**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра компьютерных технологий и систем**

**ПРИМЕНЕНИЕ КОНТЕЙНЕРИЗАЦИИ И ОРКЕСТРАЦИИ В РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

Курсовая работа

Шалаева Павла Витальевича

студента 3 курса 4 группы,

специальность «информатика»

Научный руководитель:

Доцент кафедры КТС

Дайняк В. В.

Минск, 2023

**АННОТАЦИЯ (Times New Roman)**

Шалаев П.В. Применение Контейнеризации И Оркестрации В Разработке Программного Обеспечения: Курсовая работа / Минск: БГУ, 2023. – 49 c.

Рассматриваются задачи методы развертывания веб приложений.

**АНАТАЦЫЯ**

Шаляев П.В. Выкарыстанне кантэйнерызацыі і арэстрацыі ў распрацоўцы праграмнага забеспячэння: курсавая работа / Мінск: БДУ, 2023. – 49 с. Разглядаюцца задачы метады разгортвання веб-прыложэнняў.

**ANNOTATION**

Shalev P.V. Application of Containerization and Orchestration in Software Development: Coursework / Minsk: BSU, 2023. – 49 p. The tasks and methods of deploying web applications are considered.

**РЕФЕРАТ**

**Курсовая работа**, 49 с., 11 рис., 10 источников.

**Ключевые слова:** DOCKER, KUBERNETES, КОНФИГУРАЦИЯ, КРОССПЛАТФОРМЕННОЕ ПРИЛОЖЕНИЕ, ВЕБ ПРИЛОЖЕНИЕ.

**Объекты исследования** – Docker и kubernetes. Использование их возможностей для контейнеризации и орекстрации веб приложений.

**Цель работы** – разработать кроссплатформенное приложение, позволяющее составлять, редактировать и удалять заметки.

**Методы исследования** – методы развертывания веб приложений.

**В результате** – разработано кроссплатформенное веб приложение с учетом использования средств контейнеризации .

**Область применения** – любое веб приложение.

**РЭФЕРАТ**

**Курсавая праца,** 49 с., 11 мал., 10 крыніц.

**Ключавыя словы:** DOCKER, KUBERNETES, КАНФІГУРАЦЫЯ, КРОСПЛАТФОРМЕННАЕ ПРЫЛОЖЕННЕ, ВЕБ-ПРЫЛОЖЕННЕ.

**Аб'екты даследавання** - Docker і Kubernetes. Выкарыстанне іх магчымасцей для кантэйнерызацыі і арэстрацыі веб-прыложэнняў.

**Мэта працы** - распрацоўка кросплатформеннага прыложэння, якое дазваляе складаць, рэдагаваць і выдаляць нататкі.

**Метады даследавання** - метады разгортвання веб-прыложэнняў.

**У выніку** - распрацавана кросплатформеннае веб-прыложэнне з улікам выкарыстання сродкаў кантэйнерызацыі.

**Вобласць ужывання** – любая веб прылада.

**ESSAY**

**Course work,** 49 p., 11 figures, 10 sources.

**Keywords:** DOCKER, KUBERNETES, CONFIGURATION, CROSS-PLATFORM APPLICATION, WEB APPLICATION.

**Research objects** - Docker and Kubernetes. Utilizing their capabilities for containerization and orchestration of web applications.

**Purpose** - develop a cross-platform application allowing users to create, edit, and delete notes.

**Research Methods** - methods of deploying web applications

**Result** - cross-platform web application has been developed, taking into account the use of containerization tools.

**Scope** – any web application.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 8](#_Toc135250024)

[ГЛАВА 1. ТЕХНОЛОГИИ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ 9](#_Toc135250025)

[**1.1** **Проектирование мобильных приложений** 9](#_Toc135250026)

[**1.1.1 Что такое мобильное приложение** 9](#_Toc135250027)

[**1.1.2 Разница между мобильным приложением и веб-приложением** 9](#_Toc135250028)

[**1.1.3 Преимущества и недостатки мобильных приложений** 10](#_Toc135250029)

[**1.1.4 Типы мобильных приложений** 12](#_Toc135250030)

[**1.1.5 Процесс разработки** 13](#_Toc135250031)

[**1.2** **Актуальные языки и библиотеки для создания мобильных приложений** 15](#_Toc135250032)

[**1.2.1** **Нативная разработка мобильных приложений** 15](#_Toc135250033)

[**1.2.2** **Кроссплатформенная разработка мобильных приложений** 16](#_Toc135250034)

[**1.2.3** **Язык Dart** 18](#_Toc135250035)

[**1.2.4** **Библиотека Flutter** 19](#_Toc135250036)

[ГЛАВА 2. РАЗРАБОТКА ТРЕБОВАНИЙ И ПРОЕКТИРОВАНИЕ ПРОТОТИПА ПРИЛОЖЕНИЯ 22](#_Toc135250037)

[**2.1 Обзор существующих приложений для заметок** 22](#_Toc135250038)

[**2.1.1 Evernote** 22](#_Toc135250039)

[**2.1.2 Microsoft OneNote** 23](#_Toc135250040)

[**2.1.3 Google Keep** 23](#_Toc135250041)

[**2.1.4 Apple Notes** 24](#_Toc135250042)

[**2.1.5 Выводы** 25](#_Toc135250043)

[**2.2 Задача проекта** 26](#_Toc135250044)

[**2.3 Разработка требований** 26](#_Toc135250045)

[**2.4 Проектирование интерфейса** 27](#_Toc135250046)

[**2.4.1 Инструмент проектирования интерфейсов Figma** 27](#_Toc135250047)

[**2.4.2 Рисование эскизов. Создание Wireframes. Дизайн** 29](#_Toc135250048)

[ГЛАВА 3. РАЗРАБОТКА КРОССПЛАТФОРМЕННОГО МОБИЛЬНОГО ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ FLUTTER 32](#_Toc135250049)

[**3.1 Разработка логики приложения** 32](#_Toc135250050)

[**3.1.1 Основные классы** 32](#_Toc135250051)

[**3.1.2 Пользовательский интерфейс. Реализация** 33](#_Toc135250052)

[ЗАКЛЮЧЕНИЕ 37](#_Toc135250053)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 39](#_Toc135250054)

[ПРИЛОЖЕНИЕ А 40](#_Toc135250055)

[ПРИЛОЖЕНИЕ Б 43](#_Toc135250056)

[ПРИЛОЖЕНИЕ В 46](#_Toc135250057)

# ВВЕДЕНИЕ

Контейнеризация и оркестрация являются ключевыми технологиями в современной разработке программного обеспечения. Контейнеризация позволяет разработчикам упаковывать приложения вместе со всеми необходимыми зависимостями и библиотеками в единый контейнер, что обеспечивает их портативность и согласованность на различных средах разработки, тестирования и производства. Основным преимуществом этого подхода является то, что он упрощает развертывание и управление приложениями, снижает риск ошибок, связанных с различиями в окружающей среде, и ускоряет процесс разработки и доставки ПО.

Оркестрация контейнеров, с другой стороны, относится к автоматизации управления, координации и масштабирования контейнеризированных приложений. Инструменты оркестрации, такие как Kubernetes, Docker Swarm и другие, позволяют автоматически управлять жизненным циклом контейнеров, обеспечивать их масштабирование, обновление и обеспечение необходимыми ресурсами.

#### Цели и задачи курсовой работы

Целью данной курсовой работы является изучение и анализ роли контейнеризации и оркестрации в процессе разработки программного обеспечения. Ключевые задачи включают:

1. Исследовать основные принципы и методы контейнеризации.
2. Анализировать роль и механизмы оркестрации контейнеров.
3. Рассмотреть практические примеры использования данных технологий в индустрии.
4. Оценить преимущества, недостатки и потенциальные области применения.
5. Предсказать будущие тенденции и развитие данных технологий.

# ГЛАВА 1. Теоретический обзор

* 1. **Определение контейнеризации и оркестрации**

### **1.1.1 Что такое контейнеризация**

#### Определение контейнеризации

Контейнеризация — это метод виртуализации на уровне операционной системы, который позволяет запускать и управлять приложениями и их зависимостями в изолированных средах, называемых контейнерами. Контейнеры облегчают разработку, тестирование и развертывание приложений, обеспечивая согласованность окружения на всех этапах жизненного цикла разработки ПО. Они делят ядро операционной системы хоста, но могут быть изолированы от других контейнеров и хостовой системы. Примеры технологий контейнеризации включают Docker, LXC и другие.

#### Преимущества использования контейнеров

* Портативность: Контейнеры обеспечивают согласованность окружения на разных средах, что упрощает перенос приложений.
* Эффективность: Контейнеры требуют меньше ресурсов, чем традиционные виртуальные машины, поскольку разделяют ядро ОС хоста и не требуют отдельной операционной системы для каждого экземпляра.
* Быстрое развертывание и масштабируемость:
* могут быть быстро запущены и остановлены, что улучшает масштабируемость и гибкость при управлении приложениями.
* Изоляция и безопасность: Каждый контейнер изолирован и имеет собственные ресурсы, что повышает безопасность и уменьшает риск конфликтов между приложениями.

#### 1.1.2 Определение оркестрации контейнеров

Оркестрация контейнеров — это автоматизация развертывания, управления, масштабирования, сетевого взаимодействия и мониторинга контейнеризированных приложений. Это включает в себя управление жизненным циклом контейнеров, от развертывания до утилизации, и обеспечение высокой доступности и отказоустойчивости приложений.

* + - 1. Оркестратор выполняет следующие функции:

1. Развертывание приложений: Оркестратор автоматизирует процесс развертывания контейнеров, управляя их запуском на доступных узлах.
2. Масштабирование: Он обеспечивает возможность масштабирования приложений, управляя добавлением или удалением контейнеров в зависимости от текущей нагрузки.
3. Управление ресурсами: Оркестратор следит за доступностью ресурсов на узлах и эффективно распределяет нагрузку между ними, чтобы обеспечить оптимальную производительность приложений.
4. Отслеживание состояния контейнеров: Он следит за состоянием контейнеров, перезапускает их в случае сбоев, а также обеспечивает их перераспределение для улучшения отказоустойчивости.
5. Сетевое взаимодействие: Оркестратор управляет сетевым взаимодействием между контейнерами, обеспечивая связь между различными компонентами приложения.

### **1.2** Применение Docker в разработке ПО

#### **1.2.1 Основные принципы Docker**

Docker предоставляет стандартизированную платформу для создания, доставки и выполнения контейнеров. Контейнеры - это легковесные, автономные и переносимые окружения, включающие в себя все необходимое для запуска приложения: код, библиотеки, зависимости и настройки. Принципы изоляции и переносимости делают Docker эффективным инструментом в разработке и поставке программного обеспечения.

#### **1.2.2 Управление зависимостями:**

#### В мире разработки программного обеспечения эффективное управление зависимостями является фундаментальным аспектом. Docker позволяет изолировать приложение и его зависимости в контейнер, создавая своего рода "запечатанную" среду. Это достигается за счет включения всех необходимых библиотек, инструментов и зависимостей внутрь контейнера.

Когда разработчик создает Docker-образ, включающий приложение и его зависимости, это создает надежное и воспроизводимое окружение. Такое окружение можно легко передавать между разработчиками, тестировщиками и даже между различными средами развертывания. Это исключает проблемы, связанные с различиями в версиях библиотек и окружениях, что обычно возникает при передаче кода между разработчиками или при развертывании на разных серверах.

1.2.3 Упрощение развертывания приложений в Docker:

Одним из ключевых преимуществ Docker является его способность упрощать процесс развертывания приложений. Когда приложение упаковано в контейнер, оно становится переносимым и автономным. Весь необходимый софт и настройки включаются в контейнер, и разработчик может быть уверен, что приложение будет работать так, как предполагается, независимо от среды выполнения.

Docker-образы могут быть легко переданы между разными средами развертывания, будь то локальная машина разработчика, тестовый сервер или облачная инфраструктура. Это дает уверенность в том, что приложение будет вести себя одинаково в любой среде, что является важным фактором при масштабировании и управлении сложными приложениями.

1.2.**4** Масштабируемость и управление ресурсами в Docker:

Docker обеспечивает легкость масштабирования приложений. Когда приложение упаковано в контейнер, его можно быстро развернуть в нескольких экземплярах на одном хосте или даже на нескольких хостах в кластере. Это дает возможность эффективно использовать ресурсы сервера и масштабировать приложение в зависимости от изменяющихся нагрузок.

Кроме того, Docker предоставляет инструменты для управления ресурсами контейнеров. Это включает в себя возможность ограничивать использование CPU, памяти и других ресурсов каждым контейнером. Такой подход позволяет более гибко настраивать производительность приложения и предотвращать ситуации, когда одно приложение слишком сильно влияет на другие контейнеры на том же хосте.

1.2.**5** Упрощение тестирования по с помощью Docker:

Контейнеры Docker обеспечивают легковесное и изолированное окружение, в котором приложение может выполняться в условиях, максимально приближенных к реальному производственному окружению.

Эта изоляция обеспечивает повторяемость и надежность тестирования, поскольку контейнер содержит все необходимые зависимости и библиотеки, исключая тем самым потенциальные конфликты среды выполнения. Кроме того, Docker обеспечивает возможность легко создавать образы контейнеров, содержащие как сам код приложения, так и его окружение.

Виртуализация уровня операционной системы, которую предоставляет Docker, позволяет разработчикам и тестировщикам работать в одном и том же окружении. Это существенно снижает вероятность того, что проблемы, выявленные в тестовом окружении, не воспроизведутся в продакшене из-за различий в среде выполнения.

Также стоит отметить возможность быстрого масштабирования тестовых сред с использованием Docker Compose. Этот инструмент позволяет описывать и запускать многоконтейнерные приложения, что полезно при тестировании взаимодействия между различными компонентами системы.

1.3 Применение Kubernetes в разработке ПО

1.3.1 Как Kubernetes управляет контейнерами и обеспечивает масштабируемость и отказоустойчивость

Kubernetes - это открытая система управления контейнерами, которая автоматизирует развертывание, масштабирование и управление приложениями в контейнерах. Kubernetes обеспечивает масштабируемость и отказоустойчивость при помощи:

1. Оркестрация контейнеров
2. Масштабирование
3. Отказоустойчивость
4. Самоисцеление

1.3.2 **Оркестрация контейнеров**

Kubernetes функционирует как оркестратор, обеспечивая автоматизированный контроль над жизненным циклом контейнеров. Это достигается путем описания желаемого состояния системы в виде конфигурационных файлов, которые затем интерпретируются и реализуются Kubernetes.

1.3.**3** Масштабирование

При использовании Kubernetes (k8s) горизонтальное масштабирование часто выполняется с помощью контроллеров развертывания и автоматического масштабирования (Autoscaling). Вот как это может быть сделано:

1. Развертывание с использованием ReplicaSets:
   * Создайте ваше приложение, определив шаблон Pod в манифесте ReplicaSet. Укажите желаемое количество реплик (экземпляров) приложения.

yamlCopy code

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: my-app

spec:

replicas: 3

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-container

image: my-image

* + Примените манифест с использованием kubectl apply -f filename.yaml.

1. Горизонтальное масштабирование вручную:
   * Если вы хотите изменить количество реплик вручную, обновите значение replicas в манифесте ReplicaSet и примените его снова:

yamlCopy code

spec:

replicas: 5

* + Выполните kubectl apply -f filename.yaml для обновления.

1. Горизонтальное масштабирование автоматически:
   * Для автоматического горизонтального масштабирования используйте Horizontal Pod Autoscaler (HPA).
   * Создайте манифест HPA, указав метрики и цели:

yamlCopy code

apiVersion: autoscaling/v2beta2

kind: HorizontalPodAutoscaler

metadata:

name: my-app-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: ReplicaSet

name: my-app

minReplicas: 3

maxReplicas: 10

metrics:

- type: Resource

resource:

name: cpu

targetAverageUtilization: 50

* + Примените манифест с использованием kubectl apply -f filename.yaml.

Теперь, при возрастании нагрузки на ваше приложение, HPA автоматически увеличит количество реплик, а при уменьшении нагрузки - уменьшит.

Таким образом, горизонтальное масштабирование с использованием ReplicaSets и HPA в Kubernetes обеспечивает более гибкое и эффективное управление ресурсами вашего приложения.

**1.3.4 Отказоустойчивость k8s:**

Распределение контейнеров по различным узлам и регионам в экосистеме Kubernetes служит важным механизмом обеспечения отказоустойчивости. Этот метод направлен на предотвращение потери работоспособности сервиса в случае возникновения сбоев. Путем динамического распределения экземпляров контейнеров по различным физическим и географическим точкам сети, Kubernetes стремится создать надежную иустойчивую инфраструктуру для развертывания приложений.

Эта стратегия отказоустойчивости основана на концепции изоляции и диверсификации, позволяя минимизировать воздействие возможных сбоев на работоспособность системы. При наличии географического распределения контейнеров по разным регионам, Kubernetes обеспечивает возможность переноса рабочей нагрузки в случае недоступности определенных узлов или регионов, что поддерживает непрерывность обслуживания даже при наличии нештатных ситуаций.

**1.3.5 Самоисцеление**

Самоисцеление в контексте Kubernetes представляет собой механизм автоматического обнаружения и восстановления от сбоев в работе контейнеров. Kubernetes, как система управления контейнеризированными приложениями, обеспечивает высокую доступность и надежность путем постоянного мониторинга состояния контейнеров и их компонентов.

Автоматизированный мониторинг позволяет Kubernetes наблюдать за различными параметрами, такими как использование ресурсов, состояние сети и доступность сервисов. В случае обнаружения сбоя или неправильной работы контейнера, Kubernetes принимает меры по его восстановлению.

Механизм самоисцеления Kubernetes включает в себя перезапуск контейнера или создание нового экземпляра приложения для замены поврежденного. Это происходит с минимальным воздействием на работу всей системы, поскольку Kubernetes обеспечивает динамическое масштабирование и управление ресурсами.

**1.3.6 Принципы и компоненты Kubernetes для разработчиков**

Для разработчика, использующего K8s важны следущие компаненты:

1.Поды:

Поды в Kubernetes являются основной исполняемой единицей, объединяющей контейнеры. Разработчики взаимодействуют с подами, определяя необходимые ресурсы и настройки для запуска и поддержки своих приложений в кластере.

2.Сервисы:

Kubernetes предоставляет сервисы, обеспечивающие коммуникацию между контейнерами внутри кластера. Разработчики используют сервисы для обеспечения надежного взаимодействия своих приложений, создавая сетевые связи между компонентами.

3.Конфигурация и хранилище секретов:

Разработчики активно используют механизмы хранения конфигураций и секретов в Kubernetes. Это позволяет им безопасно сохранять параметры приложений, такие как пароли и ключи, соблюдая принципы безопасности.

4.Деплойменты:

Для управления обновлениями и масштабированием приложений разработчики определяют деплойменты. Эти конфигурации позволяют им эффективно управлять жизненным циклом приложений в кластере Kubernetes.

5.Мониторинг и логирование:

Разработчики активно взаимодействуют с механизмами мониторинга и логирования Kubernetes. Это помогает им отслеживать производительность приложений, выявлять проблемы и быстро реагировать на события в кластере.

6.API и манифесты:

Описание требуемых ресурсов и параметров происходит через манифесты, используя Kubernetes API. Разработчики определяют структуру и параметры своих приложений, что обеспечивает автоматизацию процесса развертывания и управления приложениями в кластере.

**1.4 Интеграция и автоматизация сборки и развертывания**

**1.4.1** Автоматизация процессов сборки:

Автоматизация процессов сборки в современной разработке программного обеспечения является ключевым элементом для повышения эффективности, стабильности и консистентности развертывания приложений. В этом контексте внедрение систем сборки, таких как Jenkins, Travis CI и GitLab CI, играет важную роль.

Jenkins, Travis CI и GitLab CI предоставляют средства для автоматизации процессов компиляции кода, тестирования и развертывания приложений. Они позволяют создавать цепочки задач, которые выполняются автоматически при каждом изменении в репозитории кода. Это уменьшает ручной труд и ускоряет цикл разработки, обеспечивая более быструю поставку нового функционала.

Контейнеризация с использованием Docker представляет собой ещё один важный аспект автоматизации. Docker позволяет упаковывать приложение, его зависимости и конфигурацию в контейнер, создавая таким образом изолированное окружение. Это обеспечивает консистентность среды сборки и развертывания между различными этапами разработки и даже разными окружениями, такими как тестовое и продуктивное.

Применение этих методов не только улучшает процессы сборки, но также способствует повышению стабильности и предсказуемости развертывания приложений. Кроме того, автоматизация сборки позволяет сократить вероятность ошибок и ускорить обнаружение проблем в коде на ранних этапах разработки.

**1.4.2** Автоматизация процессов сборки:

Использование Docker и Kubernetes (k8s) в контексте автоматизации тестирования обеспечивает еще более высокую степень гибкости, масштабируемости и управления контейнеризированными приложениями.

Юнит-тестирование, проводимое в контейнерах Docker, позволяет разработчикам создавать изолированные среды для тестирования отдельных компонентов приложения. Кроме того, Kubernetes может использоваться для автоматического масштабирования контейнеров с тестируемым кодом, что упрощает параллельное выполнение большого количества юнит-тестов.

Интеграционное тестирование может быть улучшено с использованием Kubernetes, предоставляя возможность динамического развертывания и управления сервисами в контейнерах. Это облегчает проверку взаимодействия между различными компонентами приложения в реальных условиях.

Системное тестирование в Kubernetes может включать создание целых тестовых сред, воссоздаваемых в виде подов. Kubernetes управляет развертыванием этих сред, предоставляя средства для оркестрации и контроля окружений системного тестирования.

1.4.3**Развертывание с Kubernetes:**

Развертывание с использованием Kubernetes представляет собой эффективный способ обеспечения масштабируемости и управления приложением в современных инфраструктурах. Для начала процесса контейнеризации приложения необходимо упаковать его в контейнер, что обеспечивает изолированное и переносимое окружение. Это позволяет приложению работать в любом окружении, где установлен Docker или другой контейнерный движок.

Процесс контейнеризации также способствует легкости развертывания, поскольку контейнеры включают в себя все необходимые зависимости, минимизируя различия между различными средами выполнения. Таким образом, создается единое и консистентное окружение, что упрощает управление приложением.

Для развертывания приложения на кластере Kubernetes необходимо создать YAML-манифесты. Эти файлы содержат конфигурацию, определяющую необходимые ресурсы, такие как поды, службы, конфигмапы и секреты. YAML-манифесты позволяют описать желаемое состояние кластера, а Kubernetes автоматически обеспечивает достижение этого состояния.

Поды представляют собой минимальные единицы развертывания, содержащие один или несколько контейнеров. Службы обеспечивают стабильный доступ к подам, предоставляя DNS-имена и уникальные IP-адреса. Внешние ресурсы, такие как внешние тома данных или балансировщики нагрузки, также могут быть интегрированы через соответствующие манифесты.

1.4.3**Развертывание с Kubernetes:**

Горизонтальное масштабирование подразумевает возможность увеличения или уменьшения количества экземпляров контейнеров в зависимости от изменения нагрузки. Этот процесс позволяет системе эффективно адаптироваться к переменной нагрузке, обеспечивая при этом стабильную и высокую производительность.

Оркестрация, в данном контексте, выступает в роли умного регулировщика, координирующего работу контейнеров в зависимости от текущих потребностей системы. Она автоматизирует процессы управления ресурсами, позволяя системе быстро масштабироваться вверх или вниз.

Важным аспектом горизонтального масштабирования является обеспечение равномерного распределения нагрузки между экземплярами контейнеров. Это гарантирует оптимальное использование ресурсов и предотвращает возможные узкие места в работе системы.

1.4.4 **Мониторинг и логирование:**

Интеграция мониторинга и централизованный сбор логов являются важными компонентами современной системы управления и обеспечения безопасности в среде Kubernetes. Реализация эффективных инструментов мониторинга, таких как Prometheus, предоставляет возможность постоянного отслеживания состояния кластера, что особенно важно в условиях динамичных изменений и масштабирования приложений.

Prometheus обеспечивает сбор и агрегацию метрик, предоставляя операторам и инженерам подробную информацию о производительности, доступности и общем состоянии кластера Kubernetes. Этот инструмент дает возможность оперативно реагировать на потенциальные проблемы и улучшать работу системы.

В контексте централизованного сбора логов использование Elasticsearch и Kibana становится ключевым элементом для эффективного анализа и мониторинга приложения. Elasticsearch обеспечивает хранение и поиск логов, а Kibana предоставляет удобный интерфейс для визуализации и анализа данных. Эта интеграция позволяет операторам быстро обнаруживать и решать проблемы в приложении, а также проводить аналитику для оптимизации работы системы.

1.4.5 **Масштабирование и обновление:**

Горизонтальное масштабирование в контексте Kubernetes представляет собой метод автоматического увеличения или уменьшения количества экземпляров приложения в зависимости от текущей нагрузки. Этот механизм основан на динамическом добавлении новых подов или уменьшении их числа для обеспечения оптимального распределения нагрузки. Kubernetes предоставляет возможность настройки автоматического горизонтального масштабирования на основе метрик, таких как загрузка CPU или количество запросов.

Blue-Green и Canary развертывание являются стратегиями обновления приложений, направленными на минимизацию времени простоя и рисков для конечных пользователей. В случае Blue-Green развертывания новая версия приложения разворачивается параллельно с текущей (синей и зеленой средами соответственно), и трафик перенаправляется с одной среды на другую после успешного тестирования новой версии. Это позволяет мгновенно переключиться на предыдущую версию в случае возникновения проблем.

Canary развертывание включает пошаговое внедрение новой версии приложения для ограниченной группы пользователей или процентной доли трафика. Это позволяет тщательно мониторить поведение новой версии перед ее полным развертыванием. В случае выявления проблем можно быстро откатить изменения или провести дополнительные коррекции. Обе эти стратегии способствуют снижению рисков и обеспечивают бесперебойную работу приложений при внесении изменений.

1.4.6 Документация

Добавление комментариев к коду является неотъемлемой частью разработки программного обеспечения, особенно при работе с технологиями Docker и Kubernetes. Комментарии облегчают понимание структуры и логики кода, а также обеспечивают более эффективную поддержку и развитие системы.

В контексте Docker, комментарии в коде могут включать описания конфигурации контейнеров, указания на используемые образы, и инструкции по сетевой конфигурации. Это существенно сокращает время, затрачиваемое на понимание того, как приложение упаковано в контейнер, и какие зависимости используются.

В случае Kubernetes, комментарии помогают в понимании структуры манифестов, используемых для деплоймента и управления приложениями в кластере. Они также могут содержать пояснения к выбору ресурсов, параметров масштабирования, и другим важным аспектам конфигурации Kubernetes.

**1.5** Проблемы и решения в применении контейнеризации и оркестрации

**1.5.1проблемы при использовании Docker и Kubernetes**

Различия окружения разработки и продукции:

* + Проблема: Возможны различия между окружением разработки и производственным окружением, что может привести к проблемам при переносе контейнеризированных приложений.
  + Решение: Используйте инструменты для унификации окружений, такие как Docker Compose для локальной разработки и Helm Charts для стандартизации на стадии развертывания.

1. Управление сетевыми ресурсами:
   * Проблема: Контейнеры могут конфликтовать за сетевые ресурсы, что может привести к проблемам с доступностью и производительностью.
   * Решение: Применение сетевых плагинов Docker и правильная настройка Kubernetes Network Policies помогут управлять сетевыми ресурсами и обеспечить изоляцию контейнеров.
2. Масштабирование и балансировка нагрузки:
   * Проблема: Эффективное масштабирование и балансировка нагрузки могут быть сложными задачами, особенно при динамическом изменении нагрузки.
   * Решение: Kubernetes предоставляет средства для автоматического масштабирования и балансировки нагрузки. Используйте горизонтальное масштабирование (Auto Scaling) и встроенные средства балансировки нагрузки.
3. Сложности в мониторинге и отладке:
   * Проблема: Мониторинг и отладка контейнеризированных приложений могут потребовать специфических инструментов и подходов.
   * Решение: Используйте инструменты мониторинга, такие как Prometheus, и встроенные средства мониторинга Kubernetes. Для отладки проблем предусмотрены инструменты, такие как kubectl exec для доступа к контейнерам.

**1.5.2 альтернативные технологии для контейнеризации и оркестрации**

1.Podman и CRI-O:

* + Отличия: Podman предоставляет альтернативу Docker без необходимости демона, тогда как CRI-O фокусируется на минимальной реализации стандартов для интеграции с Kubernetes.

1. Docker Swarm:
   * Отличия: Это интегрированное решение Docker для оркестрации, проще в использовании, но может быть менее масштабируемым по сравнению с Kubernetes.
2. Amazon ECS и Microsoft Azure Container Instances:
   * Отличия: Эти облачные сервисы предоставляют управление контейнерами без необходимости настройки и управления оркестратором.

# ГЛАВА 2. РАЗРАБОТКА ТРЕБОВАНИЙ И ПРОЕКТИРОВАНИЕ ПРОТОТИПА ПРИЛОЖЕНИЯ

**2.1 Обзор существующих приложений для заметок**

### **2.1.1 Evernote**

Evernote - это одно из самых известных приложений для заметок, которое позволяет пользователям создавать, организовывать и синхронизировать заметки на разных устройствах. Оно поддерживает текстовые заметки, заметки с изображениями, записи голоса, задачи и другие форматы.

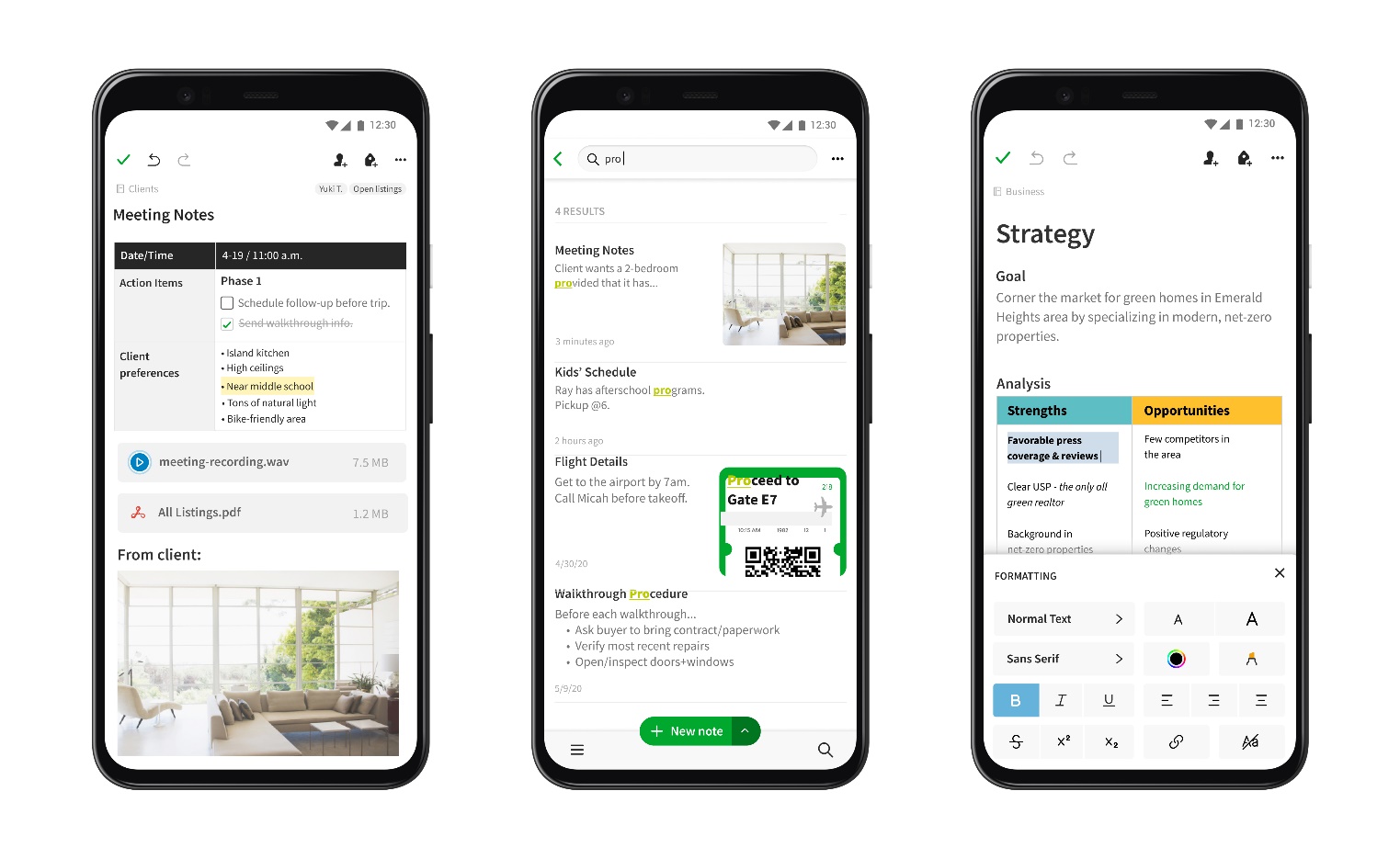


Рисунок 2.1 – Интерфейс приложения Evernote

### **2.1.2 Microsoft OneNote**

OneNote - это приложение для заметок от Microsoft, которое предлагает широкие возможности для создания и организации заметок. Оно позволяет пользователям создавать текстовые, голосовые и рукописные заметки, добавлять изображения и видео, а также синхронизировать их между устройствами.

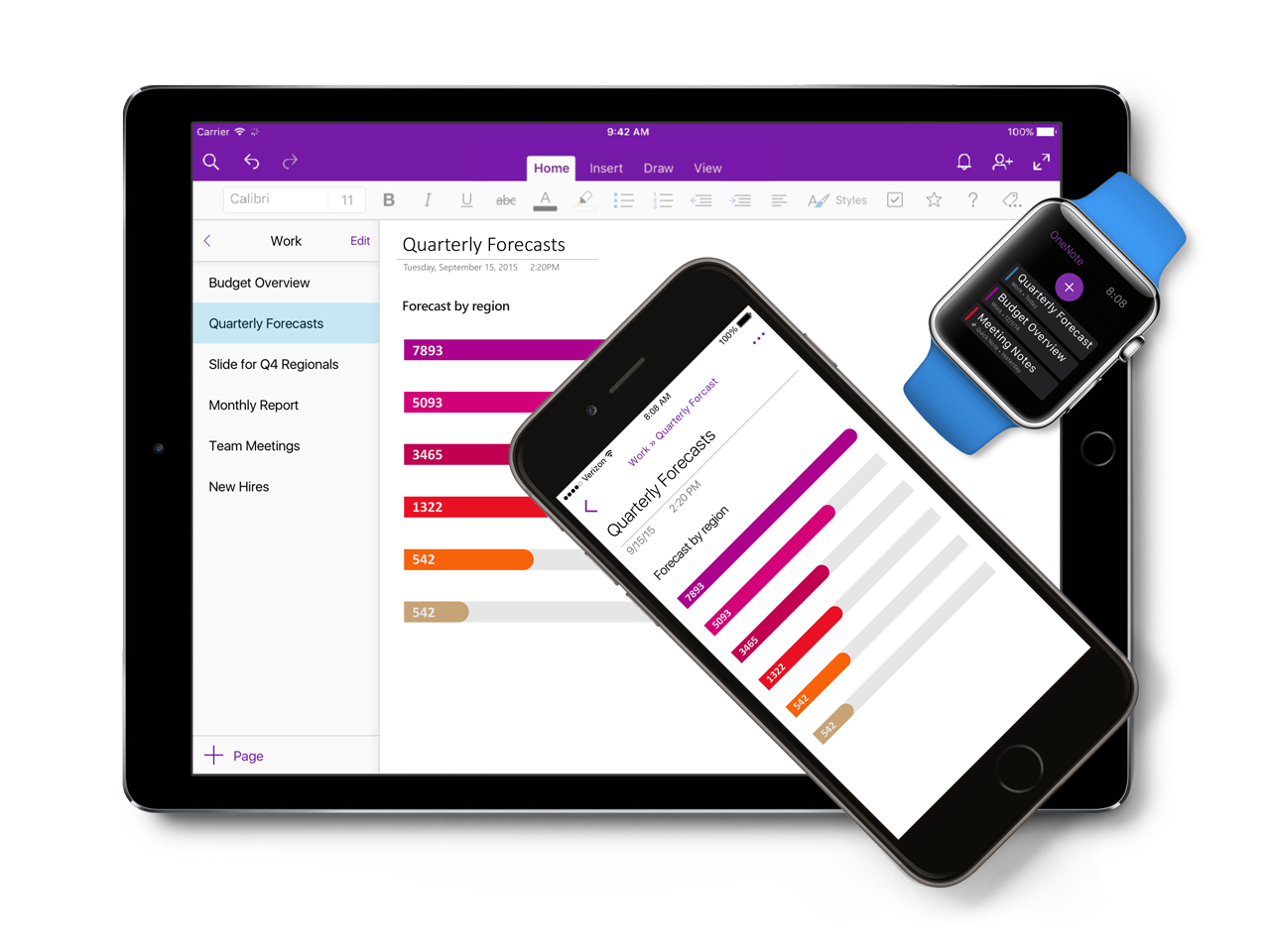


Рисунок 2.2 – Интерфейс приложения OneNote

### **2.1.3 Google Keep**

Google Keep - это простое и интуитивно понятное приложение для заметок, разработанное Google. Оно позволяет создавать заметки, списки дел, сохранять фотографии и голосовые заметки. Заметки синхронизируются с облаком Google и могут быть доступны на разных устройствах.

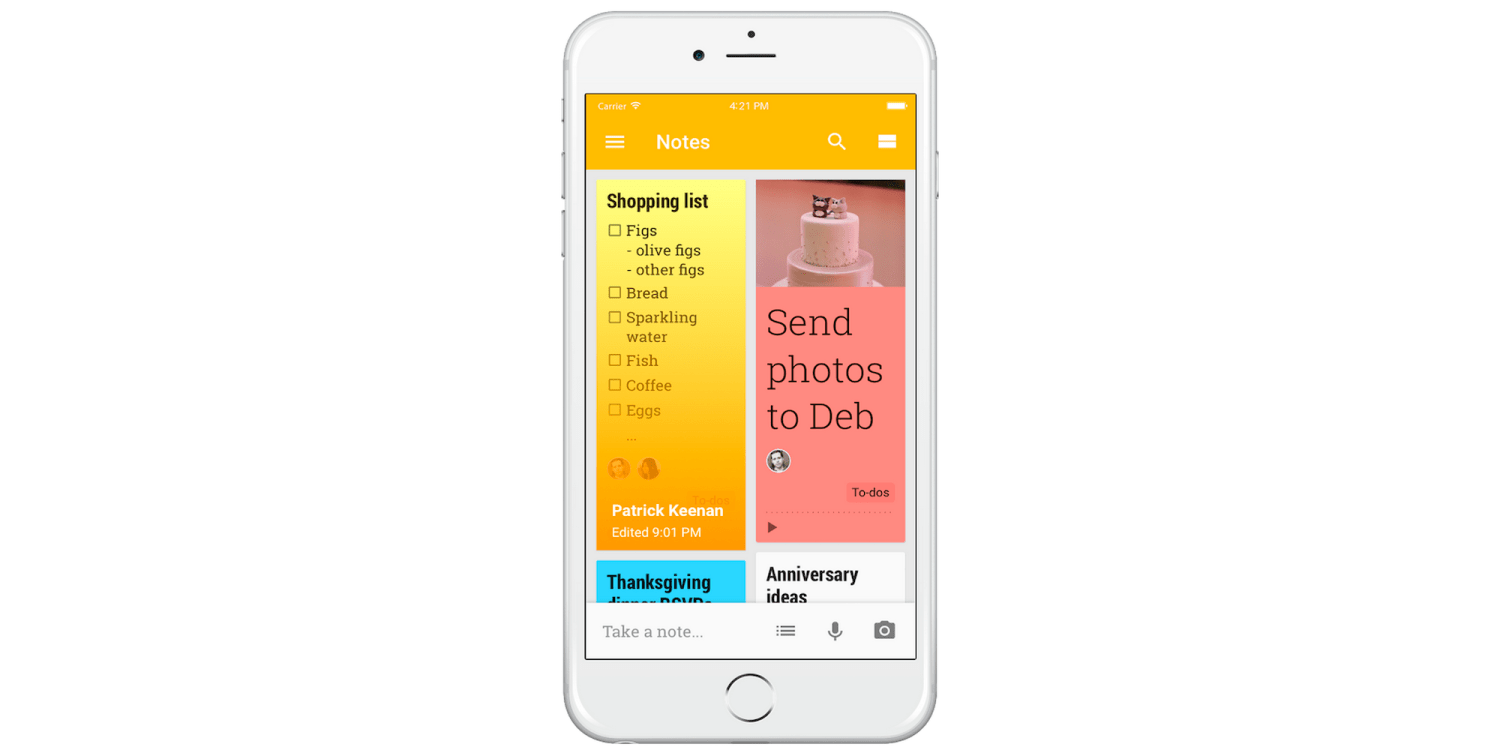


Рисунок 2.3 – Интерфейс приложения Google Keep

### **2.1.4 Apple Notes**

Apple Notes - это встроенное приложение для заметок на устройствах Apple, таких как iPhone и iPad. Оно предлагает простой интерфейс и позволяет создавать текстовые заметки, рисовать, добавлять фотографии и другие медиафайлы. Заметки синхронизируются через iCloud и могут быть доступны на разных устройствах Apple.

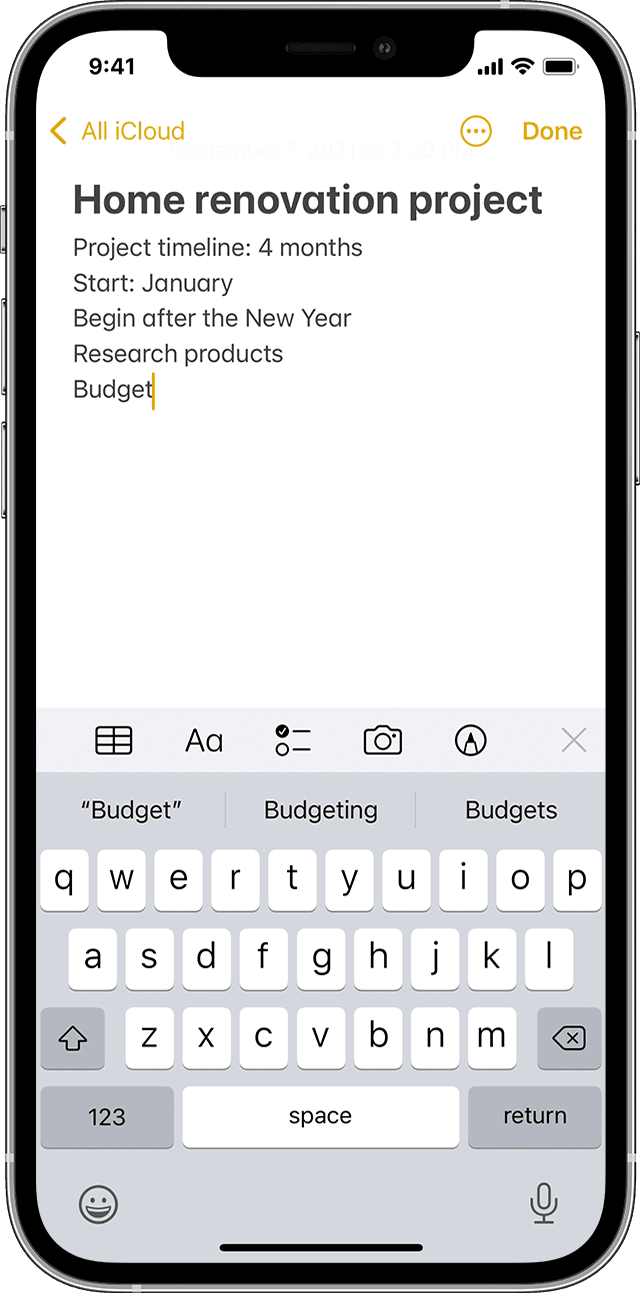


Рисунок 2.4 – Интерфейс приложения Apple Notes

### **2.1.5 Выводы**

После обзора и анализа существующих приложений для заметок, можно сделать следующие выводы:

* Организация заметок: Рассмотренные приложения предлагают различные подходы к организации заметок. Некоторые приложения, такие как Evernote и Microsoft OneNote, предоставляют возможность создания и организации заметок в виде блокнотов, папок и меток, что облегчает структурирование информации. Другие, например, Google Keep, предоставляют более простую структуру с возможностью добавления цветных меток и использования тегов для организации заметок.
* Функциональность заметок: Все рассмотренные приложения позволяют создавать и редактировать заметки, добавлять текст, фотографии, голосовые записи и другие медиафайлы. Они также предлагают возможность добавления напоминаний, тегов, маркеров выполнения и сортировки заметок по различным параметрам.
* Синхронизация и доступность: Все приложения предлагают синхронизацию заметок через облачные сервисы, такие как Evernote Cloud, Microsoft OneDrive, Google Drive и iCloud. Это позволяет пользователям получать доступ к своим заметкам на разных устройствах и синхронизировать изменения между ними.
* Интерфейс и пользовательский опыт: Важным аспектом является простота и удобство использования приложений. Некоторые приложения, например, Google Keep и Apple Notes, предлагают простой и минималистичный интерфейс, что делает их легко понятными и интуитивно понятными для пользователей. Другие приложения, такие как Evernote и Microsoft OneNote, имеют более сложные интерфейсы с большим количеством функций, которые могут быть полезны для пользователей, ищущих расширенные возможности.
* Совместное использование: Некоторые приложения, например, Evernote и Microsoft OneNote, предлагают функции совместного использования, позволяющие пользователям работать над заметками в команде или совместно с другими людьми. Это полезно для проектов, требующих коллаборации и обмена информацией.

В целом, выбор конкретного приложения для заметок зависит от предпочтений и потребностей пользователя. Важно учитывать организацию заметок, функциональность, доступность, интерфейс и возможность совместного использования при выборе приложения, которое лучше всего соответствует требованиям.

**2.2 Задача проекта**

Задача проекта состоим в разработке кроссплатформенного мобильного приложения при помощи языка Dart и библиотеки Flutter. Приложение позволяет создавать списки с заметками, редактировать и удалять их.

**2.3 Разработка требований**

Назначение: приложение предоставляет базовый инструментарий для удобного формирования, редактирования, удаления заметок.

Пользователи: владельцы мобильных устройств.

Функциональные требования (определяют поведение системы):

* Добавление / редактирование / удаление заметок;
* Создание аккаунта пользователя / авторизация / вход
* Сохранение заметок в базе данных
* Привязка заметок к аккаунту пользователя

Нефункциональные требования (описывают характер поведения системы)

* Простота в использовании для любого пользователя;
* Интуитивно понятный интерфейс.

**2.4 Проектирование интерфейса**

### **2.4.1 Инструмент проектирования интерфейсов Figma**

Прототипирование приложения производилось с использованием специализированного программного обеспечения, в качестве которого выступала Figma [10].

Figma - это популярный инструмент для проектирования интерфейсов и коллаборации при разработке мобильных и веб-приложений. Он предоставляет широкие возможности для создания и прототипирования дизайна пользовательского интерфейса.

Преимущества использования Figma в процессе проектирования приложений:

* Онлайн-коллаборация: Figma позволяет нескольким дизайнерам и разработчикам работать над проектом одновременно. Это упрощает командную работу, позволяет быстро обмениваться идеями, получать обратную связь и вносить изменения в режиме реального времени.
* Мультиплатформенность: Figma доступен на разных платформах, включая веб-версию, приложения для Windows и macOS, что делает его удобным для работы на разных устройствах.
* Прототипирование: С помощью Figma можно создавать интерактивные прототипы приложений, что позволяет протестировать пользовательский опыт и взаимодействие с интерфейсом до реализации.
* Компонентный подход: Figma позволяет создавать и использовать компоненты, которые можно повторно использовать в проекте. Это упрощает поддержку и изменение дизайна, а также обеспечивает единообразие внешнего вида элементов интерфейса.
* Возможности совместной работы с разработчиками: Figma предоставляет различные инструменты и возможности экспорта, которые облегчают совместную работу с разработчиками при передаче дизайна и спецификаций интерфейса.
* Интеграция с другими инструментами: Figma может интегрироваться с другими инструментами и плагинами, такими как Zeplin, Sketch, Jira и многими другими, что облегчает рабочий процесс и обмен информацией между различными инструментами.

Благодаря своим возможностям и функциональности, Figma является популярным выбором для проектирования и коллаборации в процессе разработки мобильных приложений. Он помогает дизайнерам и разработчикам эффективно сотрудничать, создавать качественные интерфейсы и прототипы, а также обеспечивать единообразие и согласованность внешнего вида приложений.

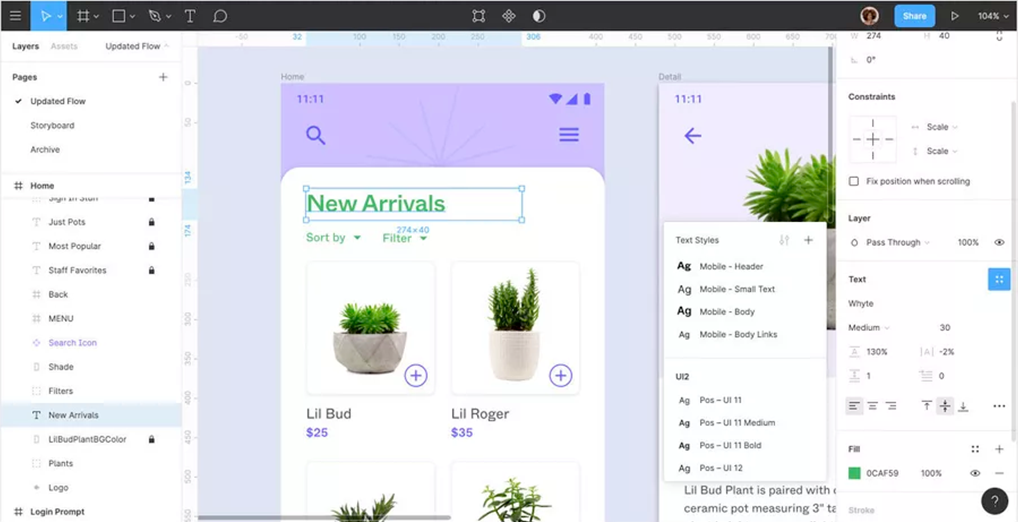


Рисунок 2.5 – Интерфейс Figma

### **2.4.2 Рисование эскизов. Создание Wireframes. Дизайн**

После исследования рынка и формирования требований проекта следует этап проектирования UI (англ. User Interface – Пользовательский Интерфейс). На этом этапе стоит задача разработки интерфейса приложения, а сам процесс можно разделить на несколько частей [9]:

1. Рисование эскизов
2. Создание Wireframes (Прототипов)
3. Дизайн

Рисование эскизов

На этом этапе схематично изображаются наброски будущего приложения. Главная задача – понять какой структурой будет обладать приложение. Какие и где будут размещены элементы, какие разделы будут и каким образом будет осуществляться переход между ними. Как будет реагировать интерфейс при том или ином действии пользователя.



Рисунок 2.6 – Эскизы приложения

Создание Wireframes

Wireframes по своей сути являются продолжением предыдущего этапа только в цифровом пространстве и большим вниманием к деталям. Wireframes — это организация полного функционала конечного продукта, в виде структуры с представлением элементов интерфейса и навигации, их взаимодействия друг с другом. Как правило, в wireframes не используется цвет и любые другие графические элементы оформления, так как основное внимание уделяется функциональности, поведению и содержанию. Wireframes фокусируется на том, как экраны взаимодействуют между собой, а не на то, как они выглядят.

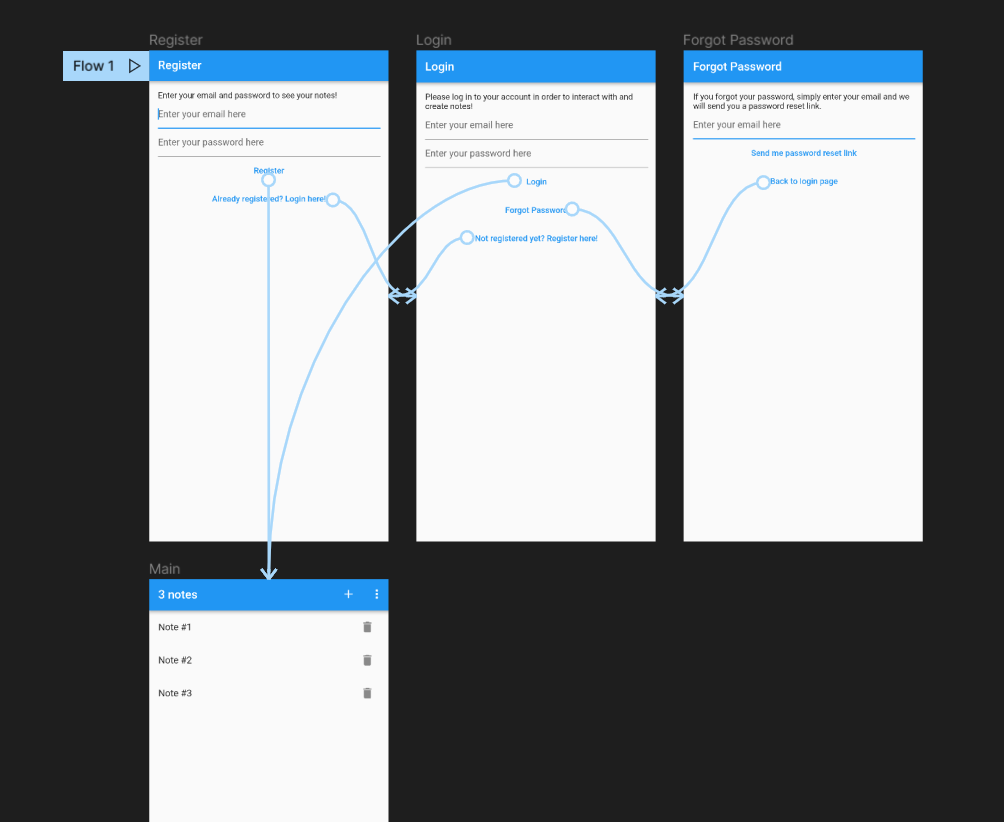


Рисунок 2.7 – Wireframes приложения

Дизайн

На данном этапе созданным wireframes придается финальный облик. Дизайн приложения — это визуальный вид программы. Хороший дизайн одновременно эстетичный и удобный для пользователя. Дизайн — это не только про красивую картинку, это в первую очередь про простоту использования. На прошлом шаге были созданы высокоточные wireframes, что позволило, практически не внося изменений, придать приложению симпатичный визуальный облик.

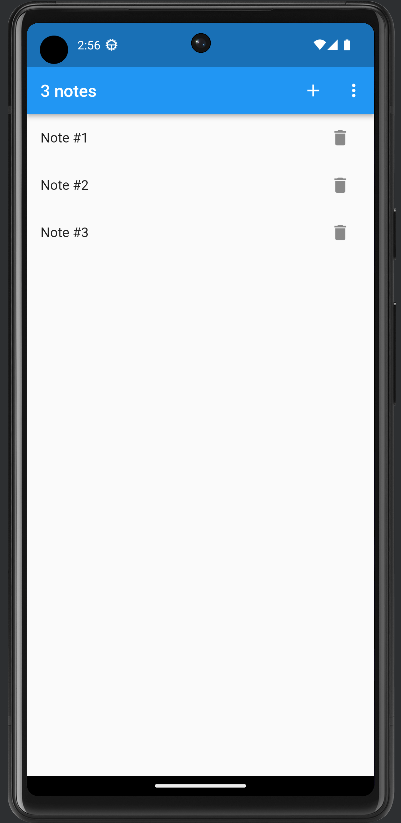


Рисунок 2.8 – Дизайн приложения

Поскольку данное приложение простое в реализации интерфейса, некоторые части процесса реализации можно совмещать. Поэтому финальный дизайн приложения практически не отличается от эскиза.

# ГЛАВА 3. РАЗРАБОТКА КРОССПЛАТФОРМЕННОГО МОБИЛЬНОГО ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ FLUTTER

**3.1 Разработка логики приложения**

На этом этапе разработки кроссплатформенного мобильного приложения происходит создание основной логики приложения. Сперва определяются основные сущности и объекты, с которыми будет работать приложение. Определяется структура этих классов и их взаимосвязи.

Затем разрабатывается функциональность приложения. Создаются методы и функции для реализации поставленного функционала.

При разработке логики приложения также уделяется внимание обработке событий и взаимодействию с пользователем. Определяется, как приложение будет реагировать на пользовательский ввод, например, при нажатии кнопки сохранения заметки или ее удаления. Создаются обработчики событий, которые будут вызываться при возникновении определенных действий пользователя.

### **3.1.1 Основные классы**

Класс AuthUser представляет пользователя и отвечает за аутентификацию и управление его учетными данными. В приложении класс AuthUser представим таким образом:

@immutable

class AuthUser {

final String id;

final String email;

final bool isEmailVerified;

const AuthUser({

required this.id,

required this.email,

required this.isEmailVerified,

});

factory AuthUser.fromFirebase(User user) => AuthUser(

id: user.uid,

email: user.email!,

isEmailVerified: user.emailVerified,

);

}  
  
Класс CloudNote представляет заметку, которая хранится и синхронизируется в облаке. В приложении класс CloudNote представим так:

@immutable

class CloudNote {

final String documentId;

final String ownerUserId;

final String text;

const CloudNote({

required this.documentId,

required this.ownerUserId,

required this.text,

});

CloudNote.fromSnapshot(QueryDocumentSnapshot<Map<String, dynamic>> snapshot)

: documentId = snapshot.id,

ownerUserId = snapshot.data()[ownerUserIdFieldName],

text = snapshot.data()[textFieldName] as String;

}

### **3.1.2 Пользовательский интерфейс. Реализация**

В пункте 2.4.2 был описан прототип нашего приложения. Поскольку дизайн приложения простой, будет использоваться базовые правила дизайна Google: Material Design 3[4]. Тема Flutter приложения, иконки, кнопки и базовые виджеты взяты из библиотеки виджетов Flutter.

Главный экран приложения состоит из AppBar в котором пользователь может видеть количество заметок, которые он сделал. А также кнопка “+”(добавить заметку и кнопка меню, где пользователь может выйти из своего аккаунта. В приложении А находится исходный код главного экрана.

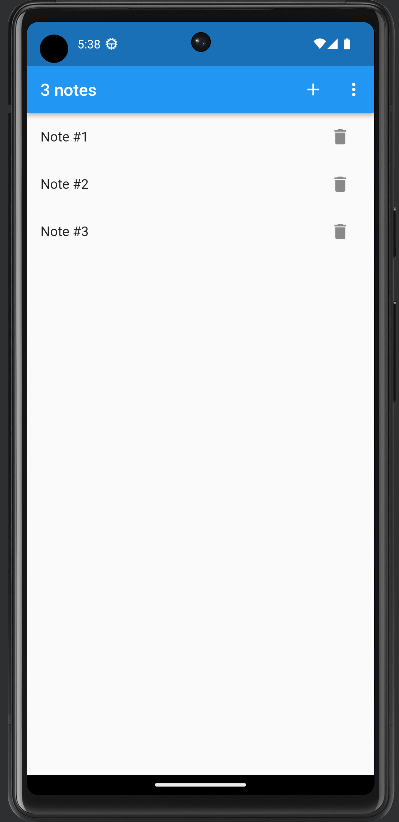


Рисунок 3.1 – Главный экран приложения

Экран входа в приложение состоит из двух текстовых полей, где пользователь вводит свой логин (почту) и пароль. После ввода своих данных пользователь нажимает кнопку “Войти”. Также есть кнопка перехода на экран, где пользователь может восстановить пароль, и кнопка “Регистрации”, если пользователь еще не зарегистрирован в приложении. В приложении Б находится исходный код экрана входа.

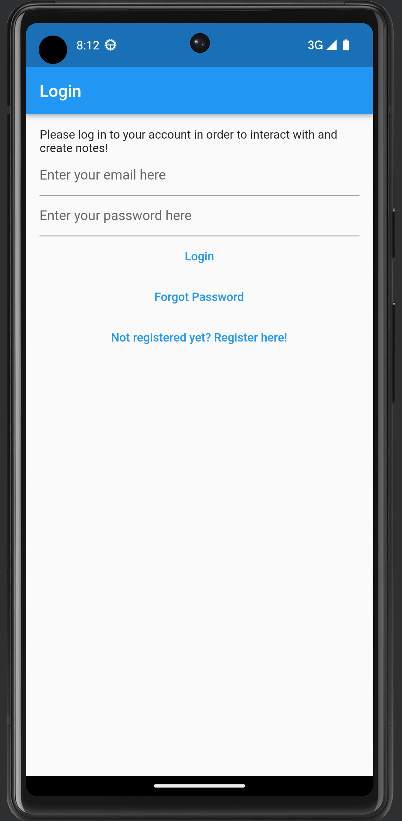


Рисунок 3.2 – Экран входа в приложение

Экран регистрации в приложение состоит, как и экран входа, из двух текстовых полей где пользователь вводит свой логин (почту) и пароль. После ввода своих данных пользователь нажимает кнопку “Регистрация”. Также внизу есть кнопка для возврата на экран входа. В приложении В находится исходный код экрана регистрации.

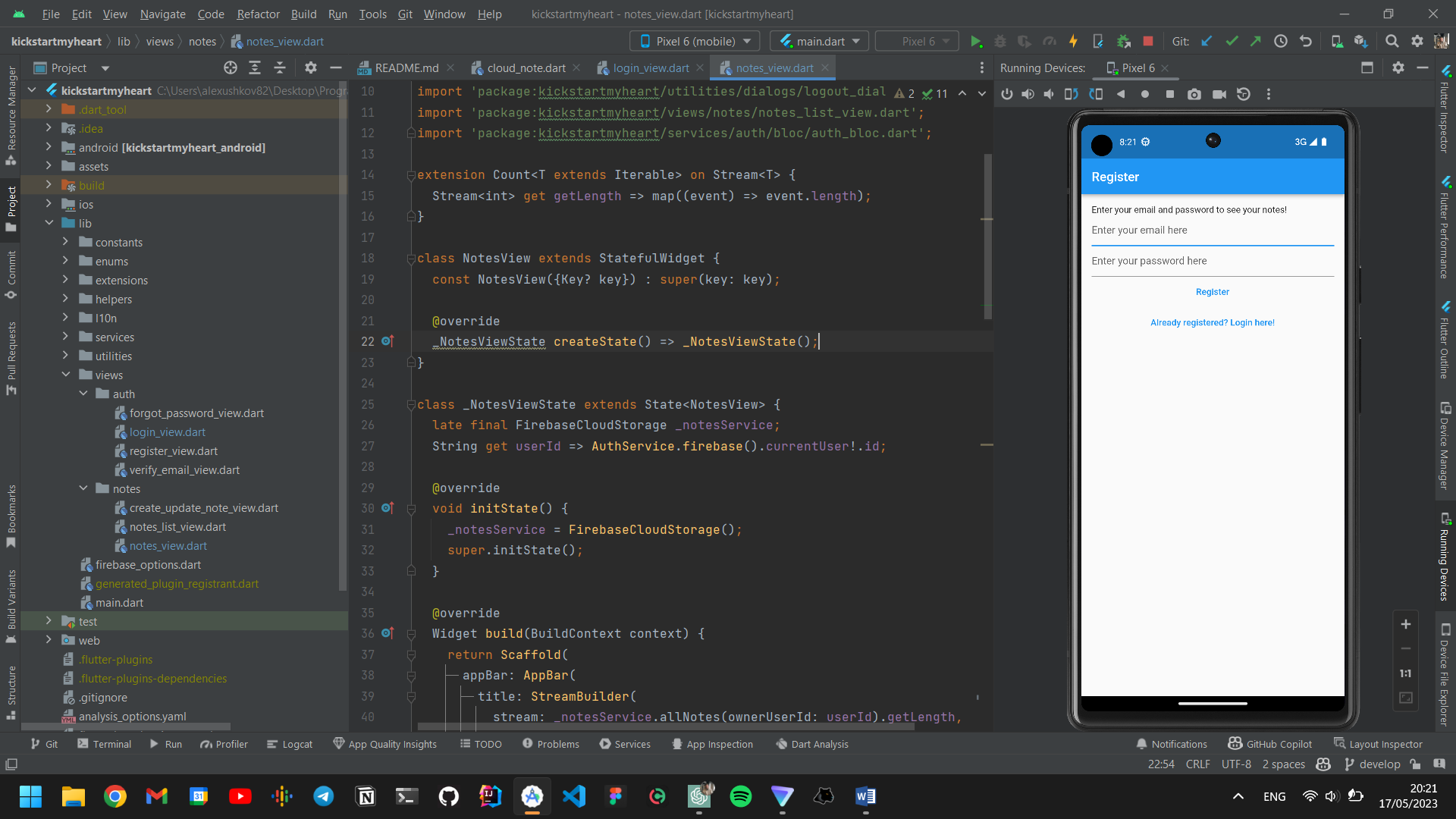


Рисунок 3.3 – Экран регистрации в приложение

# ЗАКЛЮЧЕНИЕ

Разработка мобильных приложений с использованием языка Dart и библиотеки Flutter представляет собой эффективный и мощный подход к созданию кроссплатформенных приложений. Dart обладает простым синтаксисом, сильной типизацией и высокой производительностью, что делает его отличным выбором для разработки мобильных приложений. Библиотека Flutter, в свою очередь, предоставляет множество готовых компонентов и возможностей, позволяющих создавать красивые и интерактивные пользовательские интерфейсы.

В процессе разработки приложения, был проведен анализ требований, выбраны необходимые компоненты и функциональность, разработаны интерфейс и логика приложения. Процесс разработки был облегчен благодаря широким возможностям Flutter, таким как горячая перезагрузка (hot reload), которая позволяет мгновенно видеть изменения в коде без необходимости перезапуска приложения.

В результате разработки было создано мобильное приложение для заметок, которое обладает удобным и простым интерфейсом, функциональностью добавления, редактирования и удаления заметок, а также сохранения заметок в базу данных. Приложение создано с учетом современных требований пользователей и позволяет удобно использовать его на мобильных устройствах.

Одним из основных преимуществ разработки мобильного приложения с использованием Dart и Flutter является его кроссплатформенность. Приложение может быть запущено как на операционной системе Android, так и на iOS, что позволяет охватить широкую аудиторию пользователей и снизить затраты на разработку и поддержку.

В целом, разработка мобильного приложения с использованием языка Dart и библиотеки Flutter представляет собой перспективный и эффективный подход к созданию кроссплатформенных приложений. Dart и Flutter предоставляют разработчикам мощные инструменты и возможности, чтобы создавать качественные и инновационные приложения, способные удовлетворить требования современных пользователей.

В дальнейшем планируется добавить новый функционал (добавление папок для заметок, изменение темы приложения, выбор аккаунта, беспарольная авторизация и др.).

Для реализации приложения использовалась среда Android Studio.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. "Docker: Up & Running" авторства Sean P. Kane и Karl Matthias. \2."Kubernetes: Up and Running" авторства Kelsey Hightower, Brendan Burns и Joe Beda. Эта книга поможет вам понять основы Kubernetes и его применение в контексте разработки.

3."Containerization with Docker" авторства V. Sriram, Sreejesh Suresh, и Rajdeep Dua.

4.Статья "Docker: Lightweight Linux Containers for Consistent Development and Deployment" (<https://www.usenix.org/legacy/event/atc14/techsessions/ren.html>).

5.Статья "Kubernetes: An Open-Source Platform for Automating Deployment, Scaling, and Operations of Application Containers" (<https://www.usenix.org/system/files/conference/nsdi15/nsdi15-paper-burns.pdf>).

# ПРИЛОЖЕНИЕ А

Реализация интерфейса главного экрана (notes\_view.dart):

import 'package:flutter/material.dart';

import 'package:flutter\_bloc/flutter\_bloc.dart' show ReadContext;

import 'package:kickstartmyheart/constants/routes.dart';

import 'package:kickstartmyheart/enums/menu\_action.dart';

import 'package:kickstartmyheart/extensions/buildcontext/loc.dart';

import 'package:kickstartmyheart/services/auth/auth\_service.dart';

import 'package:kickstartmyheart/services/auth/bloc/auth\_event.dart';

import 'package:kickstartmyheart/services/cloud/cloud\_note.dart';

import 'package:kickstartmyheart/services/cloud/firebase\_cloud\_storage.dart';

import 'package:kickstartmyheart/utilities/dialogs/logout\_dialog.dart';

import 'package:kickstartmyheart/views/notes/notes\_list\_view.dart';

import 'package:kickstartmyheart/services/auth/bloc/auth\_bloc.dart';

extension Count<T extends Iterable> on Stream<T> {

Stream<int> get getLength => map((event) => event.length);

}

class NotesView extends StatefulWidget {

const NotesView({Key? key}) : super(key: key);

@override

\_NotesViewState createState() => \_NotesViewState();

}

class \_NotesViewState extends State<NotesView> {

late final FirebaseCloudStorage \_notesService;

String get userId => AuthService.firebase().currentUser!.id;

@override

void initState() {

\_notesService = FirebaseCloudStorage();

super.initState();

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: StreamBuilder(

stream: \_notesService.allNotes(ownerUserId: userId).getLength,

builder: (context, AsyncSnapshot<int> snapshot) {

if (snapshot.hasData) {

final noteCount = snapshot.data ?? 0;

final text = context.loc.notes\_title(noteCount);

return Text(text);

} else {

return const Text('');

}

},

),

actions: [

IconButton(

onPressed: () {

Navigator.of(context).pushNamed(createOrUpdateNoteRoute);

},

icon: const Icon(Icons.add),

),

PopupMenuButton<MenuAction>(

onSelected: (value) async {

switch (value) {

case MenuAction.logout:

final shouldLogout = await showLogOutDialog(context);

if (shouldLogout) {

context.read<AuthBloc>().add(

const AuthEventLogOut(),

);

}

}

},

itemBuilder: (context) {

return [

PopupMenuItem<MenuAction>(

value: MenuAction.logout,

child: Text(context.loc.logout\_button),

),

];

},

)

],

),

body: StreamBuilder(

stream: \_notesService.allNotes(ownerUserId: userId),

builder: (context, snapshot) {

switch (snapshot.connectionState) {

case ConnectionState.waiting:

case ConnectionState.active:

if (snapshot.hasData) {

final allNotes = snapshot.data as Iterable<CloudNote>;

return NotesListView(

notes: allNotes,

onDeleteNote: (note) async {

await \_notesService.deleteNote(documentId: note.documentId);

},

onTap: (note) async {

Navigator.of(context).pushNamed(

createOrUpdateNoteRoute,

arguments: note,

);

},

);

} else {

return const CircularProgressIndicator();

}

default:

return const CircularProgressIndicator();

}

},

),

);

}

}

# ПРИЛОЖЕНИЕ Б

Реализация интерфейса экрана входа (login\_view.dart):

import 'package:flutter/material.dart';

import 'package:flutter\_bloc/flutter\_bloc.dart';

import 'package:kickstartmyheart/extensions/buildcontext/loc.dart';

import 'package:kickstartmyheart/services/auth/auth\_exceptions.dart';

import 'package:kickstartmyheart/services/auth/bloc/auth\_event.dart';

import 'package:kickstartmyheart/utilities/dialogs/error\_dialog.dart';

import 'package:kickstartmyheart/services/auth/bloc/auth\_bloc.dart';

import 'package:kickstartmyheart/services/auth/bloc/auth\_state.dart';

class LoginView extends StatefulWidget {

const LoginView({Key? key}) : super(key: key);

@override

State<LoginView> createState() => \_LoginViewState();

}

class \_LoginViewState extends State<LoginView> {

late final TextEditingController \_email;

late final TextEditingController \_password;

@override

void initState() {

\_email = TextEditingController();

\_password = TextEditingController();

super.initState();

}

@override

void dispose() {

\_email.dispose();

\_password.dispose();

super.dispose();

}

@override

Widget build(BuildContext context) {

return BlocListener<AuthBloc, AuthState>(

listener: (context, state) async {

if (state is AuthStateLoggedOut) {

if (state.exception is UserNotFoundAuthException) {

await showErrorDialog(

context,

context.loc.login\_error\_cannot\_find\_user,

);

} else if (state.exception is WrongPasswordAuthException) {

await showErrorDialog(

context,

context.loc.login\_error\_wrong\_credentials,

);

} else if (state.exception is GenericAuthException) {

await showErrorDialog(

context,

context.loc.login\_error\_auth\_error,

);

}

}

},

child: Scaffold(

appBar: AppBar(

title: Text(context.loc.login),

),

body: Padding(

padding: const EdgeInsets.all(16.0),

child: SingleChildScrollView(

child: Column(

children: [

Text(context.loc.login\_view\_prompt),

TextField(

controller: \_email,

enableSuggestions: false,

autocorrect: false,

keyboardType: TextInputType.emailAddress,

decoration: InputDecoration(

hintText: context.loc.email\_text\_field\_placeholder,

),

),

TextField(

controller: \_password,

obscureText: true,

enableSuggestions: false,

autocorrect: false,

decoration: InputDecoration(

hintText: context.loc.password\_text\_field\_placeholder,

),

),

TextButton(

onPressed: () async {

final email = \_email.text;

final password = \_password.text;

context.read<AuthBloc>().add(

AuthEventLogIn(email, password),

);

},

child: Text(context.loc.login),

),

TextButton(

onPressed: () {

context.read<AuthBloc>().add(const AuthEventForgotPassword());

},

child: Text(context.loc.forgot\_password),

),

TextButton(

onPressed: () {

context.read<AuthBloc>().add(const AuthEventShouldRegister());

},

child: Text(context.loc.login\_view\_not\_registered\_yet),

)

],

),

),

),

),

);

}

}

# ПРИЛОЖЕНИЕ В

Реализация интерфейса экрана регистрации (register\_view.dart):

import 'package:flutter/material.dart';

import 'package:flutter\_bloc/flutter\_bloc.dart';

import 'package:kickstartmyheart/extensions/buildcontext/loc.dart';

import 'package:kickstartmyheart/services/auth/auth\_exceptions.dart';

import 'package:kickstartmyheart/services/auth/bloc/auth\_event.dart';

import 'package:kickstartmyheart/utilities/dialogs/error\_dialog.dart';

import 'package:kickstartmyheart/services/auth/bloc/auth\_bloc.dart';

import 'package:kickstartmyheart/services/auth/bloc/auth\_state.dart';

class RegisterView extends StatefulWidget {

const RegisterView({Key? key}) : super(key: key);

@override

\_RegisterViewState createState() => \_RegisterViewState();

}

class \_RegisterViewState extends State<RegisterView> {

late final TextEditingController \_email;

late final TextEditingController \_password;

@override

void initState() {

\_email = TextEditingController();

\_password = TextEditingController();

super.initState();

}

@override

void dispose() {

\_email.dispose();

\_password.dispose();

super.dispose();

}

@override

Widget build(BuildContext context) {

return BlocListener<AuthBloc, AuthState>(

listener: (context, state) async {

if (state is AuthStateRegistering) {

if (state.exception is WeakPasswordAuthException) {

await showErrorDialog(

context,

context.loc.register\_error\_weak\_password,

);

} else if (state.exception is EmailAlreadyInUseAuthException) {

await showErrorDialog(

context,

context.loc.register\_error\_email\_already\_in\_use,

);

} else if (state.exception is InvalidEmailAuthException) {

await showErrorDialog(

context,

context.loc.register\_error\_invalid\_email,

);

} else if (state.exception is GenericAuthException) {

await showErrorDialog(

context,

context.loc.register\_error\_generic,

);

}

}

},

child: Scaffold(

appBar: AppBar(

title: Text(context.loc.register),

),

body: Padding(

padding: const EdgeInsets.all(16.0),

child: SingleChildScrollView(

child: Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

Text(context.loc.register\_view\_prompt),

TextField(

controller: \_email,

enableSuggestions: false,

autocorrect: false,

autofocus: true,

keyboardType: TextInputType.emailAddress,

decoration: InputDecoration(

hintText: context.loc.email\_text\_field\_placeholder,

),

),

TextField(

controller: \_password,

obscureText: true,

enableSuggestions: false,

autocorrect: false,

decoration: InputDecoration(

hintText: context.loc.password\_text\_field\_placeholder,

),

),

Center(

child: Column(

children: [

TextButton(

onPressed: () async {

final email = \_email.text;

final password = \_password.text;

context.read<AuthBloc>().add(AuthEventRegister(

email,

password,

));

},

child: Text(context.loc.register),

),

TextButton(

onPressed: () {

context.read<AuthBloc>().add(const AuthEventLogOut());

},

child: Text(

context.loc.register\_view\_already\_registered,

),

),

],

),

)

],

),

),

),

),

);

}

}