Государственное бюджетное профессиональное образовательное учреждение

«Нижегородский Губернский колледж»

**Итоговый отчёт**

По разработке Информационной системы: «Code Blitz»

Студент\_\_\_\_\_\_\_\_\_Фадеев А. В. 01.10.2024

Преподаватель\_\_\_\_\_\_\_\_\_\_\_\_Мамшева Ю. С. 01.10.2024

Нижний Новгород

2024

ОГЛАВЛЕНИЕ

[ОСНОВНАЯ ЧАСТЬ 4](#_Toc184058695)

[1. Разработка технического задания 4](#_Toc184058696)

[1.1. Выбор паттерна проектирования 4](#_Toc184058697)

[1.2. Требования к серверной части 4](#_Toc184058698)

[1.2.1. Функциональные требования 4](#_Toc184058699)

[1.2.2. Нефункциональные требования 5](#_Toc184058700)

[1.3. Требования к мобильному приложению 5](#_Toc184058701)

[1.3.1. Функциональные требования 5](#_Toc184058702)

[1.3.2. Нефункциональные требования 6](#_Toc184058703)

[1.4. Выбор ПО для реализации 6](#_Toc184058704)

[1.4.1. Мобильное приложение 6](#_Toc184058705)

[1.4.2. Серверная часть 6](#_Toc184058706)

[2. Разработка макета приложения 7](#_Toc184058707)

[2.1. Выбор цветовой палитры 7](#_Toc184058708)

[2.1. Выбор шрифта для заголовков и основного текста 8](#_Toc184058709)

[2.2. Разработка макета каждого экрана 8](#_Toc184058710)

[3. Разработка серверной части приложения 9](#_Toc184058711)

[3.1. Проектирование серверной части 9](#_Toc184058712)

[3.2. Разработка серверной части 17](#_Toc184058713)

[4. Тестирование серверной части 18](#_Toc184058714)

[4.1. Разработка документации для тестирования сервера приложения 18](#_Toc184058715)

[4.2. Автоматизированное тестирование серверной части 19](#_Toc184058716)

[5. Разработка мобильного приложения 24](#_Toc184058717)

[5.1. Подключение шрифтов и настройка ранее выбранных цветов 24](#_Toc184058718)

[5.2. Верстка макета 24](#_Toc184058719)

[5.3. Подключение библиотек и интернета 30](#_Toc184058720)

[5.4. Подключение к серверной части приложения 30](#_Toc184058721)

[5.5. Разработка методов 31](#_Toc184058722)

[6. Тестирование мобильного приложения 31](#_Toc184058723)

[6.1. Разработка тестовой документации 31](#_Toc184058724)

[6.2. Автоматизированное тестирование 31](#_Toc184058725)

[6.2.1. UI-тестирование 31](#_Toc184058726)

[6.2.2. Unit-тестирование 36](#_Toc184058727)

[ЗАКЛЮЧЕНИЕ 40](#_Toc184058728)

ВВЕДЕНИЕ

В современном мире программирование является одним из самых востребованных и сложных направлений профессиональной деятельности. Постоянная практика необходима для поддержания высокого уровня навыков, особенно в области алгоритмики, являющейся ключевым элементом разработки эффективных программных модулей. Однако, найти время и мотивацию для регулярных занятий бывает непросто.

Мобильное приложение, предоставляющее возможность обучаться написанию алгоритмов в игровой форме в любой момент времени, может стать решением этой проблемы. Доступность мобильных устройств в современном мире позволяет практиковаться в решении алгоритмических задач в любом месте и в любое время, делая процесс обучения более удобным и доступным.

Цель разработки – создание мобильного приложения, позволяющего пользователям решать ежедневные задачи по программированию в соревновательной форме, сравнивая свои результаты и решения с другими пользователями.

Для выполнения поставленной цели необходимо выполнение следующих задач:

* Разработать техническое задание;
* Разработать макет приложения;
* Разработать серверную часть приложения;
* Провести тестирование серверной части приложения;
* Разработать мобильное приложение;
* Провести тестирование мобильного приложения;
* Подготовить отчёт и презентацию о разработанном продукте;
* Презентовать разработанный продукт.

# ОСНОВНАЯ ЧАСТЬ

# Разработка технического задания

# Выбор паттерна проектирования

Выбранный паттерн проектирования – MVVM (Model-View-ViewModel)

Обоснование выбора:

* Паттерн обеспечивает чёткое разделение интерфейса, бизнес логики и хранения данных, тем самым предавая упорядоченность структуре проекта и упрощая разработку;
* Разделение ответственности компонентов системы упрощает их тестирование по отдельности;
* Разделение компонентов делает приложение более масштабируемым и позволяет легче вносить в него изменения.

# Требования к серверной части

## Функциональные требования

Перечень функциональных требований к серверной части:

* Реализация на основе Supabase;
* Аутентификация пользователей: создание учётных записей, авторизация;
* Хранений задач: добавление задач в БД, хранение задач в БД, удаление задач из БД, хранение даты задачи (ввиду формата ежедневных задач);
* Хранение решений: добавление решений в БД, хранение решений в БД, хранение кода решения в БД, связность пользователя с решением в БД, хранение времени решения, связность решения и задачи, хранение данных о выбранном языке программирования, удаление решений;
* Наличие API для добавления и запроса данных из мобильного приложения (реализуется Supabase);

## Нефункциональные требования

Перечень нефункциональных требований к серверной части:

* Безопасность данных пользователя при аутентификации;
* Возможность расширения системы для обработки большего количества пользователей и задач.

# Требования к мобильному приложению

## Функциональные требования

Перечень функциональных требований к мобильному приложению:

* Регистрация и авторизация: создание учетной записи, вход в систему, выход из системы;
* Просмотр задач: получение ежедневных задач, просмотр описания и условий задачи;
* Реализация задач: выбор языка программирования, ввод кода, подсветка кода, запуск таймера, отправка решения на проверку;
* Просмотр результатов: просмотр таблицы лидеров, сравнение своего времени с результатами других пользователей;
* Просмотр решений: просмотр решений других пользователей, возможность изучения их кода.

## Нефункциональные требования

Перечень нефункциональных требований к мобильному приложению:

* Пользовательский интерфейс: простота использования, интуитивно понятный дизайн, адаптируемый для различных размеров экрана;
* Быстрая загрузка приложения, быстрое выполнение операций.

# Выбор ПО для реализации

## Мобильное приложение

Для реализации мобильного приложения было выбрано следующее ПО:

1. Среда разработки Android Studio:

* Официальная IDE для Android: Android Studio предоставляет богатый набор инструментов для разработки приложений под Android, включая симуляторы, отладчик, средства профилирования и многое другое;
* Большая экосистема: Android Studio поддерживается Google и имеет большую экосистему плагинов и документации, что облегчает разработку.

1. Figma:

* Figma доступна в веб-браузере, что позволяет работать с ней на любом устройстве без необходимости установки;
* Figma предлагает широкий набор инструментов для создания высококачественных дизайнов, включая векторную графику, прототипирование, стили, компоненты и анимации.

## Серверная часть

Для реализации серверной части был выбран сервис Supabase:

* Сервис предоставляет полноценную платформу, включая базу данных (PostgreSQL), аутентификацию, хранилище файлов и API;
* Supabase обладает простым в использовании интерфейсом и предлагает богатую документацию, что сокращает время на настройку и разработку серверной части;
* Supabase является масштабируемым решением, что позволяет приложению обрабатывать большое количество запросов и данных;
* Supabase имеет бесплатный план, что делает его доступным для небольших и учебных проектов.

## Разработка макета приложения

## Выбор цветовой палитры

В рамках разработки макета приложения, был выполнен выбор основной цветовой палитры, используемой приложением:

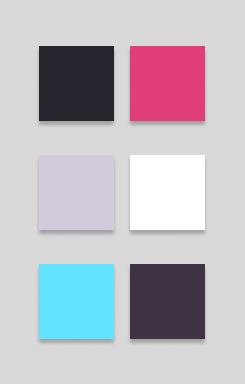


Рисунок 1 – Основная палитра приложения

Также был добавлен ряд дополнительных палитр, которые пользователи смогут устанавливать во время своей работы:



Рисунок 2 – Дополнительные палитры

## Выбор шрифта для заголовков и основного текста

Были выбраны следующие шрифты:

* Основной текст – JetBrains Mono
* Заголовки – Intro

## Разработка макета каждого экрана

Был разработан макет для каждого экрана приложения CodeBlitZ

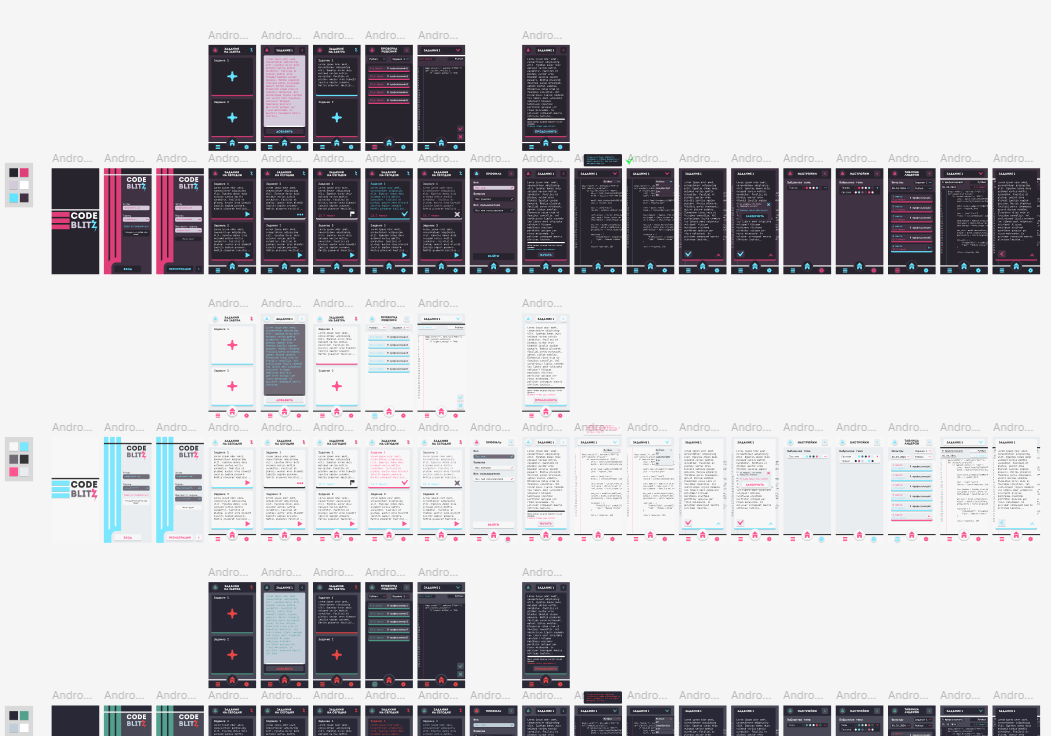


Рисунок 3 – Разработанный макет

Ссылка на макет - <https://www.figma.com/design/rTUUWG5ZbsedkDY6VORlQI/CodeBlitZ?node-id=0-1&t=pJLvMVX7pgUbVamx-1>

## Разработка серверной части приложения

## Проектирование серверной части

В качестве платформы реализации серверной части был выбран Supabase. Supabase предоставляет все необходимые инструменты для создания базы данных, аутентификации, хранилища объектов, функций и т.д.

Для серверной части была создана ER-диаграмма:

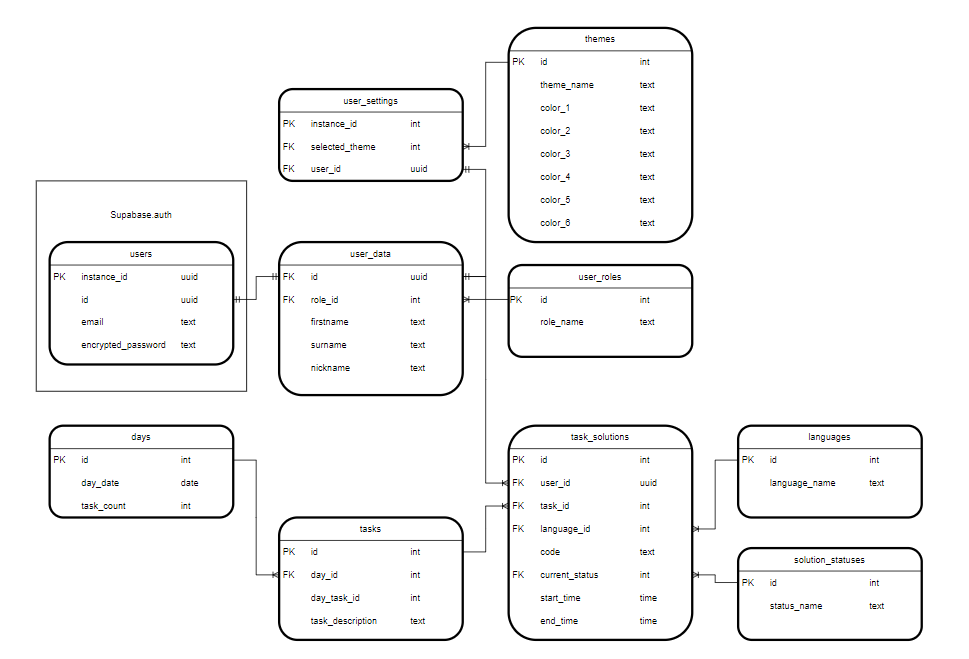


Рисунок 4 – ER-диаграмма

Всего в серверной части используется 10 таблиц:

Таблица 1 – Описание таблиц

| Таблица | Описание |
| --- | --- |
| users | Содержит информацию о пользователях (часть Supabase.auth) |
| user\_data | Содержит расширенные данные о пользователях, не относящиеся к аутентификации |
| user\_roles | Содержит информацию о ролях пользователей |
| user\_settings | Содержит информацию о настройках пользователя |
| days | Содержит информацию о днях с задачами |
| tasks | Содержит информацию о задачах |
| task\_solutions | Содержит информацию о решениях задач пользователей |
| languages | Содержит информацию о доступных языках программирования |
| solution\_statuses | Содержит информацию о статусах решения задачи |
| themes | Содержит информацию о доступных темах |

Далее описаны все поля таблиц:

Таблица 2 – Описание полей

| Таблица | Поле | Назначение | Тип данных | Ограничения |
| --- | --- | --- | --- | --- |
| users | instance\_id | Идентификатор записи о пользователе | uuid | primary key |
|  | id | Идентификатор пользователя | uuid | unique |
|  | email | Электронный адрес пользователя | text | not null |
|  | encrypted\_password | Зашифрованный пароль пользователя | text | not null |
| user\_data | id | Идентификатор пользователя | uuid | primary key |
|  | firstname | Имя пользователя | text | not null, length < 30 |
|  | surname | Фамилия пользователя | text | not null, length < 30 |
|  | role\_id | Идентификатор роли пользователя | integer | foreign key |
|  | nickname | Никнейм пользователя | text | length < 30 |
| user\_roles | id | Идентификатор роли | integer | primary key |
|  | role\_name | Название роли | text | not null, unique |
| user\_settings | id | Идентификатор настройки | integer | primary key |
|  | selected\_theme | Выбранная пользователем тема | integer | foreign key,not null |
|  | user\_id | Идентификатор пользователя | uuid | foreign key, not null, unique |
| days | id | Идентификатор дня | integer | primary key |
|  | day\_date | Дата дня | date | not null, unique |
|  | task\_count | Количество задач в дне | integer | not null, 0 <= task\_count <= 2 |
| tasks | id | Идентификатор задачи | integer | primary key, not null |
|  | day\_id | Идентификатор дня, к которому относится задача | integer | foreign key, not null |
|  | day\_task\_id | Номер задачи в рамках дня | integer | not null, 1 <= day\_task\_id <= 2 |
|  | task\_description | Описание задачи | text | not null, unique |
| task\_solutions | id | Идентификатор решения задачи | integer | primary key |
|  | user\_id | Идентификатор пользователя, решившего задачу | uuid | foreign key, not null |
|  | task\_id | Идентификатор задачи | integer | foreign key, not null |
|  | language\_id | Идентификатор языка, на котором решена задача | integer | foreign key, not null |
|  | code | Решение задачи | text | not null |
|  | current\_status | Текущий статус решения задачи | text | not null |
|  | start\_time | Время начала решения задачи | time | not null |
|  | end\_time | Время окончания решения задачи | time | not null |
| languages | id | Идентификатор языка | integer | primary key |
|  | language\_name | Название языка | text | not null, unique |
| solution\_statuses | id | Идентификатор статуса решения задачи | integer | primary key |
|  | status\_name | Название статуса решения задачи | text | not null, unique |
| themes | id | Идентификатор темы | integer | primary key |
|  | theme\_name | Название темы | text | not null |
|  | color\_1 | Цвет 1 для темы | text |  |
|  | color\_2 | Цвет 2 для темы | text |  |
|  | color\_3 | Цвет 3 для темы | text |  |
|  | color\_4 | Цвет 4 для темы | text |  |
|  | color\_5 | Цвет 5 для темы | text |  |
|  | color\_6 | Цвет 6 для темы | text |  |

Далее описаны все связи между таблицами:

Таблица 3 – Связи

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 1 | Таблица 2 | Тип связи | Описание |
| users | user\_data | 1-1 | Каждый пользователь имеет один набор данных |
| user\_data | user\_settings | 1-1 | Каждый пользователь имеет одну настройку |
| user\_data | user\_roles | м-1 | Пользователь может иметь только одну роль |
| user\_data | task\_solutions | 1-м | Пользователь может иметь множество решений |
| days | tasks | 1-м | День может содержать несколько задач |
| tasks | task\_solutions | 1-м | Задача может иметь много решений |
| user\_settings | themes | м-1 | Пользователь может выбрать одну тему |
| task\_solutions | languages | м-1 | На каждом языке программирования может быть написано несколько решений |
| task\_solutions | solution\_statuses | м-1 | Каждое решение задачи может иметь один из статусов |

Также была реализована функция Realtime для таблиц task\_solutions и user\_settings, однако в дальнейшем не была использована ввиду ограничений данного функции.

Всего для серверной части было разработано 3 триггера:

Таблица 4 – Триггеры

|  |  |
| --- | --- |
| Название | Описание |
| check\_task\_count\_trigger | Запрещает добавление задание на день, если для этого дня уже есть 2 задания |
| increment\_task\_count\_trigger | Увеличивает счетчик заданий на день после добавлении задания |
| set\_day\_task\_id\_trigger | Устанавливает day\_task\_id задачи в рамках дня (1 или 2) |

Листинг 1 – Триггер check\_task\_count\_trigger

|  |
| --- |
| *-- Триггер для проверки количества задач на день*  **CREATE** **OR** **REPLACE** **FUNCTION** check\_task\_count() **RETURNS** **TRIGGER** **AS** $$  **BEGIN**  **IF** (**SELECT** **COUNT**(\*) **FROM** tasks **WHERE** day\_id = NEW.day\_id) >= 2 **THEN**  **RAISE** **EXCEPTION** 'Нельзя добавить больше двух задач на этот день.';  **END** **IF**;  RETURN NEW;  **END**;  $$ LANGUAGE plpgsql;  **CREATE** **TRIGGER** check\_task\_count\_trigger  **BEFORE** **INSERT** **ON** tasks  **FOR** **EACH** **ROW**  **EXECUTE** **PROCEDURE** check\_task\_count(); |

Листинг 2 – Триггер increment\_task\_count\_trigger

|  |
| --- |
| *-- Триггер для увеличения количества задач на день*  **CREATE** **OR** **REPLACE** **FUNCTION** increment\_task\_count() **RETURNS** **TRIGGER** **AS** $$  **BEGIN**  **UPDATE** **days** **SET** task\_count = task\_count + 1 **WHERE** **id** = NEW.day\_id;  RETURN NEW;  **END**;  $$ LANGUAGE plpgsql;  **CREATE** **TRIGGER** increment\_task\_count\_trigger  **AFTER** **INSERT** **ON** tasks  **FOR** **EACH** **ROW**  **EXECUTE** **PROCEDURE** increment\_task\_count(); |

Листинг 3 – Триггер set\_day\_task\_id\_trigger

|  |
| --- |
| *-- Триггер для установки day\_task\_id*  **CREATE** **OR** **REPLACE** **FUNCTION** set\_day\_task\_id() **RETURNS** **TRIGGER** **AS** $$  **BEGIN**  **IF** (**SELECT** **COUNT**(\*) **FROM** tasks **WHERE** day\_id = NEW.day\_id) = 0 **THEN**  NEW.day\_task\_id := 1;  ELSE  NEW.day\_task\_id := 2;  **END** **IF**;  RETURN NEW;  **END**;  $$ LANGUAGE plpgsql;  **CREATE** **TRIGGER** set\_day\_task\_id\_trigger  **BEFORE** **INSERT** **ON** tasks  **FOR** **EACH** **ROW**  **EXECUTE** **PROCEDURE** set\_day\_task\_id(); |

## Разработка серверной части

Серверная часть была разработана в сервисе Supabase. Были реализованы все спроектированные элементы.

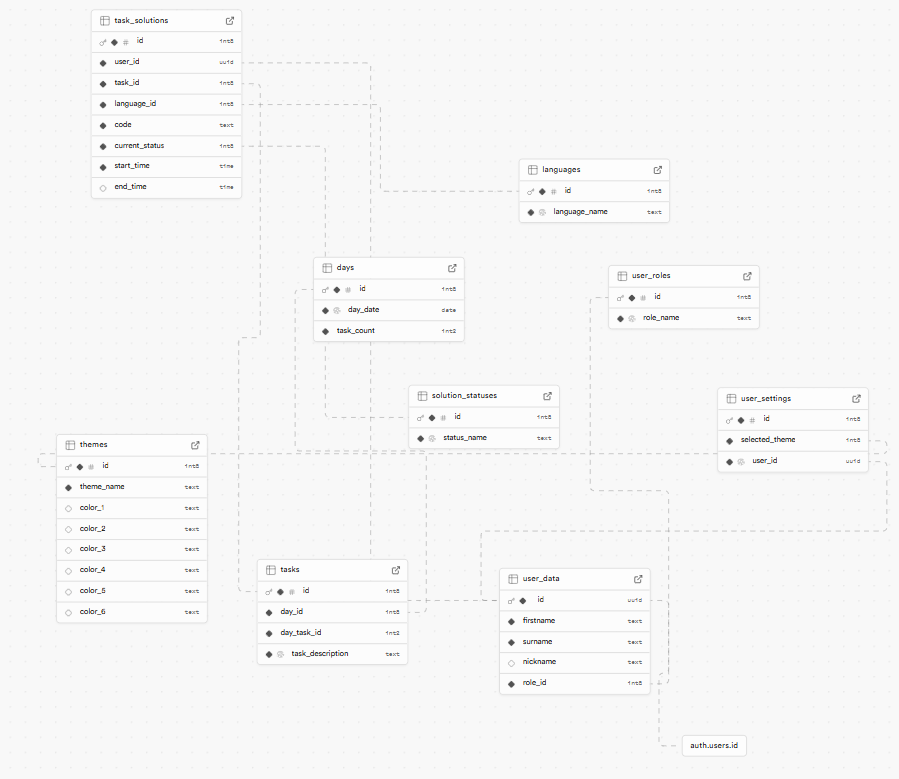


Рисунок 5 – Схема в Supabase

## Тестирование серверной части

## Разработка документации для тестирования сервера приложения

Для тестирования серверной части приложения была разработана следующая документация:

1. Тест-план;
2. Тестовый набор;
3. Тест-кейсы.

## Автоматизированное тестирование серверной части

Всего для серверной части было разработано 10 Юнит-тестов. Все юнит-тесты были написаны на языке SQL и выполнены с помощью установленного Supabase CLI.

Таблица 5 – Юнит-тесты

| Номер | Название | Описание | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | Проверка, что в схеме public присутствуют все 9 таблиц | Проверяет число таблиц в схеме Public | Число таблиц - 9 |
| 2 | Проверка, что в счётчике заданий на текущий день значение равно 2 | После вставки нового дня в базу и добавления к нему двух заданий проверяет счетчик заданий у данного дня | Значение счетчика - 2 |
| 3 | Проверка, что в таблицу заданий за текущий день добавилось 2 задания | После вставки нового дня в базу и добавления к нему двух заданий проверяет количество заданий в таблице с заданиями за текущий день | Число заданий - 2 |
| 4 | Проверка, что первое добавленное задание имеет id в рамках дня 1 | После вставки нового дня в базу и добавления к нему двух заданий проверяет индекс первого задания в рамках дня | Индекс - 1 |
| 5 | Проверка, что второе добавленное задание имеет id в рамках дня 2 | После вставки нового дня в базу и добавления к нему двух заданий проверяет индекс второго задания в рамках дня | Индекс - 2 |
| 6 | Проверка, что при попытке добавить третье задание на день происходит исключение | Проверяет, происходит ли исключение при попытке добавить задание на день, у которого уже есть 2 задания | Исключение «Нельзя добавить больше двух задач на этот день.» |
| 7 | Проверка ограничения уникальности у ролей | Проверяет, что нельзя добавить роль, если в БД уже существует роль с таким названием | Исключение «duplicate key value violates unique constraint» |
| 8 | Проверка значения по умолчанию у темы приложения в настройках | Проверяет значение темы по умолчанию при добавлении записи с настройками пользователя | Индекс темы - 1 |
| 9 | Проверка ограничения уникальности у настроек пользователя | Проверяет, что для каждого пользователя возможна только одна запись с настройками | Исключение «duplicate key value violates unique constraint» |
| 10 | Проверка каскадного удаления у параметров пользователя | Проверяет, что при удалении пользователя удаляется запись о его настройках | Запись удалена |

Ниже представлен листинг всех юнит-тестов:

Листинг 4 – Юнит-тесы

|  |
| --- |
| begin;  select plan( 10 );  --Юнит-тест 1: Проверка, что в схеме public присутствуют все 9 таблиц  SELECT is(cast(count(\*) as integer), 9) FROM information\_schema.tables  WHERE table\_schema = 'public';  --Вставка данных о дне и задачах на день для тестов ниже  BEGIN;  INSERT INTO days(day\_date) VALUES (CURRENT\_DATE);  INSERT INTO tasks(day\_id, task\_description) VALUES ((SELECT id FROM days WHERE day\_date = CURRENT\_DATE), 'test1');  INSERT INTO tasks(day\_id, task\_description) VALUES ((SELECT id FROM days WHERE day\_date = CURRENT\_DATE), 'test2');  COMMIT;  --Юнит-тест 2: Проверка, что в счётчике заданий на текущий день значение равно 2  SELECT is(cast((SELECT task\_count FROM days WHERE day\_date = CURRENT\_DATE)as integer), 2);  --Юнит-тест 3: Проверка, что в таблицу заданий за текущий день добавилось 2 задания  SELECT is(cast(count(\*) as integer), 2) from tasks WHERE day\_id = (SELECT id FROM days WHERE day\_date = CURRENT\_DATE);  --Юнит-тест 4 и 5: Проверка, что добавленные задания имеют id в рамках дня 1 и 2  SELECT is(cast(day\_task\_id as integer), 1) FROM tasks WHERE task\_description = 'test1';  SELECT is(cast(day\_task\_id as integer), 2) FROM tasks WHERE task\_description = 'test2';  --Юнит-тест 6: Проверка, что при попытке добавить третье задание на день происходит исключение  SELECT throws\_like('INSERT INTO tasks (day\_id, task\_description) VALUES ((SELECT id FROM days WHERE day\_date = CURRENT\_DATE),''3131'')', 'Нельзя добавить больше двух задач на этот день.');  --Удаление записей для очистки БД от данных, используемых для тестирования  BEGIN;  DELETE FROM tasks WHERE day\_id = (SELECT id FROM days WHERE day\_date = CURRENT\_DATE);  DELETE FROM days WHERE id = (SELECT id FROM days WHERE day\_date = CURRENT\_DATE);  COMMIT;  --Вставка данных о пользователе и его настройках для тестов ниже  BEGIN;  TRUNCATE TABLE themes RESTART IDENTITY CASCADE;  INSERT INTO user\_roles(role\_name) VALUES ('test');  INSERT INTO user\_data(id, firstname, surname, nickname, role\_id) VALUES (CAST('3818e8d2-79e3-40f8-9790-f10ecd332554' AS UUID), 'test', 'test', 'test', (SELECT id from user\_roles where role\_name = 'test'));  INSERT INTO themes(theme\_name) VALUES('test');  INSERT INTO user\_settings(user\_id) VALUES (CAST('3818e8d2-79e3-40f8-9790-f10ecd332554' AS UUID));  COMMIT;  --Юнит-тест 7: Проверка ограничения уникальности у ролей  SELECT throws\_like('INSERT INTO user\_roles(role\_name) VALUES (''test'');', 'duplicate key value violates unique constraint%');  --Юнит-тест 8: Проверка значения по умолчанию у темы приложения в настройках  SELECT is(cast(selected\_theme as integer), 1) FROM user\_settings WHERE user\_id = CAST('3818e8d2-79e3-40f8-9790-f10ecd332554' AS UUID);  --Юнит-тест 9: Проверка ограничения уникальности у настроек пользователя  SELECT throws\_like('INSERT INTO user\_settings(user\_id) VALUES (CAST(''3818e8d2-79e3-40f8-9790-f10ecd332554'' AS UUID));', 'duplicate key value violates unique constraint%');  --Юнит-тест 10: Проверка каскадного удаления у параметров пользователя  BEGIN;  DELETE FROM user\_data where id = CAST('3818e8d2-79e3-40f8-9790-f10ecd332554' AS UUID);  DELETE FROM user\_roles where role\_name = 'test';  DELETE FROM themes where theme\_name = 'test';  COMMIT;  SELECT is(cast(count(\*) as integer), 0) FROM user\_settings WHERE user\_id = CAST('3818e8d2-79e3-40f8-9790-f10ecd332554' AS UUID);  select \* from finish();  rollback; |

Все юнит-тесты были успешно пройдены:

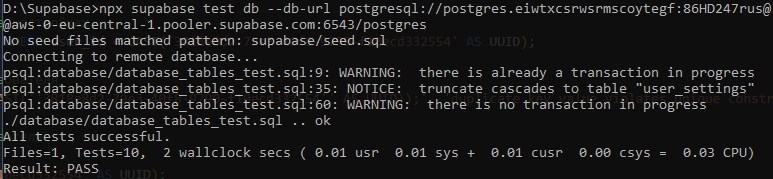


Рисунок 6 – Выполнение юнит-тестов

## Разработка мобильного приложения

## Подключение шрифтов и настройка ранее выбранных цветов

В приложении были подключены используемые шрифты JetBrains-Mono и Intro.

Также в приложении были прописаны цвета, которые относятся к его основной теме. Остальные цвета приложение получает из базы данных.

Листинг 5 – Подключение шрифтов

|  |
| --- |
| //Используемые шрифты val *Intro* = *FontFamily*(  *Font*(R.font.*intro*, FontWeight.Normal) )  val *JetBrains* = *FontFamily*(  *Font*(R.font.*jet\_brains\_mono\_regular*, FontWeight.Normal),  *Font*(R.font.*jet\_brains\_mono\_bold*, FontWeight.Bold) ) |

Листинг 6 – Цвета основной темы

|  |
| --- |
| //Базовые цвета приложения при запуске val *Background* = *Color*(0xff28242D) val *Primary* = *Color*(0xffDF3E78) val *SecondaryContainer* = *Color*(0xffD3CADC) val *Tertiary* = *Color*(0xffFFFFFF) val *Secondary* = *Color*(0xff62E3FF) val *OnBackground* = *Color*(0xff3F3343) |

## Верстка макета

Была выполнена верстка всех экранов, созданных в макете:

1. Экран авторизации. Функции:

* Отображение и скрытие пароля;
* Авторизация;
* Переход к регистрации.

1. Экран регистрации. Функции:

* Отображение и скрытие пароля;
* Проверка корректности почты;
* Проверка совпадения паролей;
* Регистрация;
* Переход на логин.



Рисунок 7 – Экраны авторизации и регистрации

1. Главный экран. Функции:

* Переходы в профиль, в настройки, в таблицу лидеров;
* Отображение списка заданий;
* Переход к описанию задания.

1. Настройки. Функции:

* Смена темы;
* Переход на главную.

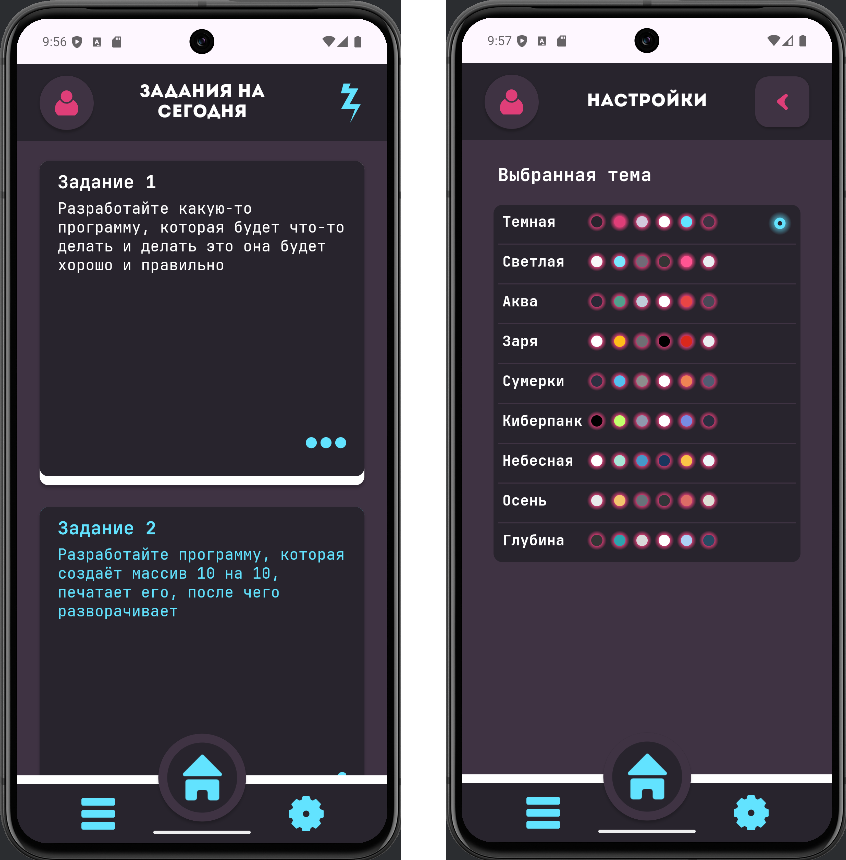


Рисунок 8 – Главный экран и настройки

1. Таблица лидеров. Функции:

* Отображение списка решений;
* Фильтрация решений по заданию, дате, языку.
* Отображение занятого места;
* Отображение потраченного времени;
* Переход к просмотру решения;
* (Для администратора) переход к проверке решения;
* Переход обратно на главную.

1. Профиль. Функции:

* Отображение Имени, Фамилии, Имени пользователя;
* Изменение Имени, Фамилии, Имени пользователя;
* Переход обратно на главную.

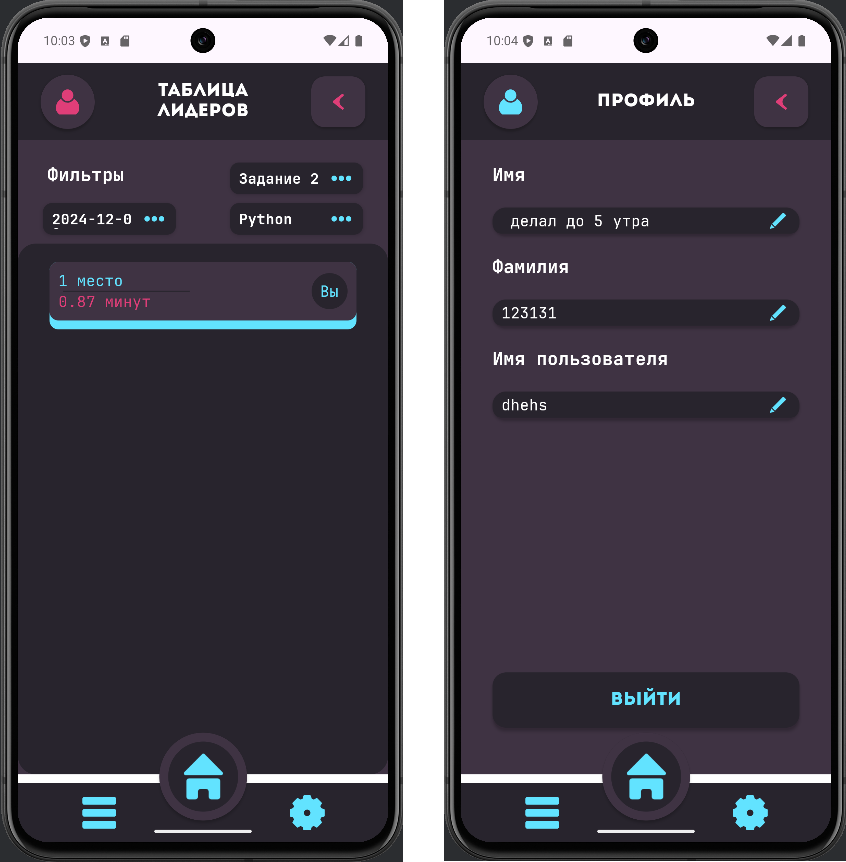


Рисунок 9 – Таблица лидеров и профиль

1. Описание задачи. Функции:

* Переход обратно на главную;
* Отображение описания задачи;
* Начало или продолжение решения задачи.

1. Редактор. Функции:

* Просмотр описания задачи;
* Написание кода решения;
* Смена языка задачи;
* Автосохранение решения в БД каждые 10 секунд;
* Завершение решения задачи.



Рисунок 10 – Описание задачи и редактор

1. Задания на завтра (Вариант главной для администратора). Функции:

* Просмотр уже добавленных заданий на след. день;
* Переход к добавлению задания;
* Переход в профиль;
* Переход к настройкам;
* Переход к проверке решений (таблица лидеров).

1. Новое задание. Функции:

* Добавление описания новой задачи;
* Создание задачи;
* Переход обратно на главную.

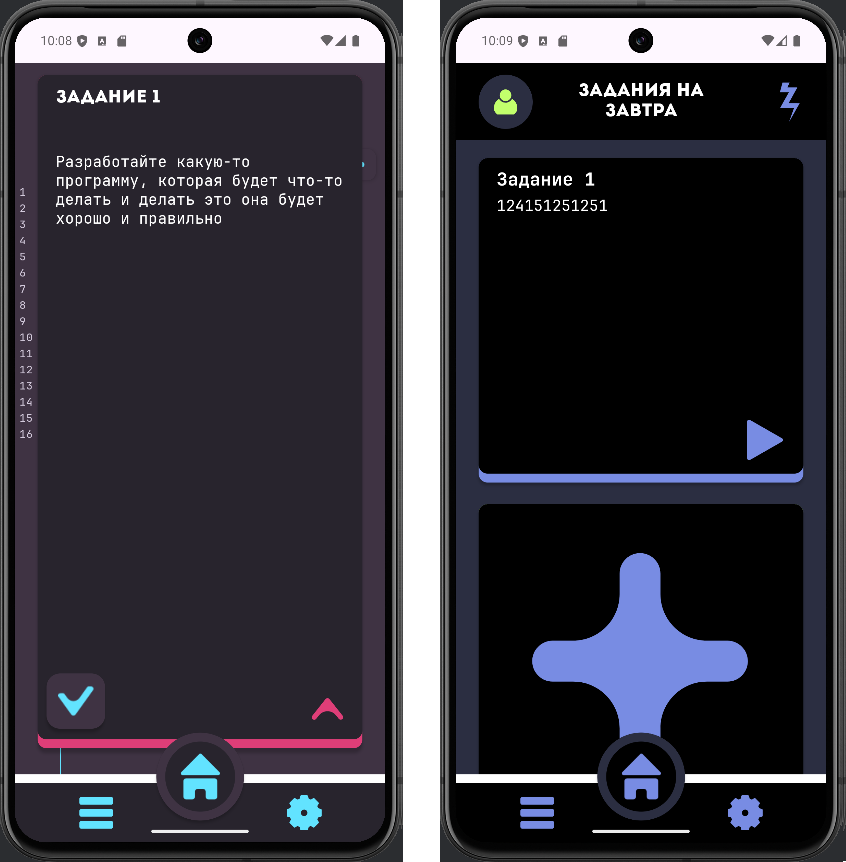


Рисунок 11 – Описание задачи в редакторе и главное окно администратора

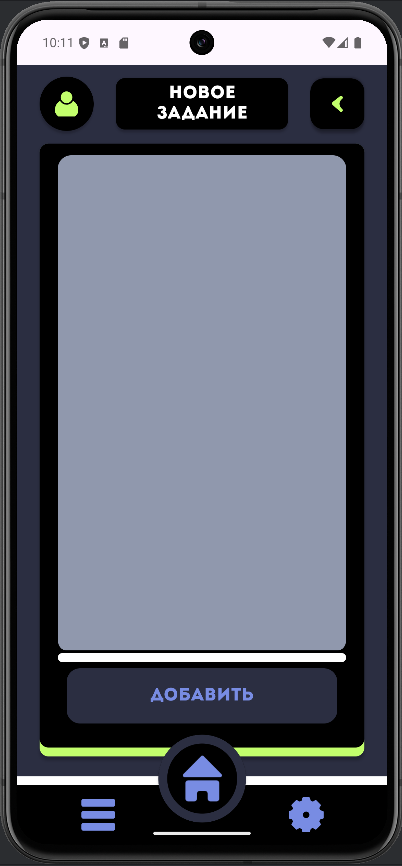


Рисунок 12 – Окно добавления задания

## Подключение библиотек и интернета

В файле AndroidManifest.xml был подключен интернет:

Листинг 7 – Подключение интернета

|  |
| --- |
| <uses-permission android:name="android.permission.INTERNET"/> |

В файле build.gradle.kts были добавлены зависимости для подключения библиотек:

Листинг 8 – Подключение библиотек

|  |
| --- |
| // Code editor *implementation*(*libs*.*compose*.*code*.*editor*) // Code editor  // Supabase *implementation*(platform(*libs*.*bom*)) *implementation*(*libs*.*postgrest*.*kt*) *implementation*(*libs*.*auth*.*kt*) *implementation*(*libs*.*realtime*.*kt*) *implementation*(*libs*.*ktor*.*client*.*cio*) // Supabase  // Hilt *implementation*("com.google.dagger:hilt-android:2.52") *kapt*("com.google.dagger:hilt-android-compiler:2.52") *implementation*("androidx.hilt:hilt-navigation-compose:1.2.0") // Hilt  //Syntax Highlight *implementation*("org.hibernate.validator:hibernate-validator:8.0.0.Final") //Syntax Highlight  //Tests *testImplementation*("org.jetbrains.kotlinx:kotlinx-coroutines-test:1.9.0") //Test |

## Подключение к серверной части приложения

Подключение к серверной части приложения было выполнено при помощи клиента Supabase для языка Kotlin.

Листинг 9 – Клиент Supabase

|  |
| --- |
| object Constants {  val supabase = *createSupabaseClient*(  supabaseUrl = "https://eiwtxcsrwsrmscoytegf.supabase.co",  supabaseKey = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6ImVpd3R4Y3Nyd3NybXNjb3l0ZWdmIiwicm9sZSI6ImFub24iLCJpYXQiOjE3Mjk3MDkxMjYsImV4cCI6MjA0NTI4NTEyNn0.71xBhsfi3ckDbjBdG9YqWPvrhb5w5YqfLBKhQWVB3cw"  ) **{** install(Auth)  install(Postgrest)  install(Realtime)  **}** } |

## Разработка методов

Были разработаны методы, позволяющие получать, преобразовывать, сохранять и фильтровать данные серверной части согласно тому, как они отображаются на экране.

Также были разработаны следующие дополнительные методы:

1. Метод для получения квадрата коэффициента соотношения экрана текущего устройства к эталонному экрану для масштабирования визуальных элементов (например – текст);
2. Метод для получения текущего времени.

## Тестирование мобильного приложения

## Разработка тестовой документации

В рамках выполнения тестирования мобильного приложения была разработана следующая тестовая документация:

1. Тест-план;
2. Тестовый набор;
3. Тест-кейс.

## Автоматизированное тестирование

## UI-тестирование

Был добавлен альтернативный класс для запуска тестов, необходимый для работы hilt в тестах.

Листинг 10 – CustomTestRunner

|  |
| --- |
| package com.example.codeblitz  import android.app.Application import android.content.Context import androidx.test.runner.AndroidJUnitRunner import dagger.hilt.android.testing.HiltTestApplication  class CustomTestRunner : AndroidJUnitRunner() {   override fun newApplication(cl: ClassLoader?, name: String?, context: Context?): Application {  return super.newApplication(cl, HiltTestApplication::class.*java*.*name*, context)  } } |

Листинг 11 – build.gradle.kts

|  |
| --- |
| *android* **{** namespace = "com.example.codeblitz"  compileSdk = 35   defaultConfig **{** applicationId = "com.example.codeblitz"  minSdk = 29  targetSdk = 34  versionCode = 1  versionName = "1.0"   testInstrumentationRunner = "com.example.codeblitz.CustomTestRunner"  vectorDrawables **{** useSupportLibrary = true  **}  }**  …… |

Все UI-тесты были написаны на языке Kotlin с использованием @HiltAndroidTest.

Таблица 6 – UI-тесты

| Номер | Название | Описание | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | Проверка перехода на страницу регистрации | Проверяет, что при нажатии кнопки «Зарегистрироваться» выполняется переход на страницу регистрации и страница отображается | Переход выполнен, страница отобразилась |
| 2 | Проверка отображения сообщений на странице регистрации и доступности кнопки | Проверяет, что при вводе значений в поля на странице регистрации предупреждения о некорректной почте и слабом пароле исчезают и кнопка «Регистрация» становится доступна | После ввода значений предупреждения перестали отображаться, кнопка стала доступна |
| 3 | Проверка ввода в поля логина и пароля и последующей авторизации | Проверяет, что возможно ввести значения в поле логина и пароля на странице авторизации и что авторизация выполняется | Значения вводятся в поля логина и пароля. Авторизация успешно выполняется |
| 4 | Проверка отображения плиток заданий у администратора | Проверяет, что у администратора все отображаются 2 плитки с заданиями | Число плиток равно 2 |
| 5 | Проверка перехода на страницу добавления задания при нажатии на элемент LazyColumn | Проверяет, что при нажатии на элемент LazyColumn, являющимся плиткой добавления задания, выполняется переход на страницу добавления задания | Переход выполнен |

Ниже представлен листинг всех UI-тестов:

Листинг 12 – UI-тесты

|  |
| --- |
| package com.example.codeblitz  import android.os.SystemClock.sleep import androidx.compose.ui.test.assertCountEquals import androidx.compose.ui.test.assertIsDisplayed import androidx.compose.ui.test.assertIsEnabled import androidx.compose.ui.test.assertIsNotDisplayed import androidx.compose.ui.test.assertIsNotEnabled import androidx.compose.ui.test.junit4.createAndroidComposeRule import androidx.compose.ui.test.onAllNodesWithTag import androidx.compose.ui.test.onNodeWithTag import androidx.compose.ui.test.performClick import androidx.compose.ui.test.performTextInput import androidx.test.ext.junit.runners.AndroidJUnit4 import com.example.codeblitz.view.MainActivity.MainActivity import dagger.hilt.android.testing.HiltAndroidRule import dagger.hilt.android.testing.HiltAndroidTest import org.junit.Before import org.junit.Rule import org.junit.Test import org.junit.runner.RunWith  */\*\*  \* UI-тесты  \*/* @HiltAndroidTest @RunWith(AndroidJUnit4::class) class CodeBlitzInstrumentalTests {   @get:Rule  val composeTestRule = *createAndroidComposeRule*<MainActivity>()   @get:Rule  var hiltRule = HiltAndroidRule(this)   @Before  fun setUp() {  hiltRule.inject()  }   */\*\*  \* Проверка перехода на страницу регистрации  \*/* @Test  fun testNavigationToRegister() {  composeTestRule.*onNodeWithTag*("Login").*assertIsDisplayed*()   composeTestRule.*onNodeWithTag*("To Register Button").*performClick*()   composeTestRule.*onNodeWithTag*("Register").*assertIsDisplayed*()  }   */\*\*  \* Проверка отображения сообщений на странице регистрации и доступности кнопки  \*/* @Test  fun testRegisterWarning() {  composeTestRule.*onNodeWithTag*("To Register Button").*performClick*()   composeTestRule.*onNodeWithTag*("Wrong Email").*assertIsDisplayed*()  composeTestRule.*onNodeWithTag*("Weak Password").*assertIsDisplayed*()   composeTestRule.*onNodeWithTag*("Login Register").*performClick*()  composeTestRule.*onNodeWithTag*("Login Register").*performTextInput*("errorfadeev@yandex.ru")  composeTestRule.*onNodeWithTag*("Wrong Email").*assertIsNotDisplayed*()   composeTestRule.*onNodeWithTag*("Password Register").*performClick*()  composeTestRule.*onNodeWithTag*("Password Register").*performTextInput*("1234567890")  composeTestRule.*onNodeWithTag*("Weak Password").*assertIsNotDisplayed*()   composeTestRule.*onNodeWithTag*("Register Button").*assertIsNotEnabled*()   composeTestRule.*onNodeWithTag*("Repeat Password").*performClick*()  composeTestRule.*onNodeWithTag*("Repeat Password").*performTextInput*("1234567890")   composeTestRule.*onNodeWithTag*("Register Button").*assertIsEnabled*()  }   */\*\*  \* Проверка ввода в поля логина и пароля и последующей авторизации  \*/* @Test  fun testLoginTextFieldsAndLoginIn() {  composeTestRule.*onNodeWithTag*("Login Login").*performClick*()  composeTestRule.*onNodeWithTag*("Login Login").*performTextInput*("errorfadeev@yandex.ru")   composeTestRule.*onNodeWithTag*("Password Login").*performClick*()  composeTestRule.*onNodeWithTag*("Password Login").*performTextInput*("123")   composeTestRule.*onNodeWithTag*("Login Button").*performClick*()  sleep(10000)  composeTestRule.*onNodeWithTag*("Main").*assertIsDisplayed*()  }   */\*\*  \* Проверка отображения плиток заданий у администратора  \*/* @Test  fun testAdminMainScreenTasksDisplay() {  composeTestRule.*onNodeWithTag*("Login Login").*performClick*()  composeTestRule.*onNodeWithTag*("Login Login").*performTextInput*("errorfadeev@yandex.ru")  composeTestRule.*onNodeWithTag*("Password Login").*performClick*()  composeTestRule.*onNodeWithTag*("Password Login").*performTextInput*("123")  composeTestRule.*onNodeWithTag*("Login Button").*performClick*()  sleep(10000)  sleep(10000)  composeTestRule.*onAllNodesWithTag*("TaskElement").*assertCountEquals*(2)  }   */\*\*  \* Проверка перехода на страницу добавления задания при нажатии на элемент LazyColumn  \*/* @Test  fun testLazyColumnOnItemClickNavigation() {  composeTestRule.*onNodeWithTag*("Login Login").*performClick*()  composeTestRule.*onNodeWithTag*("Login Login").*performTextInput*("errorfadeev@yandex.ru")  composeTestRule.*onNodeWithTag*("Password Login").*performClick*()  composeTestRule.*onNodeWithTag*("Password Login").*performTextInput*("123")  composeTestRule.*onNodeWithTag*("Login Button").*performClick*()  sleep(10000)  sleep(10000)  composeTestRule.*onNodeWithTag*("TaskElementID - 1").*performClick*()  composeTestRule.*onNodeWithTag*("AddTask").*assertIsDisplayed*()  } } |

Все UI-тесты были успешно пройдены:

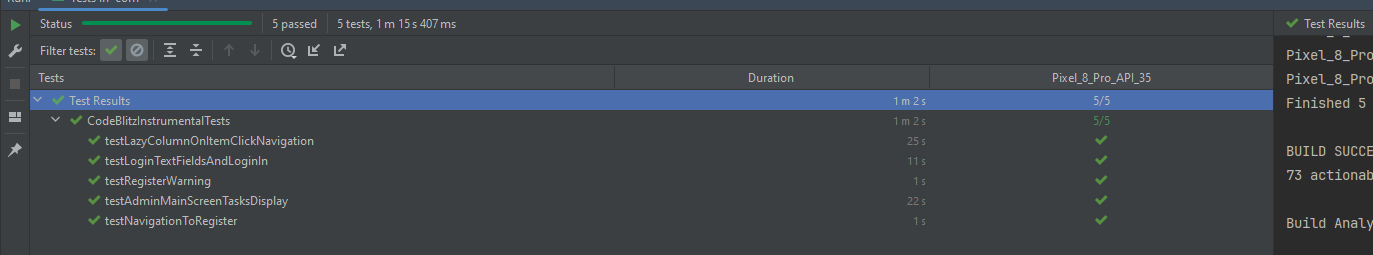


Рисунок 13 – Выполнение юнит-тестов

## Unit-тестирование

Все юнит-тесты были написаны на языке Kotlin.

Таблица 7 – Юнит-тесты

| Номер | Название | Описание | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | Проверка обновления данных регистрации | Проверяет, что после вставки данных в LoginViewModel они корректно сохраняются | Сохранились те же данные, что и были введены |
| 2 | Проверка доступности кнопки входа при вводе данных в поля | Проверяет, что поле, отвечающее за состояние кнопки, изменило свое состояние при вводе логина и пароля в поля | Поле изменило значение на true |
| 3 | Проверка потока навигации на страницу регистрации | Проверяет, что значение потока навигации после вызова функции навигации стало содержать в себе путь на страницу регистрации | Значение потока навигации равно объекту, содержащему путь к регистрации |
| 4 | Проверка функции подсчета строк | Проверяет корректный формат строки с номерами строчек редактора | Формат строки соответствует количеству строк введенного текста |
| 5 | Проверка записи описания задачи в поле | Проверяет, что описание задачи корректно записалось в поле ViewModel | В поле ViewModel записалось введенное значение |

Ниже представлен листинг всех юнит-тестов:

Листинг 13 – Юнит-тесты

|  |
| --- |
| package com.example.codeblitz  import androidx.compose.ui.text.input.TextFieldValue import com.example.codeblitz.domain.AddTaskViewModel import com.example.codeblitz.domain.LoginViewModel import com.example.codeblitz.domain.RegisterViewModel import com.example.codeblitz.domain.navigation.Routes import com.example.codeblitz.model.LoginData import com.example.codeblitz.view.MainActivity.components.countAndFormatNewLines import kotlinx.coroutines.test.runTest import org.junit.Assert.assertEquals import org.junit.Test  */\*\*  \* Юнит-тесты  \*/* class CodeBlitzUnitTests {  */\*\*  \* Проверка обновления данных регистрации  \*/* @Test  fun testUpdateRegisterData() {  val viewModel = RegisterViewModel()  *assert*(viewModel.registerData.password.*isEmpty*())  *assert*(viewModel.registerData.login.*isEmpty*())   viewModel.updateRegisterData(  newRegisterData = LoginData("123", "1234")  )   assertEquals("1234", viewModel.registerData.password)  assertEquals("123", viewModel.registerData.login)  }   */\*\*  \* Проверка доступности кнопки входа при вводе данных в поля, отвечающие за логин и пароль  \*/* @Test  fun testIsLoginButtonActive() {  val viewModel = LoginViewModel()  *assert*(!viewModel.isButtonEnabled)   viewModel.updateLoginData(  newLoginData = LoginData("123", "1234")  )   *assert*(viewModel.isButtonEnabled)  }   */\*\*  \* Проверка потока навигации на страницу регистрации  \*/* @Test  fun testToRegisterNavigationFlow() = *runTest* **{** val viewModel = LoginViewModel()  viewModel.navigateToRegister()   assertEquals(Routes.Register, viewModel.navigationStateFlow.value)  **}** */\*\*  \* Проверка функции подсчета строк  \*/* @Test  fun testCountLines() {  val string = """  # Example usage:  numbers = list(range(1, 101)) #Test with a small list  start\_time = time.time()  result = incredibly\_inefficient\_sum(numbers) # This will break with a longer list due to recursion depth  end\_time = time.time()  print(f"Sum: {result}")  print(f"Time taken: {end\_time - start\_time:.4f} seconds")     numbers = list(range(1,10)) #Test with a small list that will not cause stack overflow  start\_time = time.time()  result = incredibly\_inefficient\_sum(numbers)  end\_time = time.time()  print(f"Sum: {result}")  print(f"Time taken: {end\_time - start\_time:.4f} seconds")  """.*trimIndent*()   val expected = "1\n2\n3\n4\n5\n6\n7\n8\n9\n10\n11\n12\n13\n14\n15\n16"  val result = *countAndFormatNewLines*(TextFieldValue(string))   assertEquals(expected, result)  }   */\*\*  \* Проверка записи описания задачи в поле  \*/* @Test  fun testTaskDesc() {  val viewModel = AddTaskViewModel()   val expected = """  the most beautiful description  """.*trimIndent*()   viewModel.setTaskDesc(expected)   assertEquals(expected, viewModel.taskDesk)  } } |

Все юнит-тесты были успешно пройдены:

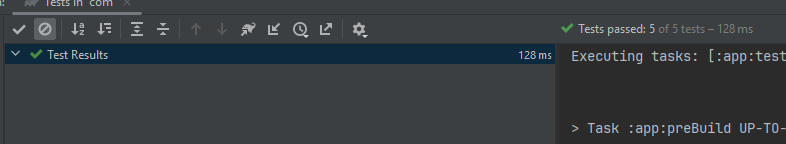


Рисунок 14 – Выполнение юнит-тестов

# ЗАКЛЮЧЕНИЕ

В заключение следует отметить, что в рамках данной работы было успешно разработано мобильное приложение для обучения программированию в игровой форме. Все поставленные задачи, от разработки технического задания до тестирования и презентации готового продукта, были выполнены.

Созданное приложение предоставляет пользователям удобный и доступный инструмент для ежедневной практики в решении алгоритмических задач, способствуя повышению уровня навыков программирования и стимулируя к постоянному самосовершенствованию в этой области.