресурс]. — URL: <https://docs.pydantic.dev/latest/>.
27. bcrypt. [Электронный ресурс]. Wikipedia. — 2024. — URL: <https://en.wikipedia.org/wiki/Bcrypt>.
28. Сайт OpenWeather [Электронный ресурс]. — URL: <https://openweathermap.org>.

29. Сайт API Яндекс Погоды [Электронный ресурс]. — URL: <https://yandex.ru/dev/weather/>.
30. Сайт Open-Meteo [Электронный ресурс]. — URL: <https://open-meteo.com>.
31. Документация HTTPX. [Электронный ресурс]. — URL: <https://www.python-httpx.org>.
32. Документация APScheduler [Электронный ресурс]. — URL: <https://apscheduler.readthedocs.io>.
33. NGINX Reverse Proxy. [Электронный ресурс]. — URL: <https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/>.
34. Документация библиотеки для тестирования Pytest [Электронный ресурс]. — URL: <https://docs.pytest.org/en/8.2.x/>.
35. Документация Swagger [Электронный ресурс]. — URL: <https://swagger.io>.
36. Документация Postman [Электронный ресурс]. — URL: <https://www.postman.com>.