МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра анализа данных и искусственного интеллекта**

**КУРСОВАЯ РАБОТА**

**РАЗРАБОТКА МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ НА JAVA С ИСПОЛЬЗОВАНИЕМ ФРЕЙМВОРКА SPRING**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.В. Сидоренко

(подпись)

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем курс 3

Научный руководитель

канд. физ.-мат. наук, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г.В. Калайдина

(подпись)

Нормоконтролер

канд. физ.-мат. наук, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г.В. Калайдина

(подпись)

Краснодар

2023

**РЕФЕРАТ**

Курсовая работа 70 с., 22 рис., 6 табл., 21 источн., 3 прил.

МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, JAVA, SPRING, SPRING BOOT, SPRING CLOUD, КЭШИРОВАНИЕ ДАННЫХ, REDIS, KAFKA, ГОРИЗОНТАЛЬНОЕ МАСШТАБИРОВАНИЕ, REACT.JS, RESTFUL API, POSTGRESQL, MONGODB

Объектом исследования данной работы является разработка и масштабирование микросервисной архитектуры на языке программирования Java с использованием фреймворка Spring, а также дополнительного стека технологий, необходимого для поддержания высоконагруженной системы.

Целью работы является разработка эффективной, масштабируемой и отказа устойчивой системы, основанной на принципах микросервисной архитектуры, которая сможет обеспечить бесперебойное уведомления пользователей о чрезвычайных ситуациях.

В процессе работы будут использованы UML-диаграммы для визуализации и описания системы. А также приложение Figma, для создания дизайна пользовательского интерфейса. Системный стек: Java, Spring, Redis, Kafka, Docker. Для лучшего проектирования приложения будет использована методология Domain-Driven Design (DDD). Взаимодействие между микросервисами будет осуществляться посредством RESTful API. Используемые базы данных PostgreSQL, MongoDB. Для разработки фронт-энда будет использован фреймворк React и React native для мобильного приложения.

В результате выполнения данной работы будет разработана эффективная и масштабируемая информационная система, основанная на принципе микросервисной архитектуры, способная обеспечить бесперебойное уведомление пользователей о чрезвычайных ситуациях. Весь процесс разработки будет документирован и описан в научной работе.

**СОДЕРЖАНИЕ**

Введение 4

1 Анализ рынка 6

1.1 Международные аналоги 6

1.1.1 FEMA (Federal Emergency Management Agency) Alert System 7

1.1.2 J-Alert 8

1.1.3 EU-Alert 9

1.1.4 Nextdoor 9

1.2 Российские системы оповещения10

1.2.1 Система-112 10

1.2.2 Система оповещения населения об опасностях 11

1.3 Итоги анализа рынка11

1.4 Современный ответ на старую проблему12

**ВВЕДЕНИЕ**

В современном информационном обществе остро встает вопрос об оповещении пользователей о чрезвычайных ситуациях. Быстрая доставка информации играет критическую роль для защиты жизни и здоровья людей, а также для минимизации материальных потерь. Кризисные ситуации, будь то природные катаклизмы или человеческие действия, могут застать врасплох каждого. В связи с этим возникает необходимость в разработке эффективной системы уведомлений.

Такая система должна быть быстрой и отказоустойчивой, ведь от её работы могут зависеть жизни. Именно поэтому так важно качественно подойти к разработке её архитектуры.

Основной целью данной научной работы является проектирование микросервисной архитектуры, способной выдержать большие нагрузки, и обеспечить пользователей оперативной доставкой уведомлений о чрезвычайных ситуациях, а также разработка пользовательского интерфейса, который будет удобен и интуитивно понятен каждому.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1) Изучить основы микросервисной архитектуры и принципы ее организации.

2) Провести аналитику и декомпозицию предметной области информационной системы.

3) Исследовать фреймворк Spring и его роль в разработке микросервисов.

4) Спроектировать работу каждого сервиса, которые будут реализованы средствами Spring Boot и Spring Cloud.

5) Разобраться с паттернами Энтерпрайз разработки, для правильной организации дизайна системы.

6) Разработать правильную организацию слоя данных.

7) Разработать функциональность кэширования данных с использованием технологии Redis.

8) Исследовать брокеры сообщений, и организовать общение сервисов через них.

Теоретическая значимость данной работы заключается в исследовании принципов организации микросервисной архитектуры на Java с использованием фреймворка Spring, а также других инфраструктурных технологий необходимых для разработки такой потенциально высоконагруженной системы. Полученные результаты будут использованы в дальнейшем при реализации данной системы. Кроме того, навыки и знания, полученные в процессе исследования, буду полезны для проектирования других подобных систем.

Практическая значимость работы состоит в возможности создания удобной платформы для обмена уведомлениями о чрезвычайных ситуациях, способной оперативно доставлять информацию пользователям. Разработанное приложение может быть использовано различными организациями и государственными структурами, занимающимися обеспечением безопасности и защитой населения.

В данной работе будет использована методология Domain-Driven Design (DDD) и микросервисный подход с применением фреймворка Spring. Также планируется использование нескольких баз данных для правильно распределённой организации хранения информации о пользователях и шаблонах уведомлений.

В следующих главах будут более подробно рассмотрены основные аспекты разработки и масштабирования высоконагруженных распределённых систем.

1. **Анализ рынка**

В первой главе основной части работы приведены результаты анализа как мирового, так и Российского рынка систем оповещающих пользователей о чрезвычайных ситуациях. Разобрано подробно с какими плюсами и минусами сталкиваются пользователи этих систем.

В первой части первой главы представлено несколько международных аналогов, таких как FEMA Alert System в США, J-Alert в Японии и EU-Alert в странах Европейского союза.

Во второй части этой главы исследование сосредоточено на российских аналогах, таких как система-112, которая объединяет различные службы экстренного реагирования, и система оповещения населения об опасностях, разработанная МЧС России. Обе системы предоставляют оповещение о чрезвычайных ситуациях через различные каналы связи, такие как SMS, громкоговорители, сирены и телевизионные и радиоэфиры.

Затем в заключении первой главы, рассмотрено чем отвечает разработанная мной приложение на плюсы и минусы систем, рассмотренных выше.

В итоге, данная глава основной части работы позволит точно оценить преимущества и недостатки моего приложения по сравнению с аналогами, а также выбрать оптимальный подход к его реализации.

* 1. **Международные аналоги**

Так как одной из основных функций государства являться защита своего народа, то не удивительно при анализе рынка каких-либо систем, связанных с оповещением о чрезвычайных ситуациях, было обнаружено, что крупнейшие системы такого рода, являются на прямую или косвенно государственными разработками, а у любых государственных программ, в отличие от бизнес-решений, ограниченное финансирование, что в последствие выливается в известные всем недостатки.

* + 1. **FEMA (Federal Emergency Management Agency) Alert System**

Федеральное агентство по чрезвычайным ситуациям - американская система оповещения о чрезвычайных ситуациях. Система использует SMS, телевизионные и радиоэфиры для отправки оповещений.

Когда вы добавляете новое местоположение для получения оповещений, вы получаете полный контроль над типами оповещений для мониторинга: наводнения, затопления прибрежных районов и озер, суровые погодные условия (грозы и торнадо), тропическая погода (ураганы и тайфуны), зимняя погода (снег, лед, ледяной дождь), лавины, пожары, экстремальные температуры, морская погода, оповещения об общественных опасностях и многое другое.

В отличие от большинства аналогичных приложений, FEMA также предоставляет экстренные оповещения об эвакуации, гражданской опасности, похищениях детей, опасных материалах, атомных электростанциях, радиационной опасности, перебоях в работе телефона 911, беспорядках, взрывах и многом другом.

FEMA также является приложением для обеспечения готовности к стихийным бедствиям, поскольку здесь представлены советы по технике безопасности в чрезвычайных ситуациях, напоминания о тестировании дымовых сигнализаций и обновлении аварийных комплектов, ресурсы для стихийных бедствий, такие как убежища, и многое другое.

Основным недостатком этой системы, как и большинства подобных, является ограничение в использовании только для оповещений на территории США. Помимо того, пользователи жалуются на устаревший и не удобный пользовательский интерфейс (Рисунок 1.1). В добавок отметим наличие только старых вариантов оповещения, которые не всегда удобны в современных реалиях.

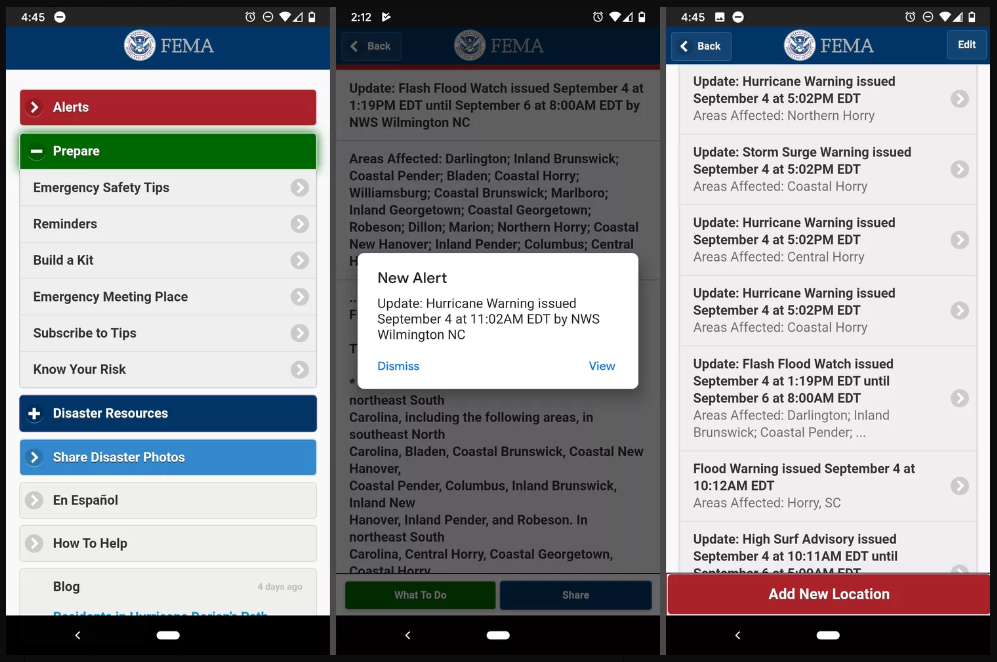


Рисунок 1.1 – Интерфейс приложения «FEMA»

**1.1.2 J-Alert**

Система оповещения о чрезвычайных ситуациях на базе спутника в Японии. Использует различные каналы связи, такие как мобильные телефоны, телевизионные и радиоэфиры, чтобы доставлять сообщения о надвигающихся опасностях.

Из плюсов можно выделить удивительную скорость реагирования, что очень полезно в сейсмически активном регионе, таком как Япония.

На бумаге эта система должна была быть внедрена очень быстро и повсеместно, но на деле, из-за большой консервативности японской культуры, какой бы развитой страной она не была, со всем, где государство встречается с цифровизацией, в Японии очень плохо. Ни о каком удобстве использования не может быть и речи. Приложений или веб-интерфейсов для пользовательского доступа к системе нет, используются аналоговые варианты связи.

**1.1.3 EU-Alert**

Система оповещения о чрезвычайных ситуациях в странах Европейского союза. Она основана на стандарте Cell Broadcast ([трансляция по сотовой связи](https://en.wikipedia.org/wiki/Cell_Broadcast)) и позволяет доставлять сообщения о чрезвычайных ситуациях на сотовые телефоны пользователей в заданной географической области.

Является стандартом схожим с Американской системой оповещения. Следовательно, имеет схожие отрицательные стороны, но помимо минусов, заимствованных у заокеанских коллег, имеет свои уникальные недостатки. Например, полное отсутствие какого-либо пользовательского интерфейса, оповещения приходят по SMS.

Нельзя не отметить, что у большинства европейских стран, есть своя реализация данного стандарта. Не углубляясь в каждую, можно утверждать, что имея свои локальные плюсы и минусы, эти системы, в общем и целом, склонны иметь одни и те же минусы, описанные выше.

**1.1.4 Nextdoor**

Заключительная в этой главе международная система, которая представляет собой социальную сеть, основанную на место положении. В отличие от аналогов, описанных выше, является не государственной и более современной разработкой.

Приложение имеет приятный и интуитивно понятный пользовательский интерфейс (Рисунок 1.2), публиковать экстремальные оповещения может любой желающий.  
 Из минусов: приложение не является строго организованной системой, для оповещения пользователей, а скорее является платформой для удобного уведомления соседей, или рассылкой любой информации на конкретный район.

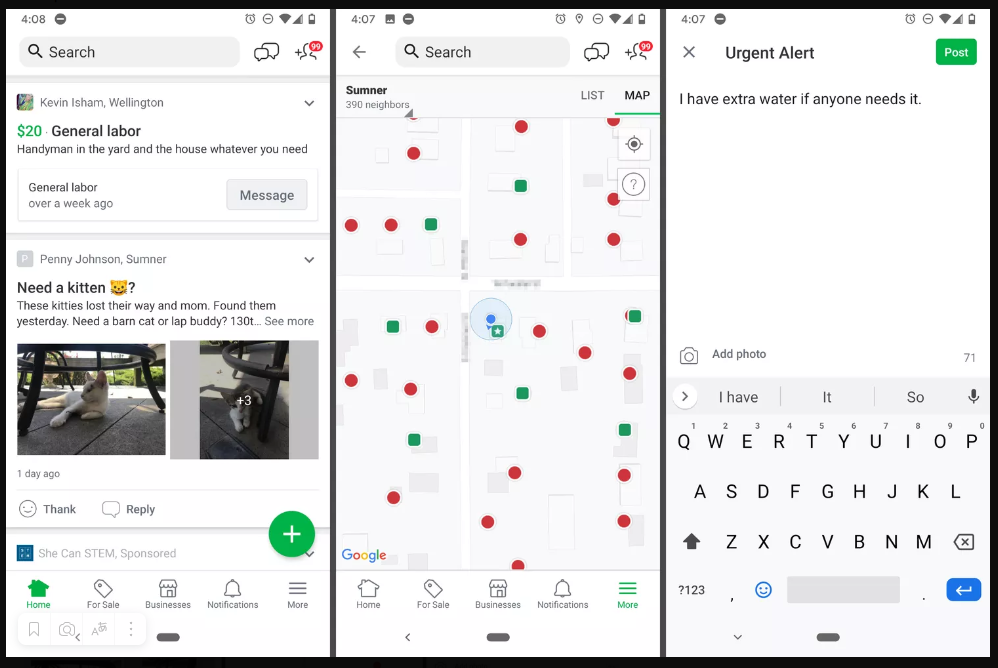


Рисунок 1.2 – Интерфейс приложения «Nextdoor»

* 1. **Российские системы оповещения**

Проанализировав российский рынок, столкнулся с ещё более не воодушевляющими результатами. Как и в консервативной Японии, у нас тоже есть только государственные программы, реализующие оповещение пользователей о чрезвычайных ситуациях. Как и ожидалось, эти системы очень далеки от необходимой современной цифровизации.

* + 1. **Система-112**

Российская система оповещения о чрезвычайных ситуациях, которая объединяет различные службы экстренного реагирования. Система основана на использовании телефонного номера 112, который можно набрать для вызова экстренных служб.

Очевидно, этих мер недостаточно в современном мире для обеспечения точечного, быстрого и постоянного реагирования.

* + 1. **Система оповещения населения об опасностях**

Система, разработанная МЧС России, которая обеспечивает оповещение населения о предстоящих опасностях через различные каналы связи, такие как SMS, громкоговорители, сирены и телевизионные и радиоэфиры. Система предоставляет информацию о пожарах, наводнениях, землетрясениях и других чрезвычайных ситуациях.

Здесь мы снова сталкиваемся с теми же минусами, что уже видели во всех устаревших государственных системах по всему миру.

* 1. **Итоги анализа рынка**

В результате анализа мирового и российского рынка систем оповещения о чрезвычайных ситуациях, можно сделать следующие выводы:

1) Государственные системы по обеспечению оповещения о чрезвычайных ситуациях имеют ограниченное финансирование, что ведет к ограниченным возможностям обновления и улучшения этих систем. Это приводит к устареванию функционала и неудобству использования для пользователей.

2) Международные системы оповещения, такие как FEMA Alert System в США и J-Alert в Японии, имеют ряд полезных функций, таких как быстрая реакция на чрезвычайные ситуации и использование различных каналов связи. Однако они также имеют свои недостатки, включая ограничение в использовании только на определенной территории и неудобный пользовательский интерфейс.

3) Социальные сети, такие как Nextdoor, имя удобный и понятный интерфейс, могут служить платформой для удобного уведомления соседей о локальных ситуациях, но они не являются организованной системой оповещения.

5) Российские системы экстренного реагирования, включая Систему-112 и систему оповещения населения об опасностях, также имеют ряд недостатков, включая ограниченные возможности реагирования и устаревший функционал.

В целом, существующие системы оповещения о чрезвычайных ситуациях имеют приличное количество недостатков, что заставляет тревожно задумываться, а что будет, если вдруг случиться ЧС. Ведь такие ситуации – явление редкое, именно поэтому наши системы недостаточно проверены в боевых условиях. Такие упущения могут по влечь за собой большие жертвы, поэтому необходимо улучшать работу оповещения до того, как мы убедимся на практике, что наши системы работают недостаточно качественно.

Разработка нового приложения должна учитывать все эти недостатки и стремиться решить их, обеспечивая удобство использования для пользователей и быструю реакцию на чрезвычайные ситуации.

* 1. **Современный ответ на старую проблему.**

Проведя детальный анализ рынка, вывод о необходимости развития данной области напрашивается сам. Успешно разработанная информационная система – это та, которая полностью отвечает всем запросом пользователя. Именно поэтому при проектировании данного приложения я полагался на Domain-Driven Design (DDD) - методологию проектирования и разработки программного обеспечения, которая ставит предметную область цифровизируемой модели в центр внимания [1].

Основные требования исходя из плюсов и минусов аналогов:

1) Наличие доступа к системе через веб-интерфейс или мобильное приложение.

2) Приятный для использования и интуитивно понятный пользовательский интерфейс.

3) Современный методы получения уведомления.

4) Централизованная общая глобальная система, но при этом возможность точно настраиваемых локальных оповещений.

5) Возможность пользователям создавать свои локальные оповещения, для поддержания ещё более точного и оперативного сообщения о чрезвычайной ситуации.

6) Наличие некого сообщества, для более активного социального взаимодействия.

7) Быстрая и бесперебойная доставка уведомлений.

Все вышеперечисленные пункты являются результатом прямого анализа фичей, которые нравились пользователям предыдущих систем, а также анализа недостатков этих систем, и поиска их альтернатив.

При этом, эти 7 пунктов, легли в основу формирования разрабатываемой мной системы. Это приложение будет представлять собой платформу для удобной организации оповещений граждан любой страны, любого города и района. Система буде иметь такую структуру, что будет удобна в независимости от местонахождения пользователя и зоны мониторинга сообществ. И будет позволять реализовывать как глобальные рассылки на уровне страны, так и локальные рассылки для пользователя на уровне его семьи.

Суть проста – система основана на принципах социальной сети:

1. В системе есть две основные сущности с разными возможностями:

а) Первая сущность – пользователь. В его возможности входит добавление в друзья других пользователей, объединение их в группы для удобных рассылок, создание своих локальных шаблонов рассылок. Ещё более важная возможность, подписаться на сообщество, которое мониторит конкретную чрезвычайную ситуацию на конкретной локации (это может быть страна, а может конкретный город).

б) Вторая сущность – сообщество. Это частная или государственная организация, которая занимается мониторингом различных чрезвычайных ситуаций на различных локациях. Они являются основой безопасности большого кол-ва граждан, ведь делают важные рассылки на всю свою аудиторию.

2) Такая организация системы позволяет ей быть удобной как для пользователей, так и для организаций, и более того, позволяет стать всем ныне существующим государственным системам своей частью, и заполучить в свои руки возможность не только аналогового уведомления, но также и такие варианты как: пуш-уведомления, уведомления через чат-боты, уведомления по электронной почте.

3) При этом система даёт пользователям возможность самим поучаствовать в организации своей безопасности, а также безопасности окружающих, ведь бывают ситуации, когда проблема очень локальна, и о ней вам скорее сообщит ваш сосед, чем какая-то организация.

Поэтому в следующих главах я подробно разберу весь используемый инструментарий, необходимый для построения дизайна такой серьёзной системы, а также пошагово пройдусь по этапам её разработки.

**2. Анализ и выбор актуальных технологий.**

В данной главе задокументирован процесс проведения анализа существующих технологий и методик, необходимых для разработки информационной системы основанной на микросервисной архитектуре. Цель этого этапа проектирования состояла в том, чтобы выбрать наиболее подходящие и актуальные технологии для разрабатываемого приложения, которые будут обеспечивать ее эффективную работу и масштабируемость.

В ходе анализа рассматривались различные аспекты, начиная с выбора языка программирования, который является фундаментом разработки, заканчивая инфраструктурными технологиями, для удобного развёртывания и дальнейшей поддержки этой системы. Ещё одним важным аспектом является выбор брокера данных, который обеспечивает коммуникацию между сервисами и поддерживает асинхронную обработку сообщений. Также были изучены вопросы выбора базы данных и работы с кэшем, так как эти компоненты играют важную роль в эффективной работе микросервисных систем.

В результате этих исследований были определенны наиболее подходящие технологии и инструменты для разрабатываемого приложения.

**2.1 Архитектура системы**

Прежде чем перейти к конкретным технологиям и техническим аспектам, важно было разобраться с общими принципами и методиками проектирования микросервисных систем.

Micro Service Architecture (MSA) – Микросервисная архитектура - [принципиальная](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF) организация распределенной [системы](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) на основе микросервисов и их взаимодействия друг с другом и со средой по сети, а также принципов, направляющих [проектирование](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) архитектуры, её создание и эволюцию [1].

Такая архитектура предлагает подход, при котором приложение разбивается на небольшие независимые сервисы, специализирующиеся на определенных функциональных областях. Это позволяет достичь высокой гибкости, масштабируемости и легкости разработки и сопровождения системы.

Понять суть микросервиса проще всего на сравнении, или даже противопоставлении его крупному приложению – монолиту (Рисунок 2.1). Каждый отдельный сервис в такой архитектуре обладает следующими свойствами:

1. Он небольшой – микросервис отличается от монолитных приложений тем, что он имеет небольшой размер и ограниченные обязанности. Он фокусируется на решении конкретных задач в пределах своей функциональности.
2. Он независимый – микросервис является автономным и независимым компонентом, который может функционировать независимо от других сервисов. Он имеет свои собственные ресурсы, базу данных и логику обработки данных.

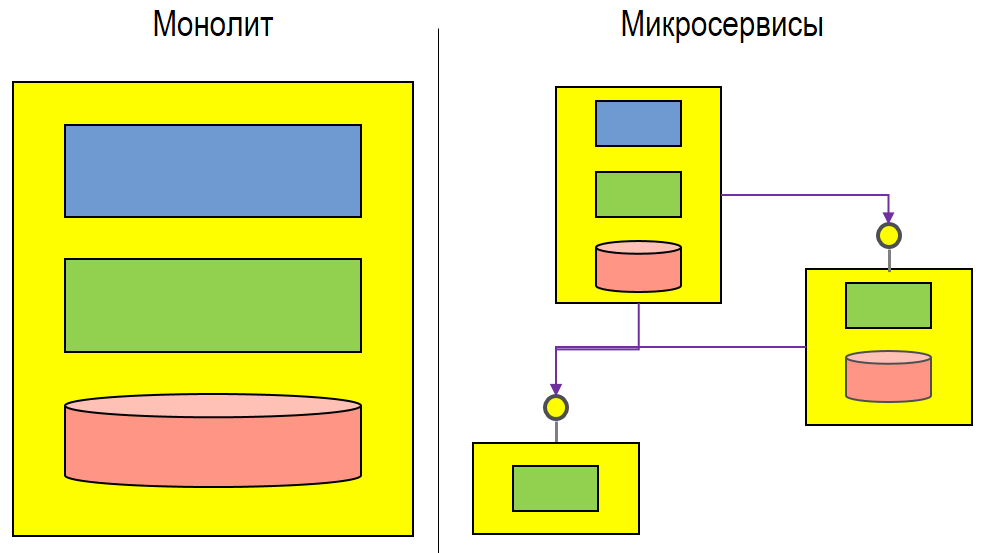


Рисунок 2.1 – Сравнение монолитной и микросервисной архитектур [2].

1. Он строится вокруг бизнес-потребности (или просто предметной потребности) и использует ограниченный контекст (Bounded Context). Bounded Context – это понятие в микросервисной архитектуре, которое описывает ограниченную область или границу в предметной области системы. В простых словах, это способ организации сложных систем, разбивая их на отдельные, логически связанные части, чтобы каждая из них отвечала за определенный участок функциональности. Каждый Bounded Context имеет свою локальную модель данных и границы, внутри которых существует ясное понимание того, как работает и взаимодействует каждый компонент. Это позволяет разрабатывать и масштабировать каждую часть независимо, обеспечивая гибкость и улучшенную поддержку изменений в развивающихся системах.
2. Он взаимодействует с другими микросервисами по сети на основе паттерна Smart endpoints and dumb pipes («умные эндпоинты и глупые каналы связи»): В микросервисной архитектуре взаимодействие между сервисами осуществляется по сети. При этом, подход «умные эндпоинты и глупые каналы связи» означает, что основная логика и функциональность находится внутри каждого микросервиса (умные эндпоинты), а каналы связи (глупые каналы) предоставляют простой способ передачи данных между сервисами.
3. Его распределенная суть обязывает использовать подход Design for failure («отказоустойчивое проектирование»): Микросервисы расположены на разных серверах и могут взаимодействовать друг с другом через сеть. Это означает, что система должна быть готова к возможным отказам сервисов, сетевым проблемам и другим ситуациям, которые могут возникнуть в распределенной среде. Подход «отказоустойчивое проектирование» подразумевает, что система разрабатывается таким образом, чтобы она могла продолжать работать даже при возникновении отказов в одном или нескольких сервисах.
4. Централизация ограничена сверху на минимуме – в микросервисной архитектуре стремятся минимизировать централизацию, особенно на уровне архитектуры и управления. Каждый сервис выполняет определенные функции и имеет свою собственную логику и хранение данных. Это позволяет улучшить гибкость и масштабируемость системы, так как изменения в одном сервисе не затрагивают остальные.
5. Процессы его разработки и поддержки требуют автоматизации –Разработка и поддержка микросервисных приложений включают в себя множество сервисов, которые работают вместе. Для облегчения и упрощения этого процесса требуется автоматизация. Это может быть автономная сборка, развертывание, мониторинг, тестирование и другие процессы, которые помогают обеспечить эффективное развитие и поддержку микросервисов.
6. Его развитие итерационное – В микросервисной архитектуре разработка и внедрение новых функций и изменений происходят итеративно. Это означает, что процесс разработки разбивается на небольшие итерации, в рамках которых выполняется планирование, разработка, тестирование и развертывание нового функционала. Такой подход позволяет быстрее реагировать на изменения и обратную связь от пользователей, обеспечивая постепенное развитие приложения.

Изучив только эти особенности микросервиса, можно провести ассоциативную параллель с классом в объектно-ориентированном программировании. Здесь мы видим принципы и соглашения схожие с основными принципами ООП, а также, вытекающими из них, принципами SOLID, только на один уровень абстракции выше. Особенно заметно влияние двух первых принципов SRP и SOP.

Single Responsibility Principle (SRP) или же Принцип единой ответственности – этот принцип гласит, что у каждого объекта есть своя ответственность и причина существования и эта ответственность у него только одна. Другими словами, через объект проходит только одна ось изменений.

Open-Closed Principle (OСP) или же Принцип открытости-закрытости – гласит, что любой модулю класс или функция должны быть открыты для расширения и закрыты для изменения.

а) Открыты для расширения. Это означает, что поведение модуля может быть расширено. То есть мы можем добавить модулю новое поведение в соответствии с изменившимися требованиями к приложению или для удовлетворения нужд новых приложений.

б) Закрыты для изменений. Исходный код такого модуля неприкасаем. Никто не вправе вносить в него изменения.

Принцип единой ответственности (SRP) аналогичен идее, что каждый микросервис должен иметь только одну четко определенную ответственность. Принцип открытости-закрытости (OCP) также перекликается с идеей микросервисной архитектуры. В контексте микросервисов, каждый сервис должен быть открыт для расширения, то есть, его функционал может быть легко расширен или изменен без влияния на другие сервисы. В то же время, каждый сервис должен быть закрыт для изменения, чтобы предотвратить нежелательное влияние на другие сервисы или систему в целом.

Благодаря соблюдению этих принципов в микросервисной архитектуре достигается высокая устойчивость, строгость и безопасность системы. Каждый сервис имеет четко определенную ответственность, что упрощает разработку и поддержку. Изменения в одном сервисе могут быть внесены без привлечения других сервисов, что улучшает масштабируемость и гибкость системы. Более того, такая структура позволяет лучше изолировать возможные ошибки и проблемы, благодаря чему отказ одного сервиса не приводит к сбою всей системы. Всё это способствует стабильности работы и безопасности микросервисной архитектуры.

Всё это и обусловило выбор данной архитектуры, ведь главные показатели, которые нужны системе, обеспечивающей уведомление пользователей о чрезвычайных ситуациях это: отказоустойчивость, масштабируемость, строгость, безопасность, скорость.

Так же при выборе не были закрыты глаза на минусы данного подхода к реализации информационных систем. Сложность взаимодействия, сложность тестирования, и цена разработки и поддержки – это то, чем необходимо пожертвовать, ради достижения безопасности.

**2.2 Предметно-ориентированный подход**

Domain-Driven Design (DDD) или же Предметно-ориентированное проектирование – это методология проектирования и разработки программного обеспечения, которая ставит предметную область бизнеса в центр внимания. На самом деле эта концепция рабочая для проектирования информационной системы любой области, где это необходимо, а не только для бизнеса. Основная идея DDD заключается в том, что разработчики и доменные эксперты должны активно взаимодействовать, чтобы понять предметные-требования и перенести их в модель предметной области, которая будет использоваться для реализации системы [3].

Принципы DDD помогают структурировать и организовать систему, чтобы она лучше соответствовала предметной области. Основные принципы DDD включают в себя:

1. Основные понятия предметной области (Building Blocks): Это концепции, которые используются для представления и описания предметной области. Они включают в себя сущности (Entities), значения (Value Objects), агрегаты (Aggregates), репозитории (Repositories) и сервисы (Services), которые помогают моделировать и организовывать систему на основе предметной-логики [3].

2. Соответствие между моделью и предметной областью (Ubiquitous Language): Разработчики и доменные эксперты должны использовать общий язык, который будет удобен и понятен для всех участников проекта. Общий язык помогает уточнить требования и согласовать понимание процессов той системы, которую необходимо цифровизировать [3].

3. Обособленные (изолированные) контексты (Bounded Contexts): Этот принцип предлагает разбить систему на изолированные контексты, где каждый контекст имеет собственную модель, правила и ограничения. Это помогает управлять сложностью системы и делить ее на небольшие компоненты, которые легче понять и поддерживать [3].

4. Фокус на аспектах значимых для предметной области (Core Domain): Принцип DDD настоятельно рекомендует обращать особое внимание на процессы и функционал, на которых сосредоточена основная ценность для предметной области. Функциональность, которая не является основной для системы, может быть вынесена в отдельные компоненты или модули [3].

5. Разработка на основе эволюции (Emergent Design): DDD призывает к гибкому и постепенному проектированию системы. У разработчиков должна быть возможность итеративно вносить изменения в модель предметной области и архитектуру системы на основе реальных требований и улучшений [3].

В общем случае получаем, что применение принципов DDD позволяет создавать более поддерживаемые, модульные и гибкие системы, которые лучше соответствуют реальным потребностям предметной области и обеспечивают успешное взаимодействие между разработчиками и бизнесом.

В частном случае, видим решение, идеально подходящее для декомпозиции предметной области, и к подготовке её к цифровизации на основе микросервисной архитектуры.

Предметно-ориентированный подход (Domain-Driven Design, DDD) и микросервисная архитектура (Microservices Architecture, MSA) дополняют друг друга и могут быть эффективно применены вместе. Оба подхода фокусируются на разделении системы на изолированные компоненты с уникальной ответственностью, что позволяет достичь лучшей модульности, масштабируемости и гибкости системы.

Одним из ключевых принципов как DDD так и MSA является ограниченный контекст (Bounded Context):

1. Каждый ограниченный контекст определяет свою собственную модель предметной области и язык, позволяя разработчикам и бизнес-экспертам иметь ясное понимание того, как работает конкретная часть системы.
2. В контексте микросервисной архитектуры, каждый микросервис может быть реализацией одного или нескольких ограниченных контекстов. Это позволяет разработчикам разрабатывать и поддерживать каждый микросервис как отдельное приложение, с четко определенной областью ответственности и языком.

DDD и MSA также поддерживают децентрализованное управление данными. В DDD, каждый ограниченный контекст может иметь собственное хранилище данных, а MSA стремится к принципу независимых баз данных для каждого сервиса. Это позволяет разрабатывать и масштабировать каждый микросервис независимо от других, улучшая производительность системы и снижая возможность конфликтов данных.

В целом, предметно-ориентированный подход в проектировании, основанный на принципах DDD, идеально подходит для микросервисной архитектуры, так как оба подхода стремятся к изоляции функциональности и управлению сложностью через Ограниченные Контексты и независимую разработку и развертывание компонентов системы. Это позволяет эффективно управлять сложностью и изменениями, создавая более гибкие и модульные системы.

**2.2 Язык программирования**

В данном разделе представлен анализ факторов, определяющих выбор языка программирования для разработки приложения, предназначенного для предоставления пользовательских уведомлений о чрезвычайных ситуациях на основе микросервисной архитектуры. Критерии выбора включают эффективность, гибкость и безопасность. В результате проведённых исследований и сравнительного анализа, был выбран язык программирования Java и сопутствующее семейство фреймворков Spring.

Java представляет собой одну из самых распространенных технологий программирования в области backend разработки и реализации сложных и масштабных приложений. В настоящее время Java находит широкое применение благодаря ряду преимуществ, которые следует рассмотреть:

1. Кроссплатформенность – Java разработан с учетом платформенной независимости, что означает, что разработанные на нем приложения могут быть запущены на различных операционных системах без необходимости изменения кода. Такой подход обеспечивает высокую гибкость и возможность развертывания приложений на различных платформах [4].
2. Обратная совместимость – принцип, который Java не нарушала никогда. Многие считают Java слишком консервативным и медленно развивающимся языком, но те, кто разрабатывает на Java всегда уверенны, что перенос своих проектов на новые версии Java будет осуществлён легче и быстрее, чем на каком-либо языке. Это качество ставит Java на уровень выше по сравнению с С#, казалось бы, очень похожим по всем принципам на неё.
3. Безопасность – Одним из привлекательных аспектов Java является наличие встроенных механизмов безопасности. Запуск программ на Java осуществляется в виртуальной машине Java (JVM), которая обеспечивает изоляцию исполняемого кода от операционной системы и снижает уязвимость к вредоносным атакам [4].
4. Отказоустойчивость – Java обладает механизмами управления памятью и обработки исключений, сделавшими его надежным и стойким к сбоям. Высокоуровневые средства управления памятью, включая сборку мусора, способствуют предотвращению утечек памяти, а система обработки исключений облегчает идентификацию и обработку ошибок. Всё это облегчает и ускоряет разработку в отличие от тех же C и С++ [4].
5. Богатая экосистема – Java располагает обширной и динамичной экосистемой, включающей в себя множество библиотек, фреймворков и инструментов разработки. Этот фактор позволяет разработчикам использовать готовые решения для типовых задач и значительно сократить время и ресурсы, затрачиваемые на разработку [4].
6. Масштабируемость – Java обладает возможностями разработки масштабируемых приложений. Он позволяет эффективно обрабатывать большое количество данных и обеспечивать высокую производительность благодаря использованию многопоточности, распределенных вычислений и других техник [4].
7. Строгость – Java строго типизированный язык, а также строго структурированный, соответствует строго объектно-ориентированному подходу. Строгость является одним из важнейших качеств языка Java, что позволяет выявлять ошибки на ранних стадиях, а также способствует читаемости кода. Это положительно выделяет её на фоне таких языков как Python и PHP. Хотя второй в своих новых версиях уже начал стримиться к строгой типизации.

Хорошим аналогом мог быть быстрый современный функциональный язык программирования Go lang, но по причинам его «незрелости», за которой не прерывно следует меньшая степень интеграции с сторонними технологиями, менее богатая экосистема и менее развитое сообщество, а также по причине того, что код в Go соответствует функциональному стилю, ломающему всю строгость и концепцию, выстроенную ранее, выбор пал на язык Java.

Spring является одним из наиболее популярных фреймворков разработки приложений на Java. Он предоставляет обширные возможности и инструменты для создания и организации микросервисных архитектур. Преимущества Spring по сравнению с другими фреймворками можно охарактеризовать следующим образом:

1. Легкая интеграция – Spring обладает возможностью интеграции с различными технологиями и фреймворками. Он поддерживает использование различных модулей и библиотек, таких как Spring Data, Spring Security и Spring MVC для реализации разнообразных функциональных возможностей в приложениях [5].
2. Инверсия управления (IoC) – Применение принципа инверсии управления в Spring позволяет снизить связанность компонентов системы. Благодаря этому, разработчики могут создавать, тестировать и заменять компоненты приложения без изменения других частей системы [5].
3. Аспектно-ориентированное программирование (АОП) – Spring поддерживает АОП, что позволяет вынести "сквозную" функциональность, например, журналирование, управление транзакциями и безопасность, в отдельные модули. Это облегчает поддержку и масштабирование приложений [5].
4. Тестирование – Spring обладает встроенными средствами для тестирования приложений. Он предоставляет мок-объекты, инструменты для модульного и интеграционного тестирования, что способствует созданию надежного и стабильного кода [5].
5. Поддержка сообщества – у Spring активное сообщество разработчиков, которое предоставляет обширную документацию, готовые решения и поддержку. Это позволяет решать возникающие проблемы более эффективно, а разработчикам обратиться за помощью в случае необходимости [5].

В результате анализа, был сделан обоснованный выбор Java и Spring для разработки микросервисных архитектур. Java обеспечивает безопасность, отказоустойчивость и масштабируемость, а Spring предоставляет удобные инструменты и функциональность для разработки и управления микросервисами.

**2.3 Базы данных**

Базы данных являются одними из основных технологий в серверной разработке. Если в общем смысле можно описать программирование как алгоритмы над данными, и при этом мы будем представлять данные, как некоторые значения, лежащие в оперативной памяти, а алгоритмы, как вычислительную логику, производимую над этими данными, то теперь на уровне энтерпрайс разработки за алгоритмы будем принимать предметную или бизнес-логику (Domain), а за часть, отвечающую за данные, мы будем принимать именно слой базы данных.

Поэтому очень важно правильно подобрать технологии, которые будут хорошо выполнять необходимые нам задачи, а также будут совместимы с нашими другими технологиями. Перед выбором существующих баз данных, необходимо понять, каким образом вообще Java будет осуществлять работу с ними. В контексте взаимодействия Java-приложений с базами данных, существует несколько ключевых технологий и понятий, которые стоит рассмотреть.

Начнем с JDBC (Java Database Connectivity) – интерфейса прикладного программирования для языка Java, который определяет, как клиент может получить доступ к базе данных. JDBC является технологией доступа к данным на основе Java и предоставляет методы для запроса и обновления данных в реляционных базах данных. Раньше, для взаимодействия с базами данных Java-разработчики часто использовали JDBC напрямую. Он требует непосредственного написания SQL-запросов и работы с ними через Java-код.

Однако с развитием технологий были созданы ORM-фреймворки (Object-Relational Mapping), которые предоставляют более удобные и абстрактные способы взаимодействия с базами данных. Одним из самых популярных ORM-решений для Java является Hibernate. Hibernate предоставляет возможности автоматического отображения Java-классов и их свойств на таблицы и столбцы базы данных. Он генерирует SQL-запросы и управляет процессом сохранения, обновления и извлечения данных, что значительно упрощает разработку и уменьшает необходимость вручную писать SQL-код. Hibernate также является реализацией спецификации JPA (Java Persistence API), которая описывает систему управления сохранением Java-объектов в реляционных базах данных.

С использованием Hibernate и JPA, разработчики могут работать с объектами в своем коде, а фреймворк автоматически преобразует эти объекты в SQL-запросы и обеспечивает их выполнение в базе данных. Такой подход существенно упрощает разработку и делает код более портируемым между различными базами данных.

В современной разработке Java-приложений также широко используется Spring Framework, который предоставляет множество полезных инструментов и модулей для разработки приложений. Один из таких модулей - Spring Data, который упрощает работу с базами данных в контексте Spring [5].

Spring Data позволяет организовать доступ к данным через репозитории, где достаточно объявить интерфейс с некоторыми методами, а фреймворк автоматически генерирует реализацию этих методов, выполняя необходимые SQL-запросы или вызывая соответствующие методы ORM-библиотеки. Это значительно сокращает объем кода, облегчает разработку и повышает читаемость [5].

Именно поэтому по мимо того, что базы данных, которые будут описаны далее, соответствуют предметным требованиям проекта, они так же отлично сочетаются с работой совместно с Hibernate и Spring Data.

**2.3.1 PosgreSQL**

PostgreSQL является одним из ведущих реляционных баз данных и обладает рядом преимуществ и уникальных особенностей, которые делают его предпочтительным выбором для множества разработчиков и организаций.

Во-первых, PostgreSQL обладает высокой степенью расширяемости, что позволяет разработчикам настраивать базу данных в соответствии с их потребностями. Возможность использования хранимых процедур, триггеров, пользовательских типов данных и операторов расширяет функциональность PostgreSQL и позволяет создавать собственные расширения.

Во-вторых, PostgreSQL строго соответствует стандартам SQL, включая последние версии стандарта, такие как SQL:2008. Это обеспечивает переносимость приложений и упрощает разработку и поддержку кода, поскольку разработчики могут полагаться на единый набор стандартных SQL-запросов в различных базах данных.

В-третьих, PostgreSQL предоставляет богатый набор расширений для работы с географическими данными, включая популярную платформу PostGIS. Это позволяет хранить, обрабатывать и выполнять сложные геопространственные запросы, что особенно важно для приложений, связанных с картографией, геолокацией и аналитикой данных [6].

Дополнительно, PostgreSQL обладает продвинутой системой репликации данных, предлагая методы потоковой репликации и логической репликации. Это позволяет создавать отказоустойчивые системы с высокой доступностью данных.

Основная сила PostgreSQL также заключается в богатом наборе типов данных. Это включает различные специализированные типы для работы с JSON, XML, массивами, полнотекстовым поиском и многими другими, что упрощает моделирование данных и обработку различных типов информации [6].

В конечном итоге, PostgreSQL предлагает различные алгоритмы индексации для оптимизации производительности запросов и повышения быстродействия базы данных при поиске и фильтрации данных.

Конкурентом данной базы является Oracle MySQL, который имеет не меньшую популярность и является очень серьёзным инструментом для работы с данными.

Первое существенное различие между ними заключается в их функциональности. PostgreSQL предлагает более широкий набор возможностей и функций, включая более полное соответствие стандартам SQL, поддержку географических данных и расширяемость. С другой стороны, MySQL позиционируется как простая в использовании и быстрая база данных, особенно подходящая для простых веб-приложений, и имеет лучшую производительность на некоторых типах нагрузки, таких как простые CRUD-операции.

Также следует учесть экосистему и инструментальную поддержку. Обе базы данных имеют разнообразные экосистемы и инструменты, однако PostgreSQL, обладает более разнообразным сообществом пользователей и разработчиков, а также большим количеством готовых решений для различных задач.

Поскольку многие приложения на языке Java требуют эффективной и надежной работы с базами данных, PostgreSQL представляет собой идеальный вариант для использования в связке с Java и Hibernate.

PostgreSQL предоставляет хорошо развитый и популярный драйвер JDBC, обеспечивающий удобное взаимодействие с базой данных на языке Java. Это создает удобную и мощную среду для разработки Java-ориентированных приложений.

Hibernate, также тесно интегрирован с PostgreSQL. Он обеспечивает удобное отображение объектов Java на таблицы в базе данных PostgreSQL, а также автоматическую генерацию SQL-запросов. Такая комбинация облегчает разработку приложений и упрощает работу с данными.

**2.3.2 MongoDB**

MongoDB является одним из лидеров среди документных баз данных по ряду причин. Во-первых, ее гибкая схема данных позволяет каждому документу в коллекции иметь собственную структуру, что облегчает изменение схемы данных без обновления всей базы данных. Во-вторых, MongoDB обладает уникальными возможностями горизонтального масштабирования, позволяющими распределить данные по нескольким серверам и обеспечить высокую производительность даже при работе с большими объемами данных. Это особенно важно для растущих проектов. MongoDB также поддерживает репликацию данных для обеспечения отказоустойчивости. Кроме того, MongoDB обеспечивает высокую производительность благодаря хранению данных в близкой к памяти форме и использованию индексов. Она также предоставляет возможность выполнения распределенных запросов, что позволяет эффективно распределять нагрузку на серверы. Наконец, MongoDB предлагает богатый набор функций, включая индексы, запросы на основе текста, агрегационные запросы, отслеживание изменений и географические запросы, делая ее удобной для различных типов приложений [7].

При сравнении MongoDB и Cassandra, обе базы данных являются популярными и масштабируемыми решениями, но они имеют несколько различий. Во-первых, модель данных в MongoDB основана на документах, где каждый документ представляет собой JSON-подобный объект (Рисунок 2.2).

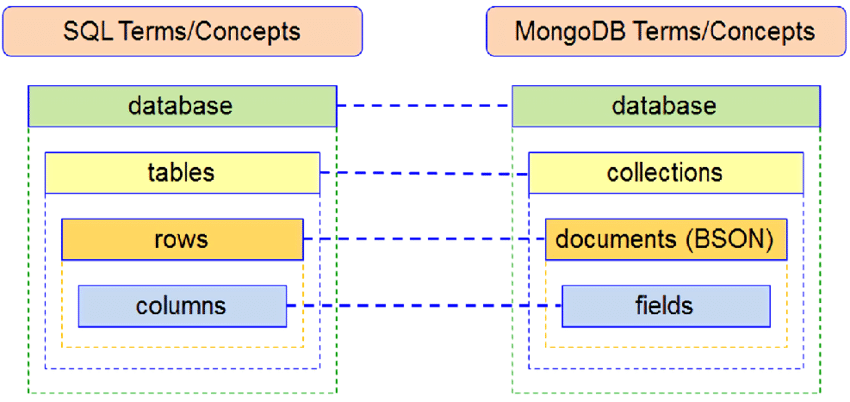


Рисунок 2.2 – Сравнение MongoDB с SQL базами. [7].

В то время как Cassandra использует модель данных на основе колонок, где данные хранятся в виде широких рядов с неопределенным числом столбцов. Во-вторых, MongoDB обеспечивает согласованность данных по умолчанию (strong consistency), в то время как Cassandra обеспечивает связь между доступностью и разделением (tuneable consistency). Это означает, что Cassandra может обеспечить высокую доступность данных в условиях распределенных систем, но может потерять согласованность в некоторых случаях. Наконец, язык запросов MongoDB более гибок и похож на язык запросов для реляционных баз данных, в то время как CQL (Cassandra Query Language) было специально разработано для работы с моделью данных Cassandra [7].

Для использования MongoDB с Java и Hibernate существует несколько причин, почему эта комбинация может быть идеальной. Во-первых, MongoDB предоставляет официальный драйвер для Java, который обеспечивает нативную интеграцию между Java-приложениями и базой данных MongoDB. Этот драйвер предлагает различные функции, такие как поддержка аннотаций, маппинг объектов на документы MongoDB и простота в использовании. Во-вторых, Hibernate также предоставляет модуль под названием Hibernate OGM (Object/Grid Mapper), который позволяет работать с документными базами данных, включая MongoDB. Это позволяет разработчикам использовать привычные средства Hibernate для работы с MongoDB.

**2.3.3 Распределение задач**

Так как разрабатываемая архитектура основана на микросервисах, базы данных которых обособленны друг от друга, мы без лишних сложностей можем использовать одновременно различные технологии для работы с базами данных.

Для чего понадобилось использование двух различных технологий, таких как PosgreSQL и MongoDB.

Реляционная база, такая как PosgreSQL, необходимо в разрабатываемом проекте по причине своей надёжности. В реляционных базах данных существуют принципы, обеспечивающие безопасность на уровне транзакций (ACID), эти принципы могут ассоциативно напоминать принципы SOLID для объектно-ориентированной разработки.

ACID – это аббревиатура, которая описывает набор принципов и свойств, обычно применяемых в контексте транзакций в базах данных [7]. Каждая буква в слове ACID представляет отдельный принцип:

1. Атомарность (Atomicity) – Принцип атомарности гарантирует, что транзакция будет выполнена либо полностью, либо не выполнена совсем [7]. Это означает, что если в рамках транзакции возникает ошибка или проблема, то все изменения, сделанные до этой точки в транзакции, откатываются, и не происходит никаких изменений в базе данных. Таким образом, принцип атомарности обеспечивает целостность данных.
2. Согласованность (Consistency) – Принцип согласованности гарантирует, что транзакция приводит к изменению базы данных из одного согласованного состояния в другое согласованное состояние. Это означает, что транзакция должна соблюдать все ограничения целостности данных, установленные в базе данных. Если транзакция не соблюдает эти ограничения, то она откатывается, и база данных остается в прежнем состоянии.
3. Изолированность (Isolation) – Принцип изолированности гарантирует, что каждая транзакция выполняется независимо от других транзакций. Изменения, сделанные в рамках одной транзакции, не будут видимы другим транзакциям до тех пор, пока первая транзакция не будет завершена. Это предотвращает нежелательное взаимодействие между транзакциями и обеспечивает консистентность данных.
4. Долговечность (Durability) – Принцип долговечности гарантирует, что результаты выполненных транзакций сохраняются даже в случае сбоев или перезапуска системы. Это достигается с помощью сохранения изменений в постоянное хранилище, такое как жесткий диск или несколько реплик базы данных. Таким образом, даже после сбоя данные остаются надежными и доступными.

ACID-принципы важны для обеспечения надежности и целостности данных в базах данных, особенно в контексте транзакций [7]. Их соблюдение гарантирует, что операции записи и чтения данных выполняются надежно и предсказуемо, даже в случае сбоев или параллельного выполнения множества транзакций.

Поэтому эта база будет использована как хранилище пользовательской информации, их статусов, паролей, логинов, и другой важной информации, такой как статус отправки уведомления.

MongoDB основанная на документах представляющих собой JSON-подобный объект удобна в тех сервисах, которые будут содержать шаблоны уведомлений создаваемых пользователями и сообществами.

**2.4 Работа с кэшем**

В современных микросервисных архитектурах широко применяется кэширование данных для повышения производительности, снижения задержек и облегчения нагрузки на базы данных и другие службы. Выбор подходящей технологии кеширования играет важную роль в обеспечении эффективной работы микросервисов. В данной главе мы рассмотрим несколько популярных технологий кэширования, их плюсы и минусы.

In-Memory Data Grids (IMDG) - IMDG-системы предлагают хранение данных в оперативной памяти распределенных систем, что обеспечивает высокую производительность и быстрый доступ к данным. Примерами IMDG-систем являются Apache Ignite, Hazelcast, Oracle Coherence и GridGain. Они предоставляют удобные API для работы с кэшем и масштабируются горизонтально для управления большими объемами данных. Однако, в некоторых случаях, использование IMDG может потребовать дополнительной конфигурации и настройки. Обобщая, IMDG-системы обладают высокой производительностью и легко масштабируются, но требуют дополнительной конфигурации [8].

Distributed Caches - Распределенные кэши - предназначены для хранения данных на нескольких узлах в распределенной сети. Они имеют возможность обрабатывать большие объемы данных и обеспечивают отказоустойчивость. Некоторыми из популярных распределенных кэш-систем являются Redis, Memcached и Apache Cassandra. Redis отличается высокой скоростью и поддержкой широкого спектра функций (например, структуры данных, пайплайнинг, публикация/подписка), что делает его привлекательным вариантом для кэширования данных в микросервисных архитектурах. Резюмируя, распределенные кэши обладают высокой производительностью, поддерживают отказоустойчивость. Redis является мощным вариантом, предлагая широкий набор функций [8].

Библиотеки и фреймворки кэширования - существуют различные библиотеки и фреймворки, которые предоставляют специализированные средства для работы с кэшем. Примерами таких фреймворков в экосистеме Java являются Spring Cache, Ehcache и Caffeine. Они предлагают удобные абстракции для работы с кэшем, интеграцию с другими компонентами приложений и легко настраиваются. Однако, они могут быть менее гибкими и масштабируемыми по сравнению с IMDG или распределенными кэш-системами [8].

БД с поддержкой кэширования - некоторые базы данных (например, Oracle Database или Microsoft SQL Server) имеют встроенные функции кэширования для оптимизации доступа к данным. Они позволяют предварительно загрузить данные в кэш или кэшировать часто запрашиваемые данные, что повышает производительность системы. Однако, эти функции могут быть лимитированы, и базой данных может быть сложно масштабироваться горизонтально для обработки растущей нагрузки [8].

HTTP-кэширование - для веб-приложений, использующих протокол HTTP, можно использовать механизмы кэширования, предоставляемые браузерами и прокси-серверами. Это позволяет сохранять результирующие данные запросов на клиентской стороне или промежуточных серверах для повторного использования. HTTP-кэширование прост в использовании и может снизить нагрузку на серверы, но оно имеет ограниченные возможности настройки и контроля над данными в кэше [8].

На основании представленной выше информации, напрашивается очевидный вывод: Redis является привлекательной технологией кэширования данных в микросервисной архитектуре, особенно для Java и Spring разработчиков. Redis обладает высокой производительностью, поддерживает широкий спектр функций и имеет хорошую совместимость с Java и Spring.

[Redis](http://redis.io/) (Remote Dictionary Service) — это опенсорсный сервер баз данных типа ключ-значение. Redis — это база данных, размещаемая в памяти, которая используется, в основном, в роли кеша, находящегося перед другой, «настоящей» базой данных, вроде MySQL или PostgreSQL (Рисунок 2.3) [9].

Redis представляет собой распределенную кэш-систему, которая может хранить данные в оперативной памяти, обеспечивая высокую скорость доступа. Он поддерживает различные структуры данных, такие как строки, списки, хэши, множества и сортированные множества, что делает его гибким инструментом для кэширования различных типов данных. Redis также предлагает дополнительные функции, такие как публикация/подписка, транзакции и Lua-скрипты, что дает возможность реализации сложной логики кэширования.

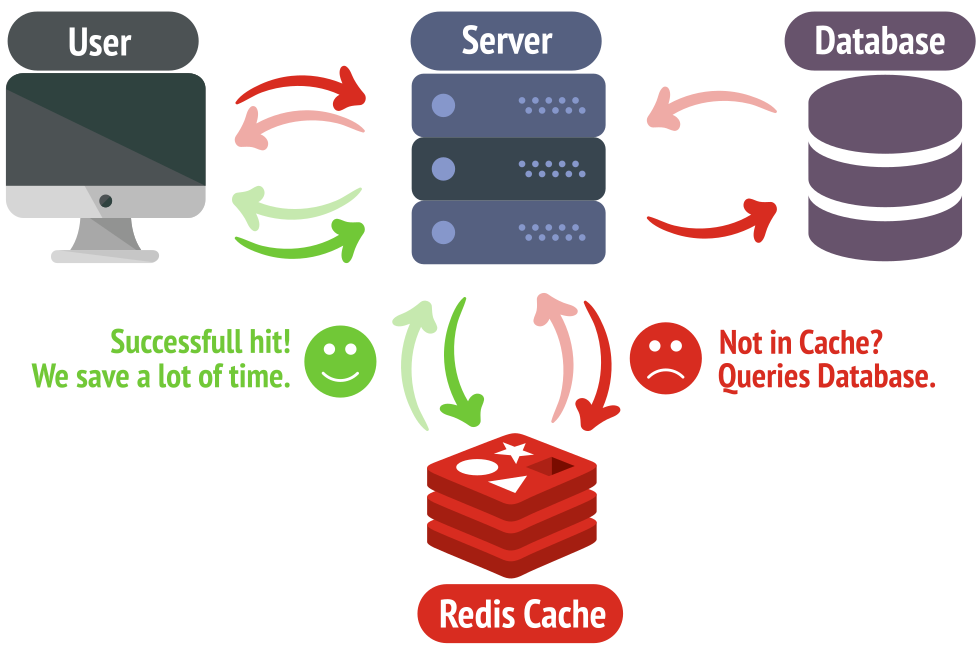


Рисунок 2.3 – Принцип работы с Redis кешированием. [9].

Redis имеет открытый и активно развивающийся экосистему, которая предлагает различные клиентские библиотеки и инструменты для интеграции с Java и Spring. Например, существует официальная библиотека Jedis для взаимодействия с Redis на языке Java, а также Spring Data Redis, который предоставляет удобные абстракции и интеграцию с Spring-приложениями.

**2.5 Брокеры данных**

**2.6 Связь между сервисами**

**2.7 Экосистема**

**3. Проектирование системы**

**3.1 Анализ и декомпозиция предметной области**

**3.2 Общая структура**

**3.3 Сервисы**

**3.4 Проектирование баз данных**

**3.5 Проектирование пользовательского интерфейса**

**ЗАКЛЮЧЕНИЕ**

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Создание микросервисов. / Сэм Ньюмен; 2-е изд. — СПб.: Питер, 2023. — 624 с.: ил. — (Серия «Бестселлеры O’Reilly»); ISBN 978-5-4461-1145-9 (рус.), ISBN 978-1492034025 (англ.).

2. Просто о микросервисах // Хабр: [сайт]. – 2018. – URL: https://habr.com/ru/companies/raiffeisenbank/articles/346380/ (дата обращения: 07.11.2023).

3. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем / Эрик Эванс; Пер. с англ. - М.: 000 "И.Д. Вильямс", 2011. - 448 с.: ил. - Парал. тит. англ. ISBN 978-5-8459-1597-9 (рус.), ISBN 978-0-321-12521-7 (англ.).

4. Java. Полное руководство, 12-е изд. / Герберт Шилдт.: Пер. с англ. - СПб. "Диалектика"; 2023. - 1344 с.: ил. - Парал. тит. англ. ISBN 978-5-907458-86-4 (рус.), ISBN 978-1-26-046341-5 (англ.).

5. Spring в действии. 6-е изд. / Крейг Уоллс, пер. с англ.А. Н. Киселева. – М.: ДМКПресс, 2022. – 544 с.: ил. ISBN 978-1-6172-9757-1 (англ.), ISBN 978-5-93700-112-2 (рус.).

6. High-Performance Java Persistence: Get the most out of your persistence layer / Vlad Mihalcea.: Leanpub, 2015 - 2020. – 444 c.

7. Эрик Редмонд, Джим. Р. Уилсон / Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL. Под редакцией Жаклин Картер / Пер. с англ. Слинкин А. А. – М.: ДМК Пресс, 2013. – 384с.: ил. ISBN 978-5-94074-866-3 (рус.), ISBN 978-1-93435-692-0 (англ.).

8. Шаблоны интеграции корпоративных приложений / Грегор Хоп, Бобби Вульф, Кайла Брауна, Конрада Ф. Д’Круза [и др.].: Пер. с англ. - М.: ООО ‘‘И.Д. Вильямс’’, 2007. - 672 с.: ил - Парал. тит. англ. ISBN 978-5-8459-1146-9 (рус.), ISBN 0-321-20068-3 (англ.).

9. Redis in Action / Josiah L. Carlson, 2013 by Manning Publications Co. All rights reserved. - 294c. ISBN 9781935182054 (анг.).

10. Apache Kafka. Потоковая обработка и анализ данных. 2-е изд. / Гвен Шапира, Тодд Палино, Раджини Сиварам, Крит Петти — СПб.: Питер, 2023. — 512 с.: ил. — (Серия «Бестселлеры O’Reilly»). ISBN 978-5-4461-2288-2 (рус.), ISBN 978-1492043065 (англ.).

11. RabbitMQ in Action / Alvaro Videla, Jason J.W. Williams; - 2012 by Manning Publications Co. All rights reserved. - 287 с.: ISBN 9781935182979.

17. Микросервисы Spring в действии / Джон Карнелл, Иллари Уайлупо Санчес, пер. с англ. А. Н. Киселева. – М.: ДМК Пресс, 2022. – 490 с.: ил. ISBN 978-5-97011-971-2 (рус.), ISBN 978-1-617-29695-6 (анг.).

**ПРИЛОЖЕНИЕ А**



Рисунок А.1 – Макет лендинга разрабатываемого приложения

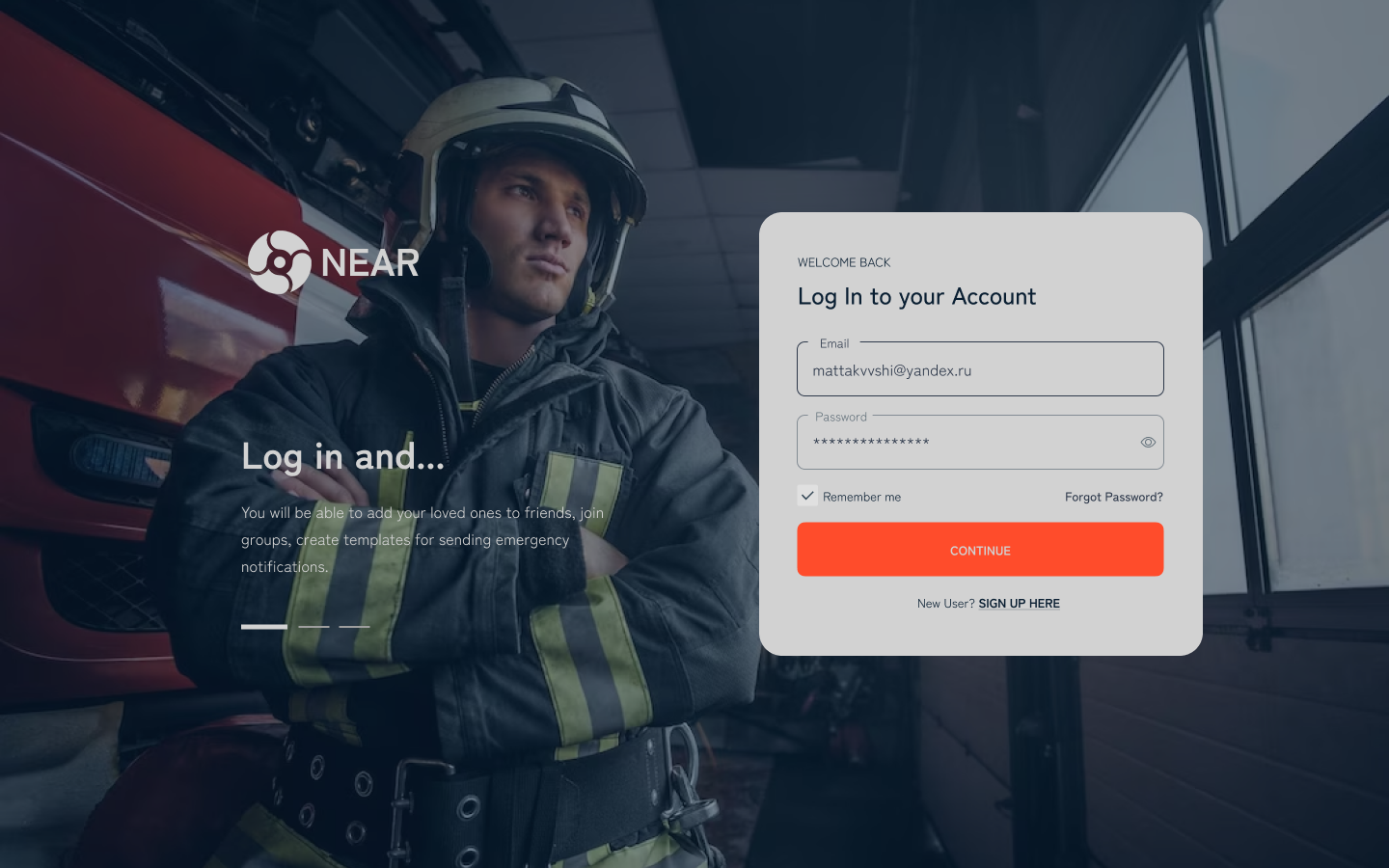


Рисунок А.2 – Макет страницы авторизации пользователя разрабатываемого приложения

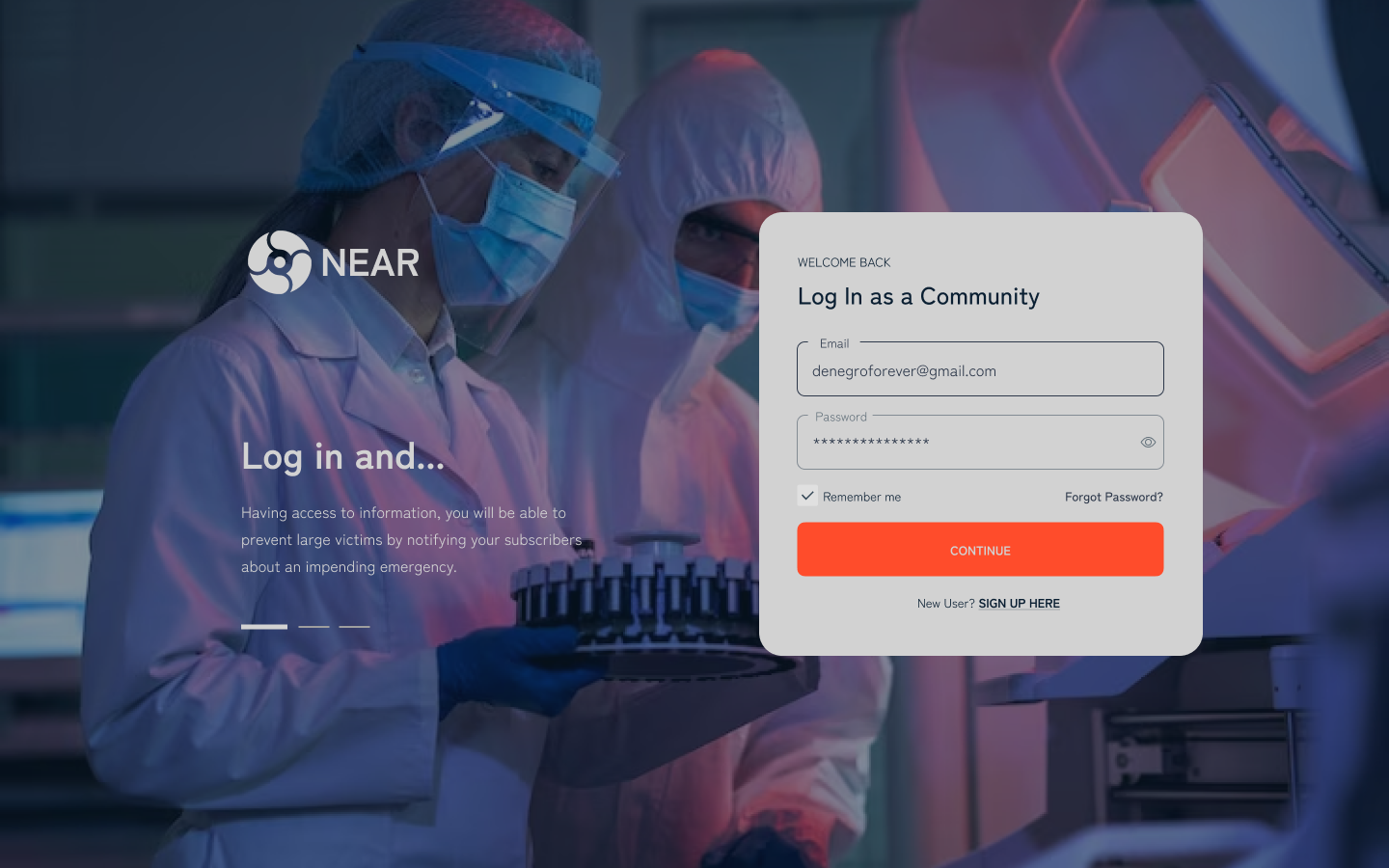


Рисунок А.3 – Макет страницы авторизации сообщества разрабатываемого приложения

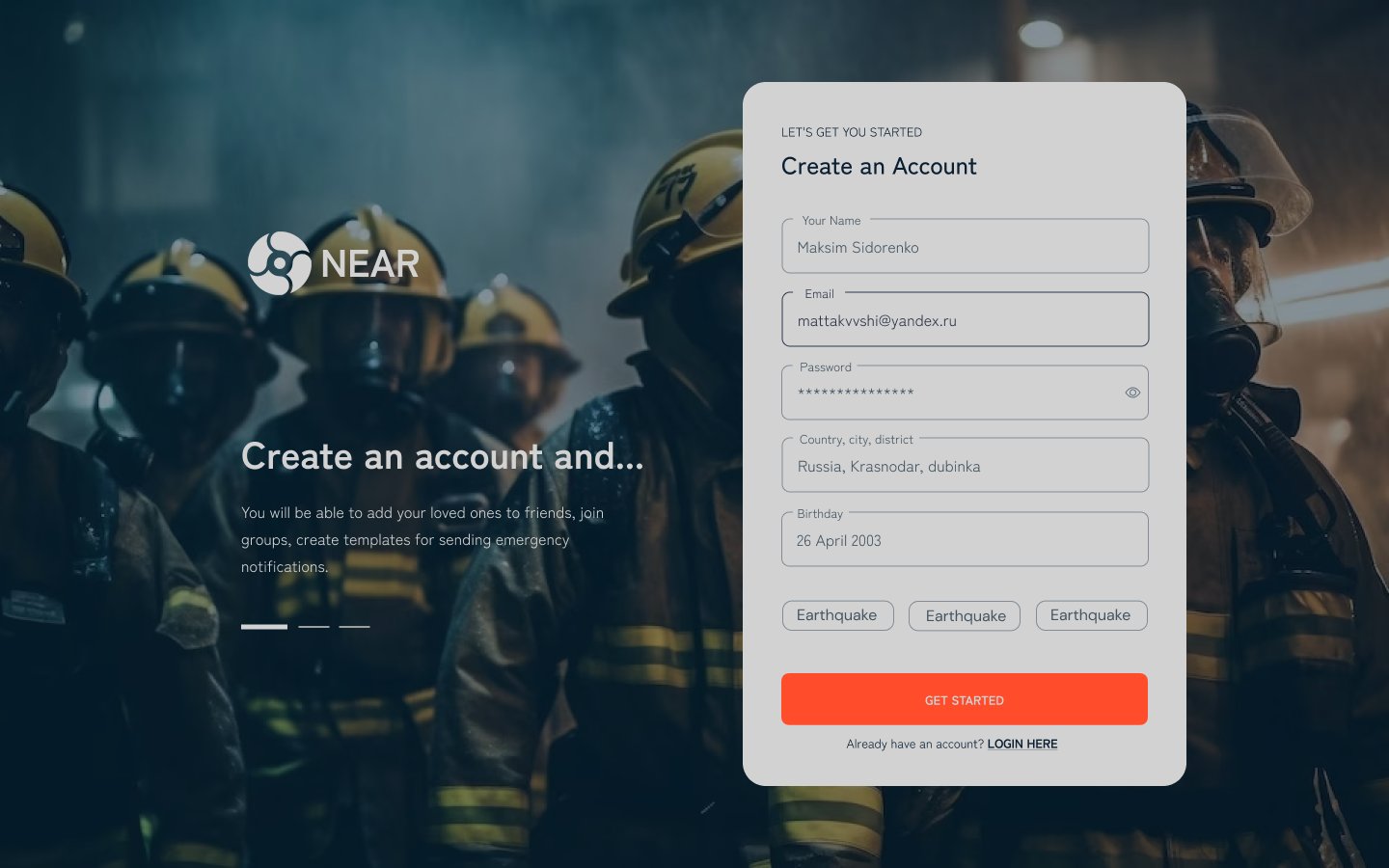


Рисунок А.4 – Макет страницы регистрации пользователя разрабатываемого приложения

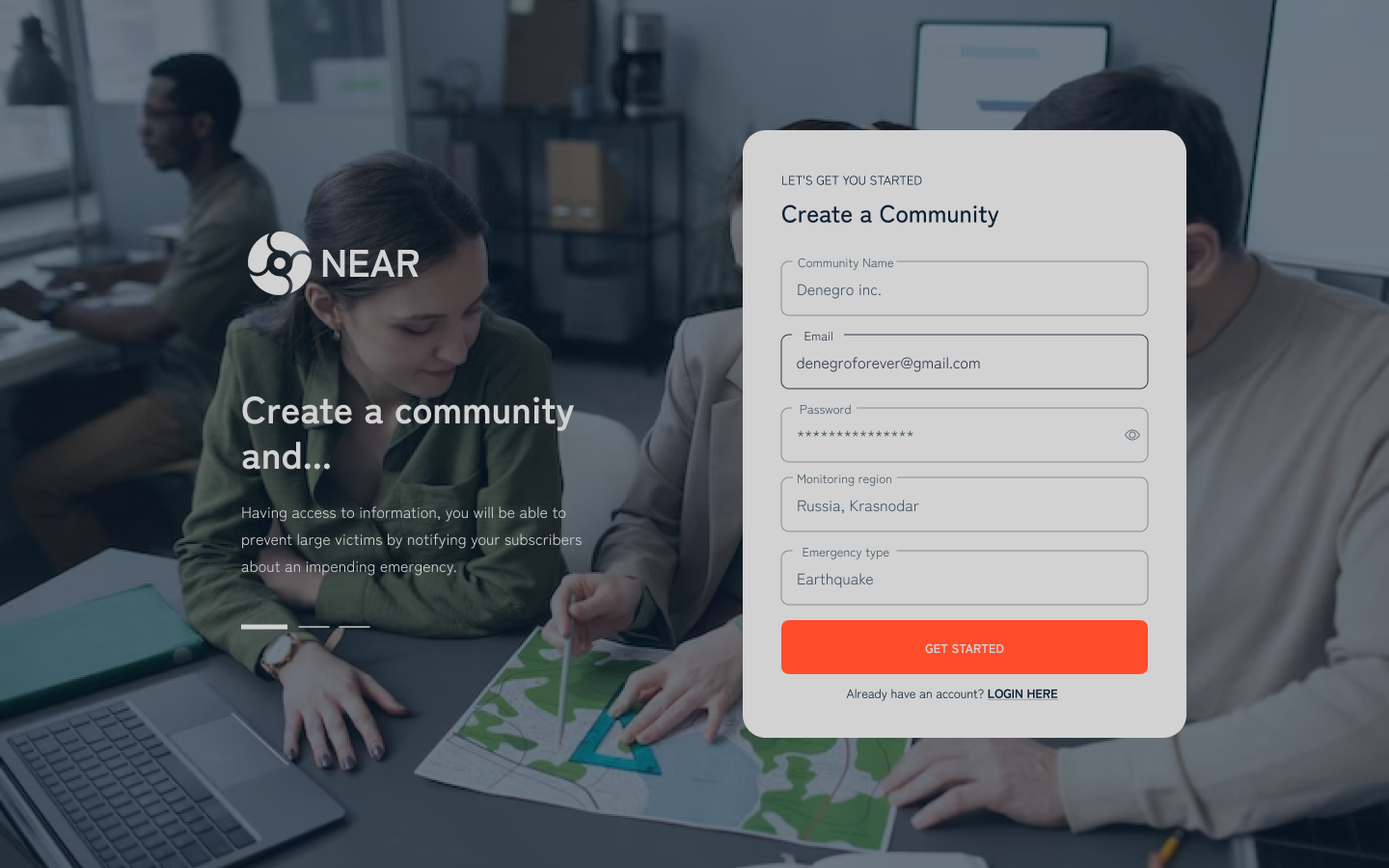


Рисунок А.5 – Макет страницы регистрации сообщества разрабатываемого приложения

**ПРИЛОЖЕНИЕ Б**

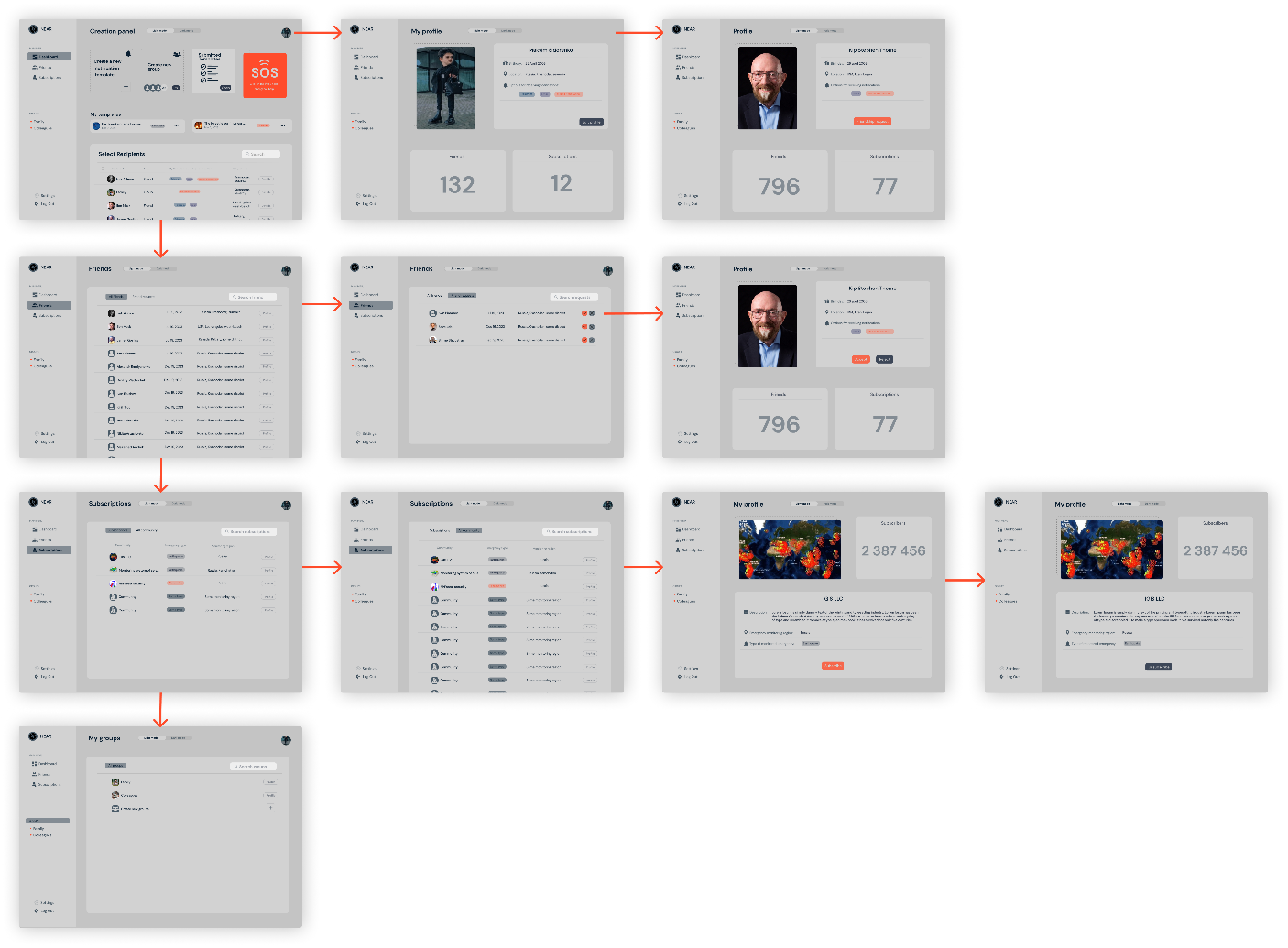
****

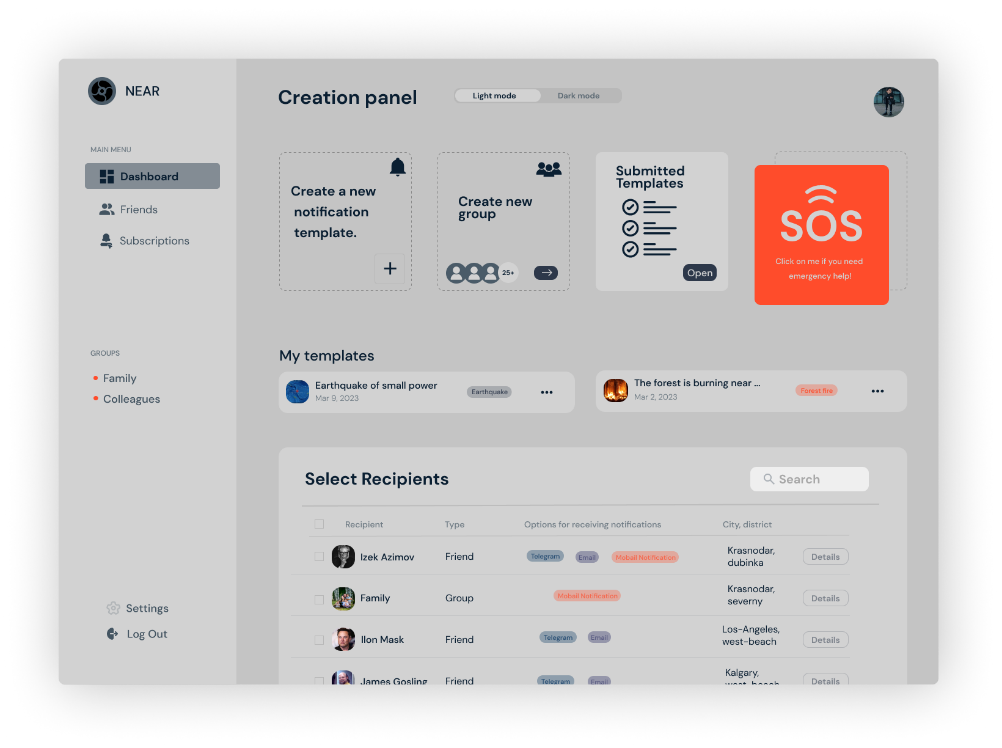
Рисунок Б.1 – Макет экранов пользовательского интерфейса (пользователь) 

Рисунок Б.2 – Макет главного экрана пользовательского интерфейса (пользователь)

**ПРИЛОЖЕНИЕ В**

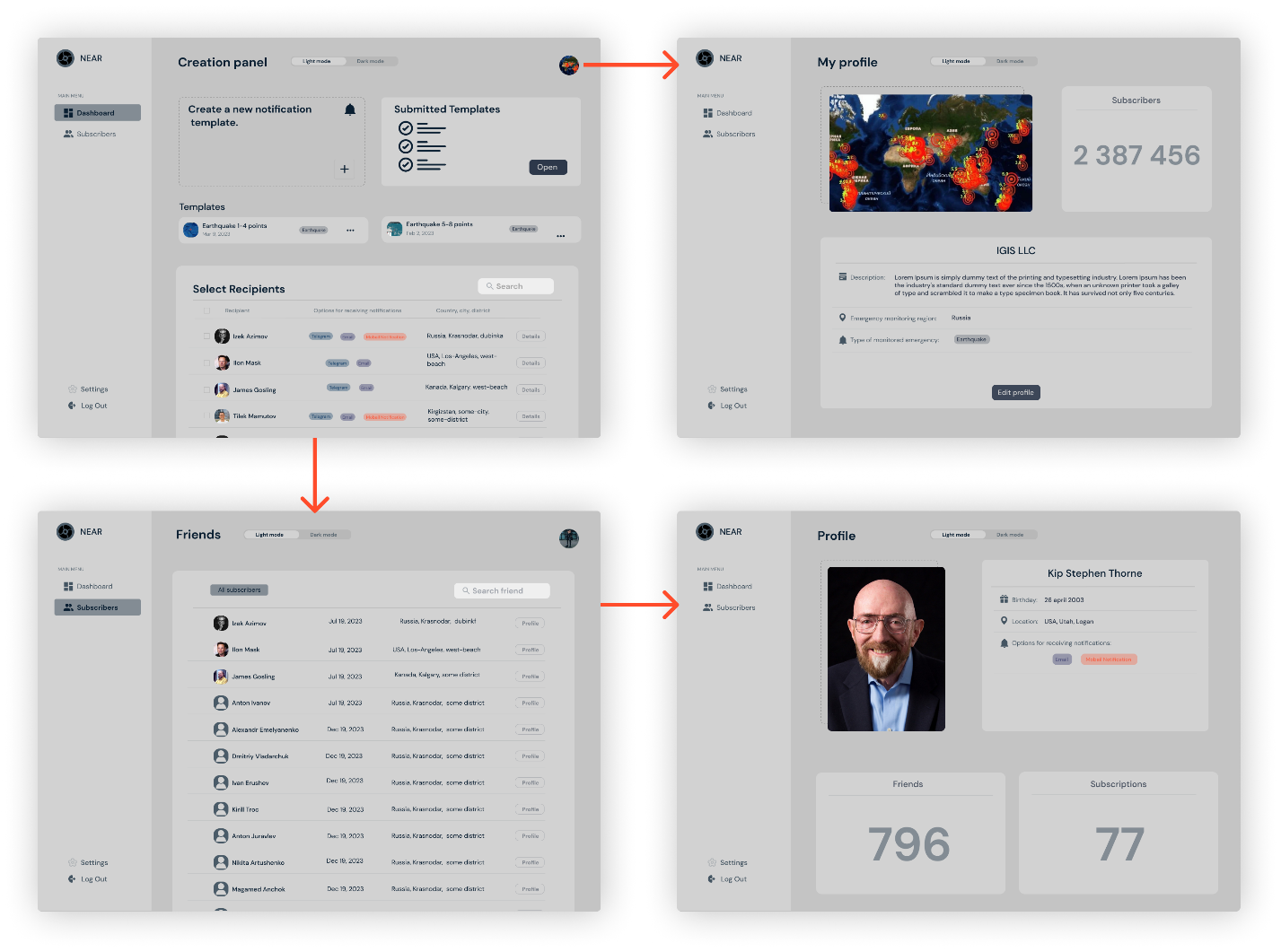


Рисунок В.1 – Макет экранов пользовательского интерфейса (сообщество)

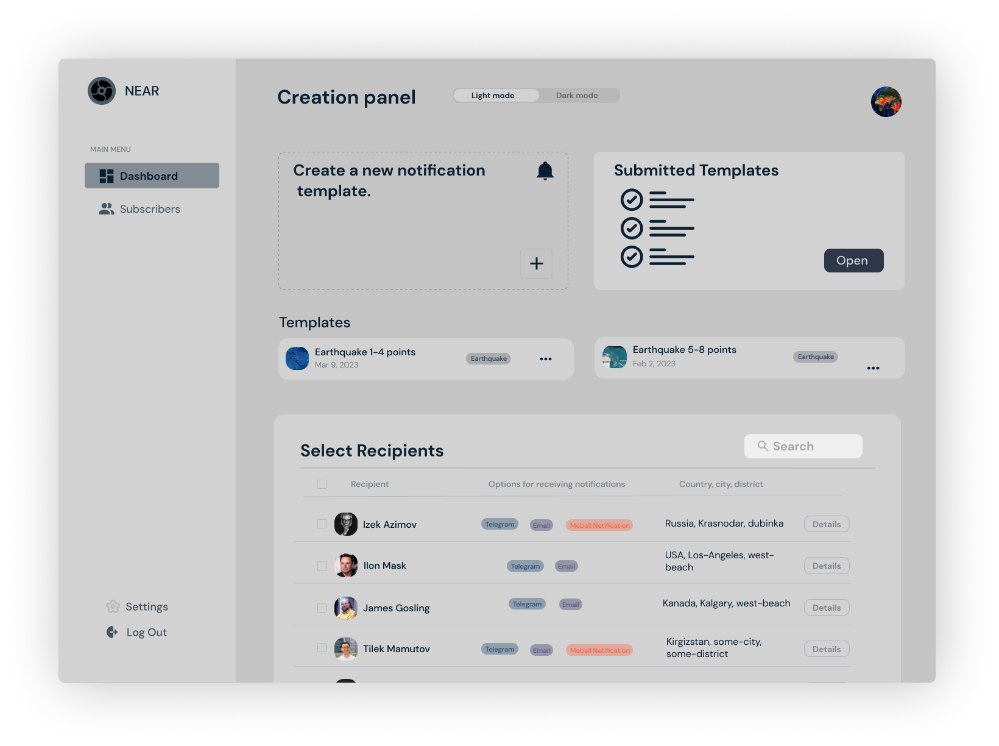


Рисунок В.2 – Макет главного экрана пользовательского интерфейса (сообщество)