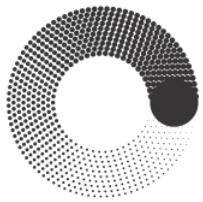


ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

*Факультет информационных технологий
Кафедра Информатики и информационных технологий*

**направление подготовки
09.03.02 «Информационные системы и технологии»**

КУРСОВОЙ ПРОЕКТ

Дисциплина: Технологии кроссплатформенного программирования

Тема: Онлайн-сервис для организации интерактивных викторин в реальном
времени “Рубильник”

Выполнили: студенты группы 211-726

Монахов А. А., Рыбалко К. А., Лебедев С. В.

(Фамилия И.О.)

Дата, подпись _____
(Дата) _____ (Подпись)

Проверил: _____
(Фамилия И.О., степень, звание) _____ (Оценка)

Дата, подпись _____
(Дата) _____ (Подпись)

Замечания:

Москва

2024

Оглавление

ВВЕДЕНИЕ.....	3
Глава 1. Проектирование.....	5
1.1 Описание предметной области.....	5
1.2 Выбор инструментов.....	7
1.3 Планирование.....	9
1.4 База данных.....	12
1.5 Дизайн.....	14
Глава 2. Разработка.....	18
2.1 Сервер.....	18
2.2 Веб-сайт.....	24
2.3 Android.....	32
ЗАКЛЮЧЕНИЕ.....	34
Библиографический список.....	35
ПРИЛОЖЕНИЯ.....	36

ВВЕДЕНИЕ

Тема проекта – онлайн-сервис для организации интерактивных викторин в реальном времени “Рубильник”.

Проект “Рубильник” представляет собой инновационную платформу, разработанную с целью решения проблемы отсутствия качественных онлайн викторин на отечественном рынке. Эта платформа предоставляет пользователям возможность создавать и проводить онлайн викторины любой сложности и тематики. Она также обеспечивает удобный интерфейс для участников, позволяющий им легко присоединиться к викторине, отвечать на вопросы и отслеживать свои результаты.

Платформа “Рубильник” также предлагает широкий спектр функций для организаторов викторин, включая возможность создания персонализированных викторин, управления участниками и мониторинга результатов. Благодаря этому проекту, пользователи могут наслаждаться увлекательным и интерактивным опытом проведения и участия в онлайн викторинах, что ранее было недоступно на отечественном рынке.

Концепция проекта состоит в создании единой платформы участия и проведения викторин в формате реального времени, ориентированной на русскоговорящую аудиторию. Проект состоит из мобильного, веб-приложения и сервера.

Цели проекта:

- Предоставить пользователям возможность создавать, проводить интерактивные викторины в реальном времени и участвовать в них.
- Решить проблему отсутствия качественных онлайн викторин на отечественном рынке.
- Содействовать развитию образовательных и развлекательных мероприятий, предоставляя удобный инструмент для проведения викторин.

- Повысить уровень вовлеченности пользователей и создать новые возможности для социального взаимодействия через проведение онлайн викторин.

Глава 1. Проектирование

1.1 Описание предметной области

Потребность рынка в онлайн сервисе для организации викторин “Рубильник” обусловлена растущим спросом на интерактивные формы обучения, развлечения и ведения статистики. Поэтому благодаря нашей разработке, современные люди будут активнее участвовать в мероприятиях и получать положительные опыт и воспоминания. Через интерактивный формат проще проводить обучение и контроль знаний.

Потенциальные потребители могут столкнуться с проблемой отсутствия такого инструмента на территории РФ.

Проект может привлечь покупателя своей простотой и достаточно широким функционалом, а также доступностью для широкой аудитории пользователей.

Целевые сегменты рынка, куда планируется продвигать проект:

- Образовательные организации;
- Бизнес (проведение опросов);
- Развлекательная сфера.

Уникальные особенности данного сервиса: проведение викторин в реальном времени.

Аналоги:

- kahoot.it
- AhaSlides
- quizizz.com

Сервис “Рубильник” состоит из трех модулей;

1. Модуль веб-приложения. Модуль является системой создания, редактирования и проведения викторин в виде комнат (с подключением участников).
2. Модуль мобильного приложения. Модуль является Android-приложением с функциональными возможностями подключения к сессии (комнате) и участии в викторинах как пользователь.
3. Серверный модуль. Функционал модуля состоит в создании и поддержке сессии хост-пользователи для возможности проведения викторины в реальном времени.

1.2 Выбор инструментов

Для разработки Web-модуля был выбран frontend фреймворк React.JS, благодаря удобности использования функциональных компонентов и динамического серверного рендеринга. Поверх него был использован React-Router для реализации routing`а. Сборка веб-модуля осуществлена с помощью утилиты Vite.

Дополнительные библиотеки, используемые в Web-модуле: axios – реализация REST API связи с бэкэндом, socket.io для использования websocket, cryptjs – хэширование паролей, random – генерация случайных значений ключей компонентов, react-icons – импорт иконок и eslint – импортированный линтер для соблюдения стилизации кода.

В качестве БД был выбран сервис noSQL MongoDB по причине отсутствия необходимости в relation-связях между классами данных и гибкости noSQL-модели.

Для связи Back-сервера с БД использовался модуль ORM Prisma.

Для организации непрерывного канала связи во время проведения викторины была выбрана технология WebSocket, конкретнее – библиотека SocketIO поскольку предоставляет удобные функции для кодирования.

Так как нет необходимости в непрерывном канале связи при передачи команд в БД со стороны клиента (при регистрации, удалении, обновлении данных пользователя) использовались http-запросы на Back-сервер на NodeJS.

Для описания логики Back-сервера использовалась среда NodeJS, поскольку использует привычный всем участникам команды язык web-программирования JavaScript совместно с веб-фреймворком Express, упрощающим код.

Для разработки мобильного приложения была выбрана платформа Android Studio и язык программирования Java. Android Studio является официальной интегрированной средой разработки (IDE) для разработки приложений для Android, разработанной самим Google. Включает в себя встроенную поддержку Android Software Development Kit (SDK), что упрощает процесс создания, отладки

и тестирования приложений для Android. Java является одним из самых популярных языков программирования, существует обширное сообщество разработчиков.

Для создания дизайн-проекта использовалась Figma. Figma позволяет нескольким пользователям работать в одном и том же проекте одновременно, просматривать изменения в режиме реального времени и взаимодействовать друг с другом через комментарии и обсуждения.

Для взаимодействия участников внутри команды были использованы:

- Вконтакте: вся команда зарегистрирована на этой площадке; используется в большей мере для учебы; невозможность удалить совместный чат у всех участников команды; оперативная коммуникация.
- Discord: более простой способ организации видеоконференции; наличие аккаунтов на площадке; стабильная видеосвязь.
- GitHub: предоставляет мощные инструменты для управления версиями кода, интегрируется с другими инструментами.

Для организации командной работы над проектом был выбран сервис smartsheet. Smartsheet предоставляет широкий спектр инструментов для создания и настройки различных типов планов, таблиц, графиков Ганта, диаграмм и других элементов планирования. Smartsheet обеспечивает возможность совместной работы над проектами, обмена информацией, комментирования задач и обсуждения проектных вопросов. Есть автоматизация процессов, ведение отчетности и построение аналитики.

1.3 Планирование

Календарный план проекта – это инструмент для планирования и управления ходом проекта. Он помогает организовать работу, определить необходимые ресурсы и сроки выполнения задач, а также управлять временем и распределением задач между участниками проекта. Календарный план позволяет создать структуру и организовать работу, определить необходимые ресурсы для выполнения задач, управлять временем и сроками выполнения задач, выявить потенциальные риски и проблемы, связанные с сроками выполнения задач, оценить производительность команды и прогресс выполнения проекта. (рис. 1.3.1)

	Tasks	Пред...	Начало выполнения	Завершен... выполнения	Прод...	Задержка	Статус задачи	Исполнитель
1	СТАРТ ПРОЕКТА		01.11.23	01.11.23	1д	55д		
2	▪ Провести исследование и анализ рынка	1НН	01.11.23	04.11.23	4д	4д	Выполнено	Рыбакло Константин
6	▪ Определить основные функциональные требования к проекту	2НН	01.11.23	07.11.23	7д	7д	Выполнено	Лебедев Степан
10	▪ Организация БД	2НН	01.11.23	05.11.23	5д	5д	Выполнено	Монахов Артем
15	▪ Создание черновых прототипов дизайна	2	05.11.23	13.11.23	9д	9д	Выполнено	Рыбакло Константин
19	▪ Создать прототип для тестирования и получения обратной связи, определить итоговый перечень функциональных требований к проекту	6	08.11.23	13.11.23	6д	6д	Выполнено	Лебедев Степан
24	▪ Настроить БД	10	06.11.23	13.11.23	8д	8д	Выполнено	Монахов Артем
29	▪ Разработать уникальный стиль пользовательского интерфейса	15	14.11.23	20.11.23	7д	7д	Выполнено	Рыбакло Константин
33	▪ Закончить веб-разметку прототипа приложения и его реализовать логику	19	14.11.23	20.11.23	7д	7д	Выполнено	Лебедев Степан
36	▪ Создать связь между хостом и "пультами"	24	14.11.23	19.11.23	6д	6д	Выполнено	Монахов Артем
41	▪ Закончить создание пользовательского интерфейса с учетом корректировок обратной связи	29	21.11.23	27.11.23	7д	7д	Выполнено	Рыбакло Константин
45	▪ Совершить переход от прототипа к реализации веб-компонентов проекта	33	21.11.23	26.11.23	6д	6д	Выполнено	Лебедев Степан
49	▪ Создание шаблона мобильного приложения	36	20.11.23	22.11.23	3д	3д	Выполнено	Монахов Артем
53	▪ Frontend в мобильной разработке	41	28.11.23	02.12.23	5д	5д	Выполнено	Рыбакло Константин
55	▪ Закончить создание frontend'a	45	27.11.23	04.12.23	8д	8д	Выполнено	Лебедев Степан
59	▪ Организация взаимодействия сервера	53; 55	05.12.23	10.12.23	6д	6д	Выполнено	Монахов Артем
62	▪ Подготовка материала для презентации проекта	59	11.12.23	15.12.23	5д	5д	В работе	Рыбакло Константин
63	▪ Выполнить тестирование и отладку приложения	59	11.12.23	15.12.23	5д	5д	В работе	Лебедев Степан
68	▪ Тестирование сервиса	63НН	11.12.23	15.12.23	5д	5д	В работе	Монахов Артем
74	▪ Создание видеопрезентации проекта. Сбор и анализа обратной связи. Корректировка выявленных неточностей	62	16.12.23	17.12.23	2д	2д	Выполнено	Рыбакло Константин
75	▪ Закрытое тестирование сервиса	68	16.12.23	22.12.23	7д	3д	Выполнено	Лебедев Степан Монахов Артем
80	ЗАВЕРШЕНИЕ ПРОЕКТА	75	23.12.23	23.12.23	1д			

Рисунок 1.3.1 Календарный план

Визуализация календарного плана в виде временных линий для каждой задачи – диаграмма Ганта. Было произведено распределение времени на выполнение каждой задачи, учет зависимостей между задачами и ресурсы, необходимые для выполнения. (рис. 1.3.2)

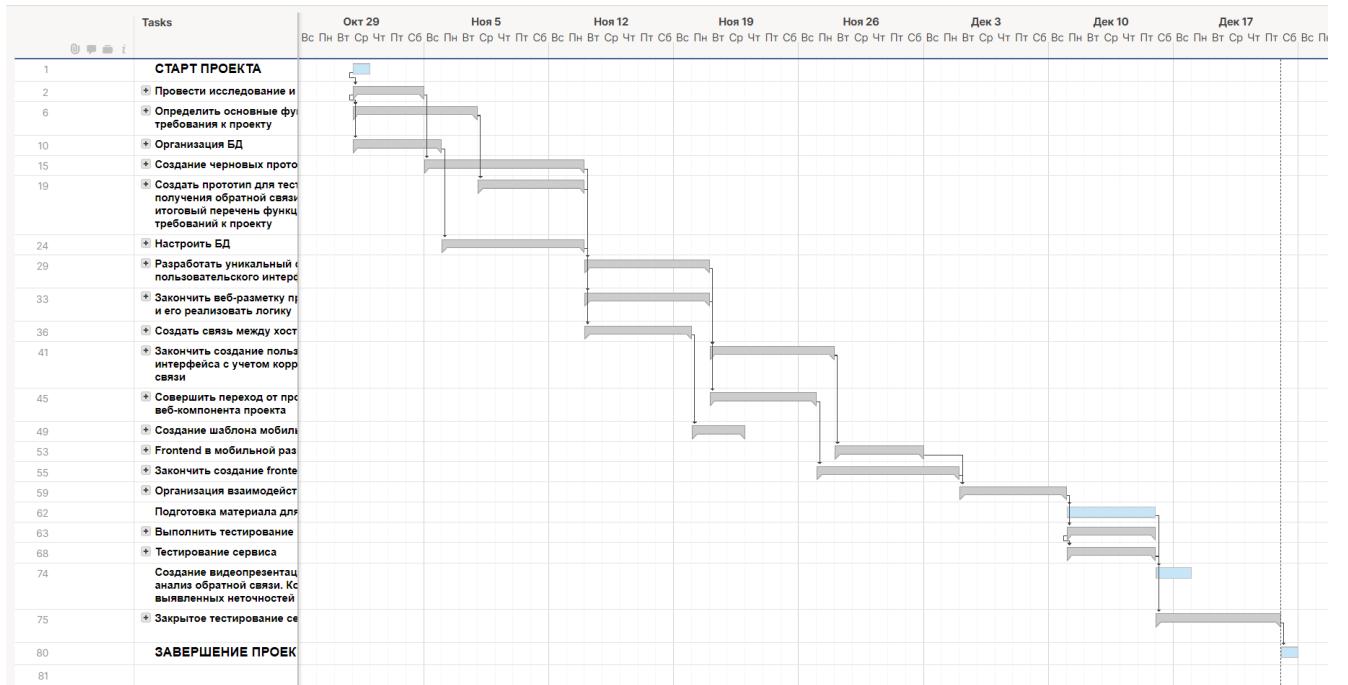


Рисунок 1.3.2 Диаграмма Ганта

Необходимо было создать отдельные карточки каждой задачи для визуализации краткосрочных планов разработки проекта. Они содержат название задачи, описание, сроки выполнения, ответственного исполнителя и возможные зависимости от других задач. (рис. 1.3.3)

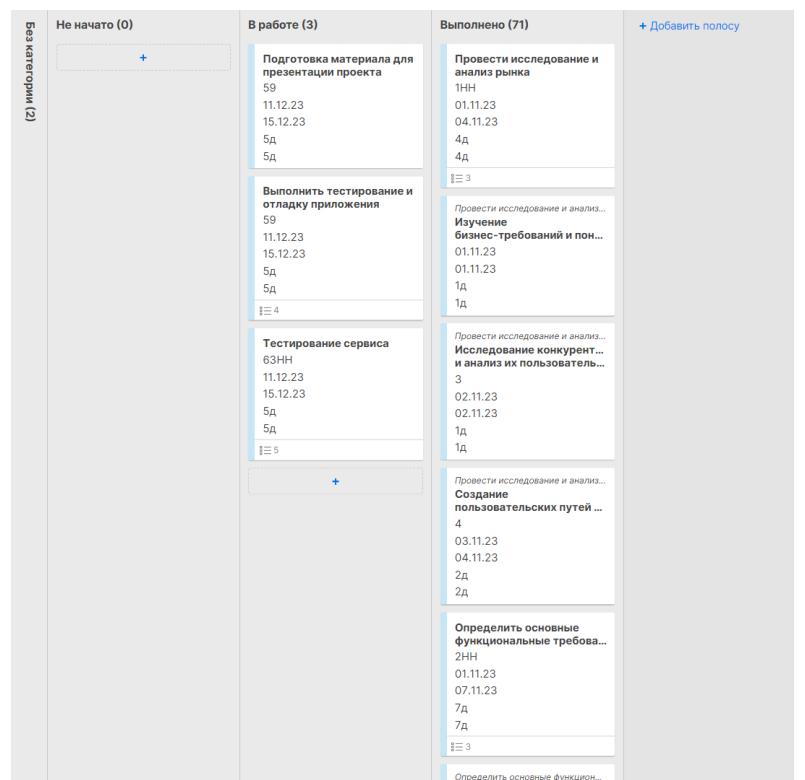


Рисунок 1.3.3 Постановка задач

Минимальные требования проекта “Рубильник”: реализация веб-хоста, текстовый редактор викторин, мобильное приложение (пульт), подключение пультов к хосту.

Максимальные требования проекта “Рубильник”: реализация веб-хоста, графический редактор викторин и опросов, различные режимы и анимации, мобильное приложение (пульт), подключение пультов к хосту, аккаунты, сохранение прогресса.

1.4 База данных

prisma.schema, описание структур хранимых данных (Листинг 1.4.1):

Листинг 1.4.1 База данных

```
generator client {
    provider = "prisma-client-js"
}

datasource db {
    provider = "mongodb"
    url      = env("DATABASE_URL")
}

model User {
    id      String @id @map("_id")
    email   String @unique
    name    String
    password String
    quizzes Quiz[]
}

type Quiz {
    title   String
    questions Question[]
}

type Question {
    text     String
    answers  Choice[]
    validIndex Int
}

type Choice {
    text String
}
```

Данный код определяет схему Prisma для простого приложения викторины.

Клиентский блок generator указывает, что клиент Prisma должен быть сгенерирован с помощью библиотеки prisma-client-js.

В блоке db источника данных указывается поставщик базы данных (MongoDB) и URL-адрес подключения. Этот URL-адрес считывается из переменной среды “DATABASE_URL”.

Модель-блок “User” определяет коллекцию пользователей с полями для их электронной почты, имени, пароля и списком тестов. Поле “id” помечено как первичный ключ, и его имя сопоставляется с полем “MongoDB _id”.

Блок типа “Quiz” определяет коллекцию тестов с заголовком и списком вопросов.

Блок типа “Question” определяет вопрос с текстом, списком ответов и validIndex, указывающим, какой ответ правильный.

Блок “type Choice” определяет вариант ответа, используя только текстовое поле.

1.5 Дизайн

Онлайн-сервис “Рубильник” создан в большей мере для организации викторин в образовательных целях. Именно поэтому дизайн должен быть интуитивно понятным и простым для пользования обучающихся и педагогов. Для этого были созданы элементы, которые имеют свой характерный стиль, цвет и текстовое сопровождение. Например, кнопка “Подключиться” имеет не иконку подключения, а слово “Подключиться”.

Определение функциональных требований для мобильной версии:

- Реализация подключения к викторине с вводом имени.
- Реализация просмотра и ответа на вопрос.
- Реализация просмотра итога викторины.
- Реализация англо-русской версии.
- Реализация синхронизации информации.

Определение функциональных требований для веб-версии:

- Реализация конструктора викторин с возможностью создания, редактирования, удаления.
- Реализация конструктора вопросов с возможностью создавать новые вопросы с параметрами: текст вопроса, варианты ответа, правильный вариант ответа. Возможность изменять кол-во вариантов ответов на вопрос (2-4), выбор правильного варианта ответа, возможность редактировать и удалять вопросы.
- Реализация элементов аутентификации, регистрации, выхода и удаление аккаунта пользователя.
- Реализация синхронизации информации.
- Реализация сохранения и получения викторин из backend-сервера.
- Реализация модуля “игры” (проведение викторины) с элементами: “Лобби” – выводить код подключения, кол-во и имена подключенных игроков, “Игра” – выводить тексты вопросов викторины, варианты ответов на вопросы викторины, правильные варианты ответов на

вопросы викторины, “Конец игры” – вывод рейтинговой таблицы с отсортированным списком имен игроков, их места в рейтинговой таблице, кол-во их очков.

Функционал веб-версии должен быть устойчив, адаптивен, соответствовать стилю обозначенного дизайна, поддерживать одновременную работу нескольких инстансов.

Определение страниц для мобильной версии (приложение А):

1. Окно подключения к хосту;
2. Окно ожидания сессии;
3. Окно вопроса;
4. Окно результата викторины;
5. Окно настроек.

Определение страниц для веб-версии (приложение А):

1. Страница приветствия;
2. Страница аутентификации и регистрации пользователя;
3. Страница викторин;
4. Страница создания и редактирования викторины;
5. Страница ожидания викторины;
6. Страница вопроса;
7. Страница итога викторины.

Определение цветовой палитры (рис. 1.5.2)



Рисунок 1.5.2 Цветовая палитра

Используемый шрифт: Russo One.

Основные элементы дизайна Веб:

- Кнопка. Высота: 70. Ширина: 185, 230, 330, 390. Скругление: 5. Ширина обводки: 2. При наведении на кнопку ширина обводки: 4. Цвет обводки и текста: D6BF81. Padding: 20. Размер текста: 26.
- Кнопка выбора. Высота: 300, 620. Ширина: 300. Скругление: 5. Цвет заливки: 709B95, F08A5D, B83B5E, D9D9D9. Цвет текста: 242424. Размер текста: 36.
- Квадратная кнопка. Высота: 50, 300. Ширина: 50, 300. Скругление: 5. Ширина обводки: 2. При наведении на кнопку ширина обводки: 4. Цвет обводки и текста: D6BF81, 242424. Размер текста: 26. При нажатии заливка цвета D6BF81 на 10% либо ничего.
- Кнопка добавления (как квадратная кнопка). Высота: 150. Ширина: 150.
- Поле ввода. Высота: 70. Ширина: 330, 390. Скругление: 5. Цвет заливки: B0B0B0. При наведении цвет заливки: B0B0B0. Цвет текста: 242424. Цвет подсказки: 525252. Размер текста: 18.
- Заголовок. Размер: 36. Цвет: 0FD4CD, 242424, D9D9D9.
- Основной текст. Размер: 26. Цвет: 242424, D6BF81.
- Дополнительный текст. Размер: 18. Цвет: 0FD4CD, D6BF81.
- Подвал (футер). Размер: 12. Цвет: B0B0B0. Прозрачность: 50%.

Основные элементы дизайна Android:

- Кнопка. Высота: 70. Ширина: 230. Скругление: 5. Ширина обводки: 2. При наведении на кнопку ширина обводки: 4. Цвет обводки и текста: D6BF81. Padding: 20. Размер текста: 26.
- Кнопка выбора. Высота: 70. Ширина: 70. Скругление: 5. Цвет заливки: 709B95, F08A5D, B83B5E, D9D9D9. При наведении цвет заливки: B0B0B0. Цвет текста: 242424. Размер текста: 20.
- Switch (переключатель). Высота: 70. Ширина: 150. Скругление: 5. Цвет головы и текста: D6BF81. Цвет заливки трэка: B0B0B0. Padding: 20. Размер текста: 20.

- Поле ввода. Высота: 70. Ширина: 230. Скругление: 5. Цвет заливки: B0B0B0. При наведении цвет заливки: B0B0B0. Цвет текста: 242424. Цвет подсказки: 525252. Размер текста: 18.
- Заголовок. Размер: 34. Цвет: 0FD4CD, D6BF81.
- Основной текст. Размер: 26. Цвет: D6BF81.
- Дополнительный текст. Размер: 20. Цвет: 242424.

Создание пользовательского интерфейса, прототипов и макетов. (рис. 1.5.2)

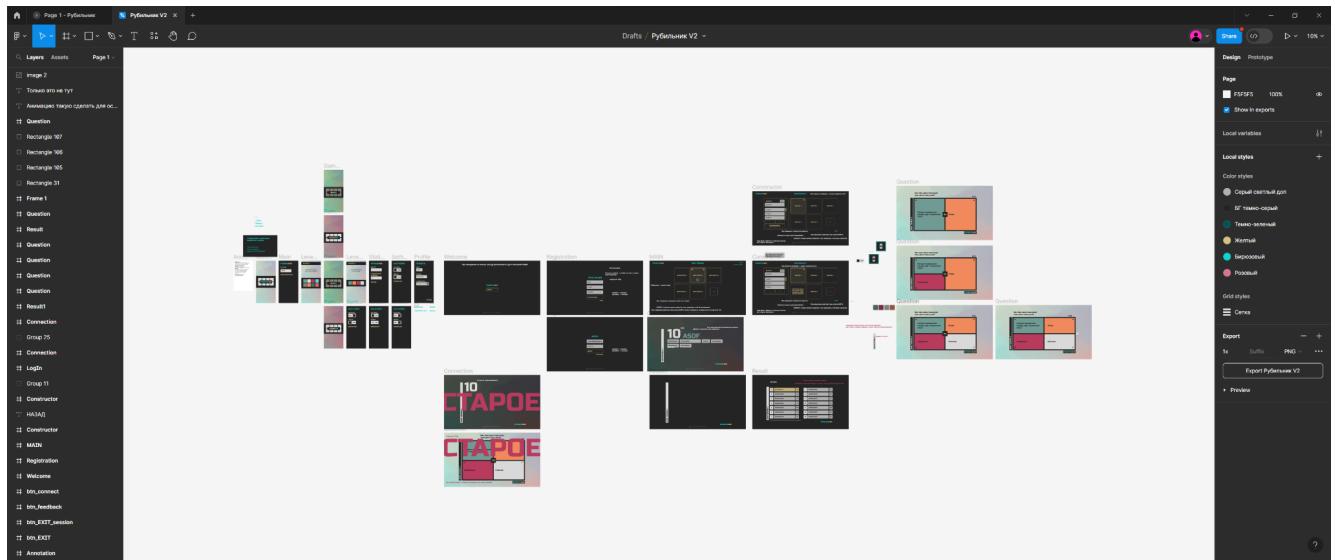


Рисунок 1.5.2 Дизайн-проект в Figma

Документирование результатов: дизайн-проект смотреть приложение А.

Глава 2. Разработка

2.1 Сервер

На сервере для взаимодействия с данными из БД были разработаны функции, такие как:

- updateUsersQuizzes
- deleteAllUsers
- createUser
- deleteUserById
- getUser

Листинг 2.1.1 (пример)

```
export async function createUser(id, {name, email, password}, quizzes) {
  var find = await prisma.user.findUnique({
    where: {
      email: email
    }
  })
  if (find) {
    return null
  } else {
    var result = await prisma.user.create({
      data: {
        id,
        email,
        name,
        password,
        quizzes,
      },
    });
    return result
  }
}
```

Данный код является частью функции в приложении Node.js, которое использует инфраструктуру Prisma для взаимодействия с базой данных. Функция

отвечает за создание нового пользователя в базе данных, если предоставленный адрес электронной почты еще не используется.

Код начинается с импорта класса PrismaClient из модуля '@prisma/client'. Этот класс предоставляет API для взаимодействия с базой данных.

Далее создается новый экземпляр класса PrismaClient. Используется для взаимодействия с базой данных в остальной части функции.

Затем функция определяет новую переменную 'quizzes' и присваивает ей пустой массив.

Затем код проверяет, используется ли уже предоставленный адрес электронной почты, запрашивая базу данных с помощью метода findUnique экземпляра PrismaClient. Если адрес электронной почты уже используется, функция возвращает null, указывая на то, что пользователь не может быть создан.

Если адрес электронной почты не используется, функция создает новую запись пользователя в базе данных с помощью метода create экземпляра PrismaClient. Параметр data метода create содержит объект, который содержит свойства пользователя, включая его идентификатор, адрес электронной почты, имя, пароль.

Для выдачи пользователям простого уникального ID был разработан алгоритм дерева, где в узлах – использованные символы в ID, каждая полная ветка дерева – полная строка используемого ID.

Функции взаимодействия:

- getFreeId()

является рекурсивной функцией, которая используется для получения свободного идентификатора из дерева идентификаторов. Она начинается с установки некоторых переменных, таких как idChars (массив разрешенных символов для идентификаторов), idLength (длина идентификаторов), lvl (текущий уровень рекурсии), freeId (массив для хранения сгенерированных идентификаторов) и isFound (логическое значение для отслеживания того, был ли найден свободный идентификатор).

Затем функция переходит в цикл, который продолжается до тех пор, пока не будет найден свободный идентификатор или пока не будут проверены все возможные идентификаторы.

Функция возвращает сгенерированный идентификатор, соединяя элементы массива freeId.

- `deleteId(idString)`

функция принимает `idString` в качестве аргумента и пытается удалить `id` из дерева.

Код начинается с вызова `checkIdString`, чтобы убедиться, что `idString` является допустимой строкой. Если `id` недействителен, функция возвращает ошибку.

Если `id` действителен, функция запускается с создания переменной `visitor`, указывающей на корень дерева. Затем он перебирает каждый символ в `idString`, начиная с крайнего правого символа. Для каждого символа он ищет узел с таким же значением в дочерних элементах текущего узла. В конце ветки, если совпадение найдено, функция проверяет, совпадают ли оставшиеся символы в `idString`. Если это так, функция удаляет совпадающий узел из дочерних узлов текущего узла.

- `pushId(idString)`

является частью функции `pushId` в классе `IdTree`. Эта функция принимает `idString` в качестве аргумента и проверяет, соответствует ли она требованиям (является строкой правильной длины и содержит только разрешенные символы). Если `id` является допустимым, функция вызывает функцию `_pushId` с `idArray` в качестве аргумента.

Функция `pushId` возвращает результат функции `_pushId`, который может быть сообщением об ошибке или успешном выполнении.

- `visualize()`

визуализация текущего состояния дерева в консоли

Для взаимодействия с клиентами реализованы endpoint'ы для HTTP:

- app.post('/getUser',(req,res)
- app.post('/deleteUser',(req,res)
- app.post('/usersQuizzes',(req,res)
- app.post('/user',(req,res)

Листинг 2.1.2

```
app.post('/user', (req, res) => {
  reqNotifier(req);
  if (doesJsonHave(req.body, handleMissingProperties,
    'name', 'email', 'password')) {
    try {
      var userId = userIds.getFreeId();
      if (userId) {
        createUser(userId, {...req.body}, []).then(result=>{
          if (result) res.status(200).json({id: userId, msg: 'Success'})
          else res.status(400).json({msg:'Failed to create'})
        })
      } else{
        res.status(500).json({msg:"Failed to create: ran out of id's"})
      }
    } catch (error) {
      res.status(500).json({msg:"Server error"})
    }
  }
})
```

и endpoint'ы для SocketIo:

- io.on('connection', (socket)

при получении сигнала о подключении пользователя выводит данные в консоль

- socket.on('bark', (data)

при получении сигнала от пользователя перенаправляет сигнал всем пользователям, подключенным к той же комнате, что и отправитель

- socket.on('join', (data)

при получении сигнала о присоединении игрока к комнате сокет-соединение добавляется в запрашиваемую комнату, выявляется и присваивается сокету (пользователю) уникальный ID, который отправляется по одноименному возвратному сигналу.

- socket.on('create', (data)

при получении сигнала о создании комнаты отправляет возвратный одноименный сигнал об успехе\неудачи.

- socket.on('start', (data)

при получении сигнала начала викторины отправляет одноименные сигналы всем пользователям в комнате.

- socket.on('choice', (data)

при получении данных о выборе пользователем ответа, отправляет одноименный сигнал хосту комнаты

- socket.on('next', (data)

при получении сигнала переключения на следующий вопрос викторины отправляет одноименные сигналы всем пользователям в комнате

- socket.on('end', (data)

при получении сигнала завершения викторины отправляет одноименные сигналы всем пользователям в комнате.

- socket.on('reveal', (data)

при получении сигнала об отображении правильного ответа на вопрос отправляет одноименные сигналы всем пользователям в комнате.

- socket.on('disconnect', ()

при получении сигнала выхода пользователя отправляет одноименные сигналы всем пользователям в той же комнате.

Листинг 2.1.3 (пример):

```
socket.on('join', (data) => {
  if (io.sockets.adapter.rooms.has(data.roomId)) {
    if (data.userId) {
      console.log(`join received from user ${data.userId} ${data.userName}`)
```

```

to room: ${data.roomId}`);
    socket.join(data.roomId)
    io.to(data.roomId).emit('join', {userName: data.userName, userId:
data.userId})
} else {
    console.log(`join received from player ${data.userName} to room:
${data.roomId}`);
    var playerId = playerIds.getFreeId();
    if (playerId) {
        socketsData[socket.id] = {};
        socketsData[socket.id].userId = playerId;
        socketsData[socket.id].userName = data.userName;
        socketsData[socket.id].roomId = data.roomId;
        console.log(`player's id assigned ${playerId} ${data.userName}`);
        socket.join(data.roomId)
        socket.emit('joined', {playerId: playerId})
        io.to(data.roomId).emit('join', {userName: data.userName, userId:
playerId})
    } else { console.log('failed'); }
}
} else {
    console.log(`${data.userName} failed to join. Room ${data.roomId} does
not exist`);
    socket.emit('joined', {})
}
);

```

2.2 Веб-сайт

Веб-сайт был собран сборщиком Vite и разработан на React.JS функциональных компонентах. Для routing`а использовался react-router.

Схема routing`а и рендеринга компонентов:

Листинг 2.2.1 - фрагмент файла компонента main.jsx

```
...
  {path: "/",
    element: <App />
  }, {path: "/editor",
    element: <Editor />
  }, {path: "/rooms",
    element: <Rooms />
  }, {path: "/login",
    element: <Login />
  }, {path: "/register",
    element: <Register />
  }, {path: "/debug",
    element: <Debug />
  }, {path: "/play",
    element: <SockerWrapper />}, ...
```

Для каждого изменения пути пользователя рендерится отдельный функциональный компонент, который имеет подобную структуру:

Листинг 2.2.2 - фрагмент файла функционального компонента App.jsx

```
import './App.css'

function App() {
  return (
    <>
    <h1>РУБИЛЬ<span style={{color: "#D6BF81" }}>НИК</span></h1>
    ...
  </>
)
}

export default App
```

Логика функциональных компонентов реализована с помощью React “хуков” – useState и useEffect, они нужны для сохранения “состояния” и динамической работы приложения. Пример использования useState и useEffect:

Листинг 2.2.3 - фрагмент файла компонента Login.jsx

```
...
const Login = () => {
    ...
    const [email, setEmail] = useState(null)
    const [pass, setPass] = useState(null)
    const [error, setError] = useState(false)
    ...

    useEffect(() => {
        if (!email || !pass) {
            setError("ЗАПОЛНИТЕ ВСЕ ПОЛЯ")
        } else if (pass.length < 8) {
            setError("МИНИМУМ 8 СИМВОЛОВ")
        } else if (!email.match(regexp)) {
            setError("НЕСУЩЕСТВУЮЩИЙ E-MAIL")
        } else {
            setError(false)
        }
    }, [email, pass])
...
}
```

Пример выше иллюстрирует использование состояний (useState) для почты, пароля, и текста ошибки. Состояния нужны для динамической обработки интерфейса. useEffect - react “хук”, позволяющий при изменении состояний (в данном случае он зависит от изменения состояний почты email и пароля pass) выполнять некоторый код. В случае с компонентом Login, представленным выше, логика такова:

Изначальные состояния почты (email), пароля (pass) и ошибки (error) - нулевые.

При любом изменении email и pass (ввод через форму входа изменяет данные состояния) запускается участок кода внутри useEffect, который смотрит, удовлетворяют ли email и pass условиям. Если удовлетворяют, то ничего не происходит. В случае, когда одно из условий нарушается, в состояние error

подается строка ошибки. В свою очередь, при изменении состояния error, эта строка выводится пользователю. Состояния и useEffect позволяют выполнять это динамически, благодаря чему достигается плавное динамическая работа интерфейса веб-страницы. В каждом из описанных ниже функциональных компонентов основополагающим является использование динамических состояний useState и useEffect для формирования логики интерфеса при изменения состояний.

На страницах “/editor”, “/play” и “/rooms” использовалась множественная вложенность компонентов, схема вложенности компонентов:

- Editor
 - QuestionEdit
 - QuestionWrapper
 - QuestionViewet
 - EditingQuestion
- SockerWrapper
 - Endgame
 - Game
 - Lobby
- Rooms
 - QuizCard

Каждый функциональный компонент описывался в своем отдельном файле разрешения .jsx. Хранение файлов компонентов осуществлялось в папке “src”.

Список всех использованных функциональных компонентов и их описание:

- main – основной компонент, монтирующийся в index.html файл, содержит реализацию routing`а .
- App – приветственный компонент главной страницы с кнопкой, перенаправляющей на страницу “/login” .
- Debug – компонент выводом функций дебаггинга.

- Login – содержит форму для входа пользователей и отправляет данные входа на back-сервер.
- Register – содержит форму для регистрации пользователей и отправляет данные входа на back-сервер.
- Rooms – главная страница пользователя, содержит кнопки входа-выхода и включает в себя функционал вывода созданных пользователем викторин, полученных запросом с сервера (на сервер из БД) в виде компонентов QuizCard.
- QuizCard – оберточный компонент викторины, содержит название викторины, кнопки редактирования, удаления, запуска викторины.
- SockerWrapper – оберточный компонент, создает websocket канал связи через библиотеку socket.io и передает его дочерним компонентам - Lobby, Game, Endgame, реализует комнату викторины и логику.
- Lobby – компонент, получающий подключенный socket, и выводящий имена подключенных пользователей, кол-во подключенных пользователей, код подключения к комнате, название викторины, кнопку начала викторины.
- Game – компонент, выводящий текущий вопрос запущенной викторины (с первого по последний), кол-во вопросов, текст текущего вопроса, варианты ответа на текущий вопрос, кнопку, показывающей на 4 секунды правильный ответ на текущий вопрос, отправляющей соответствующие сообщения через websocket, переходящей к следующему вопросу. Обновляет внутренние данные на каждый вопрос.
- Endgame – компонент, появляющийся после конца игры, выводящий список подключенных пользователей, кол-во набранных ими очков, их место в рейтинговой таблице текущей игры.

- Editor – компонент, используемый для редактирования текущей викторины и создания новых, является оберткой компонентов EditingQuestion, QuestionWrapper.
- EditingQuestion – компонент, осуществляющий функционал создания нового вопроса викторины. Реализует функционал ввода текста вопроса, выбора кол-ва вариантов ответа, ввода текстов вариантов ответов, выбор верного варианта ответа, сохранения вопроса в редактируемую\создаваемую викторину.
- QuestionWrapper – оберточный компонент, выводящий сетку из компонентов QuestionViewer на каждый вопрос текущей викторины.
- QuestionViewer – компонент, являющийся выводом вопроса викторины. Содержит кнопки удаления вопроса и редактирования вопроса, при редактировании вопроса выводится компонент QuestionEdit.
- QuestionEdit – компонент, содержащий ввод текста вопроса, со значением текста текущего редактируемого вопроса, ввод текстов вариантов ответа, со значениями текстов ответов текущего редактируемого вопроса, кнопками выбора правильного ответа, кнопку сохранения изменений в текущем выбранном вопросе выбранной викторины.

Логика использования routing`а и компонентов потенциальным пользователем:

Изначально пользователь попадает на главную страницу (путь “/”), на которой отображается компонент App и имеет кнопку перехода на страницу входа с путем “/login”.

На странице входа “/login” рендерится компонент Login, пользователь может ввести данные для входа и произвести вход (при успешном ответе сервера на запрос входа) или перейти на страницу регистрации с путем “/register”.

На страница регистрации “/register” рендерится компонент Register, пользователь может заполнить форму для регистрации аккаунта, и его хешированные данные отправятся на сервер. При успешном ответе сервера производится вход и перенаправление на страницу “/rooms”. Также есть возможность вернуться на страницу входа.

Страница “/rooms” является главной страницей пользователя. На ней отображается компонент Rooms с его наследуемыми компонентами. при переходе на нее пользователь с сервера (на сервер – с БД) список сохраненным за ним викторин. Имеет возможность создания новой викторины, запуска викторины как комнату для игры, удаления или редактирования существующей викторины. Любое локальное изменение в викторинах пользователя вызывает отправку локальных викторин на сервер, и, соответственно, сохраняет все изменения в БД. Для изменения или создания новой викторины пользователь перенаправляется на страницу редактирования “/editor”. Для запуска викторины как комнаты для игры пользователь перенаправляется на страницу “/play”.

Страница “/editor” является редактором, который используется и при редактировании существующей викторины, и при создании новой. Отображает компонент Editor. При создании новой викторины данные на странице пустые, пользователь создает викторину с нуля (На момент разработки проекта, при создании викторины автоматически добавлялись четыре тестовых вопроса, упрощающие процесс создания викторины для отладки работы веб-модуля). При редактировании существующей викторины, на компонент подаются данные выбранной для редактирования викторины, который пользователь может изменять также, как и при создании викторины. Страница имеет функционал создания нового вопроса, редактирования созданных вопросов викторины и удаления вопросов. В создании нового вопроса есть возможность указать текст вопроса, кол-во вариантов ответов, тексты вариантов ответов и выбор правильного варианта ответа. При сохранении созданной или редактируемой викторины, пользователь перенаправляется на страницу “/rooms”.

Страница “/play” отображает компонент SockerWrapper, который является оберткой компонентов Lobby, Game и Endgame. Оберточный компонент нужен для установки, сохранения и передачи между компонентами канала связи websocket. Изначально отображается подкомпонент Lobby, на котором видно кол-во подключенных пользователей, их имена, код подключения формата “AAAA”, который пользователи вводят через мобильное приложение и подключаются к викторине. Нажатие кнопки старта переводит игру из состояния ожидания в активное состояние, вместо компонента Lobby отображается компонент Game. На нем отображен текущий вопрос (с первого по последний). Конкретнее - текст данного вопроса, варианты ответов, название викторины и кнопка, нажатие которой останавливает регистрацию ответов пользователей на данный вопрос и отображает правильный ответ на 4 секунды, после этого подкомпонент Game ре-рендерится со следующим вопросом. Когда компонент отображает последний в викторине вопрос, нажатие кнопки отображает правильный ответ последнего вопроса на 4 секунды, затем вместо компонента Game отображается компонент Endgame, и игра переходит из активного состояния, в состояние завершения. Компонент Endgame отображает список пользователей, который содержит имена пользователей, кол-во заработанных ими очков (правильно выбранный ответ дает 50 очков пользователю) и их место в рейтинговой таблице данной игры. После этого пользователь может вернуться на главную страницу пользователя “/rooms” и игра считается завершенной.

Стилизация интерфейса была осуществлена с помощью css, для редактирования расположения компонентов и частей компонентов использовался grid и flexbox. Пример использования css классов для изменения стилей компонентов:

Листинг 2.2.4 - фрагмент файла компонента Rooms.jsx

```
...
<div className="button_plus_container">
...
</div>
```

```
...
```

Листинг 2.2.5 - фрагмент файла стилей App.css

```
...
.button_plus_container {
  width: 300px;
  height: 300px;
  display: flex;
  justify-content: center;
  align-items: center;
}
...
```

2.3 Android

Для разработки мобильного приложения использовалась среда Android Studio на языке программирования Java.

Сначала был перенесен дизайн-проект. Были прописаны layout'ы и фрагменты, элементы (button, textedit, textview, switch и др.).

Были описаны 2 activity по несколько фрагментов, заменяющих друг друга при событиях, в каждом:

- MainActivity – отвечает за основное состояние приложения до подключения сокет-соединения
 - MainFragment – главная страница приложения;
 - SettingsFragment – страница настроек приложения.
- QuizActivity – отвечает за состояние приложения во время установленного сокет-соединения при завершении activity прерывает соединение по сокету.
 - WaitingFragment – страница комнаты в ожидании начала викторины;
 - QuestionFragment – динамическая страница, отображающая вопрос, полученный от хоста викторины;
 - ResultFragment – страница, отображающая результат пользователя в баллах и место среди других участников викторины.

Реализованы обработчики событий SocketIo в MainActivity:

- mSocket.on(Socket.EVENT_CONNECT, args -> (обработчик события успешного установления сокет-соединения);
- mSocket.on(Socket.EVENT_CONNECT_ERROR, args -> (обработчик события провального установления сокет-соединения);
- mSocket.on("joined", args -> (при получении сигнала от сервера об успешном подключении сокета данного пользователя к выбранной комнате запускает QuizActivity, при получении сигнала от сервера о провале

подключения сокета данного пользователя к выбранной комнате отключает сокет-соединение).

Реализованы обработчики событий SocketIo в QuizActivity:

- MainActivity.mSocket.on("join", onJoin) – подключения других пользователей к комнате;
- MainActivity.mSocket.on("leave", onLeave) – отключения других пользователей от комнаты;
- MainActivity.mSocket.on("start", onStart) – старта викторины хостом комнаты;
- MainActivity.mSocket.on("next", onNext) – подключения других пользователей к комнате;
- MainActivity.mSocket.on("choice", onChoice) – выбора пользователями варианта ответа на вопрос из предложенных;
- MainActivity.mSocket.on("reveal", onReveal) – раскрытия правильного ответа;
- MainActivity.mSocket.on("end", onEnd) – завершения викторины;
- MainActivity.mSocket.on("bark", onBark) – получения сигнала от другого пользователя в комнате;

ЗАКЛЮЧЕНИЕ

Проект прошел через этапы: планирование, проектирование, разработка, тестирование и отладка, корректировка. Были созданы: логика сервиса, дизайн-проект, серверный модуль, веб-модуль, android-модуль.

В “Рубильнике” были успешно реализованы:

- Создание и проведение интерактивных викторин в реальном времени и участие в них.
- Решение проблемы отсутствия качественных онлайн викторин на отечественном рынке.
- Удобный инструментарий для проведения онлайн викторин, содействующий развитию образовательных и развлекательных мероприятий.
- Новые возможности для социального взаимодействия через проведение онлайн викторин.

Проект “Рубильник” был успешно завершен. Минимальные требования (см. пункт 1.3 Планирование) соблюdenы и закончены в срок. Изначально поставленные цели были благополучно достигнуты.

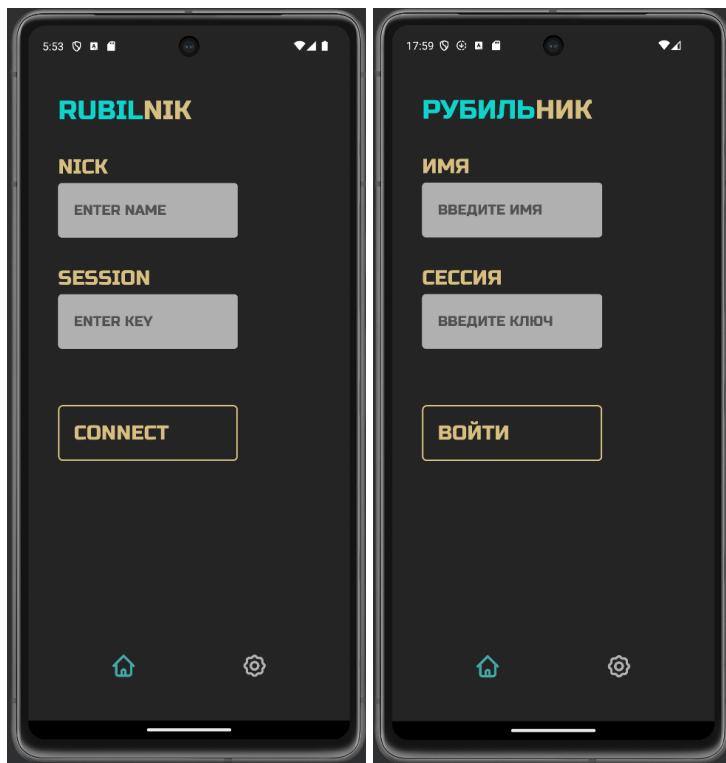
Библиографический список

1. [CryptoJS - CryptoJS \(gitbook.io\)](#) [Электронный ресурс].
2. [Introduction | Socket.IO](#) [Электронный ресурс].
3. [Managing State – React](#) [Электронный ресурс].
4. [Getting Started | Axios Docs](#) [Электронный ресурс].
5. [Documentation | Android Developers](#) [Электронный ресурс].
6. [Express routing \(expressjs.com\)](#) [Электронный ресурс].
7. [Описание программ Adobe](#) [Электронный ресурс].

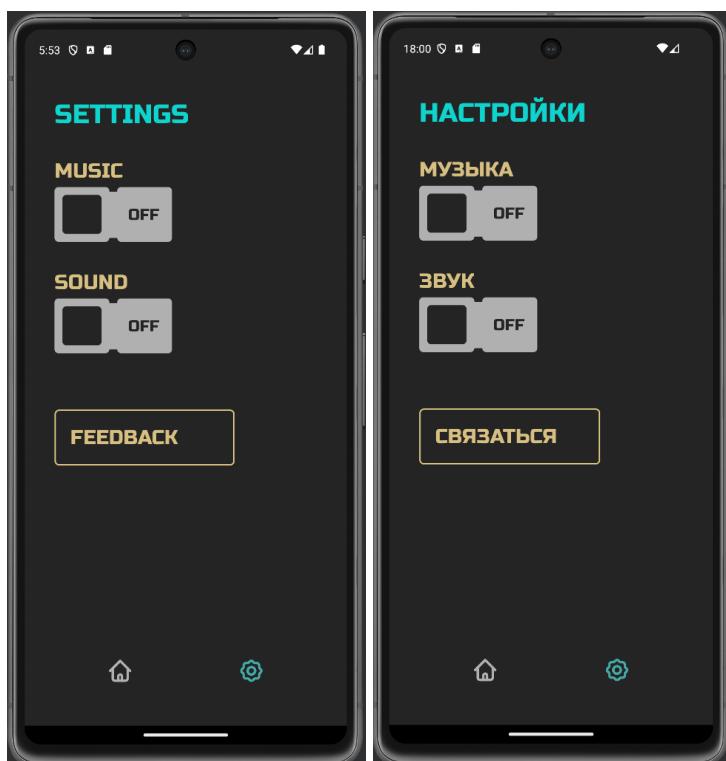
ПРИЛОЖЕНИЯ

Приложение А. Дизайн-проект.

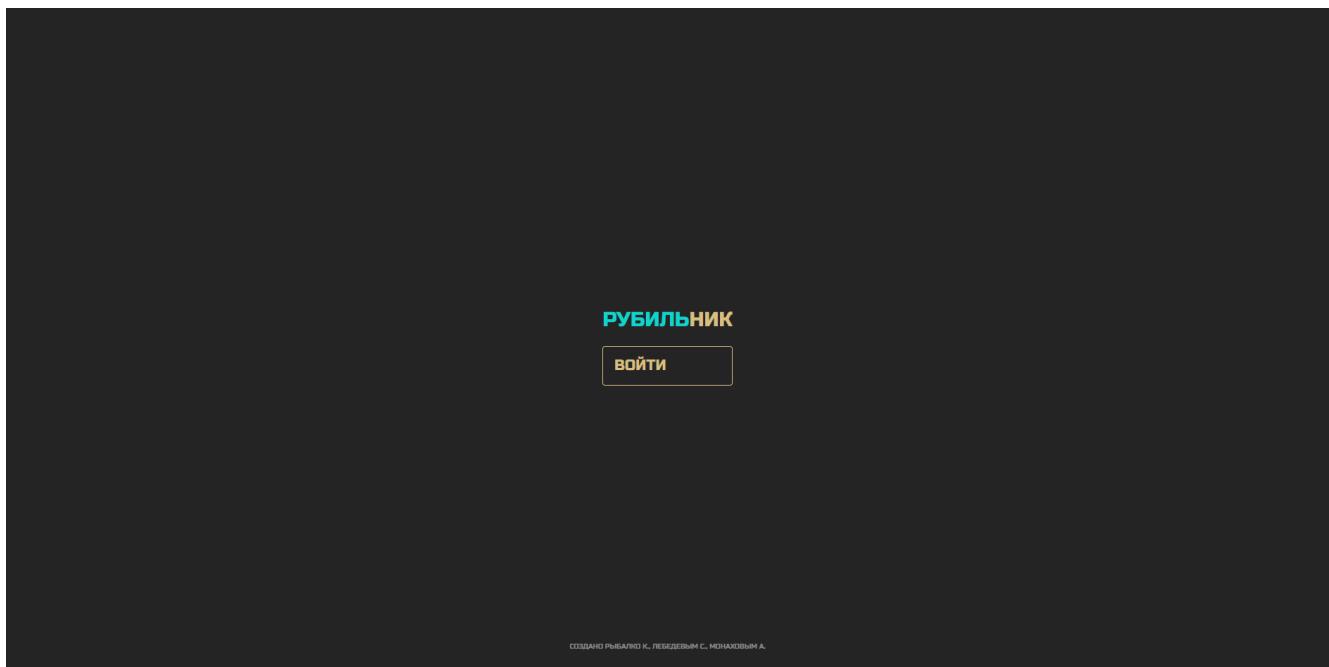
Окно подключения к хосту:



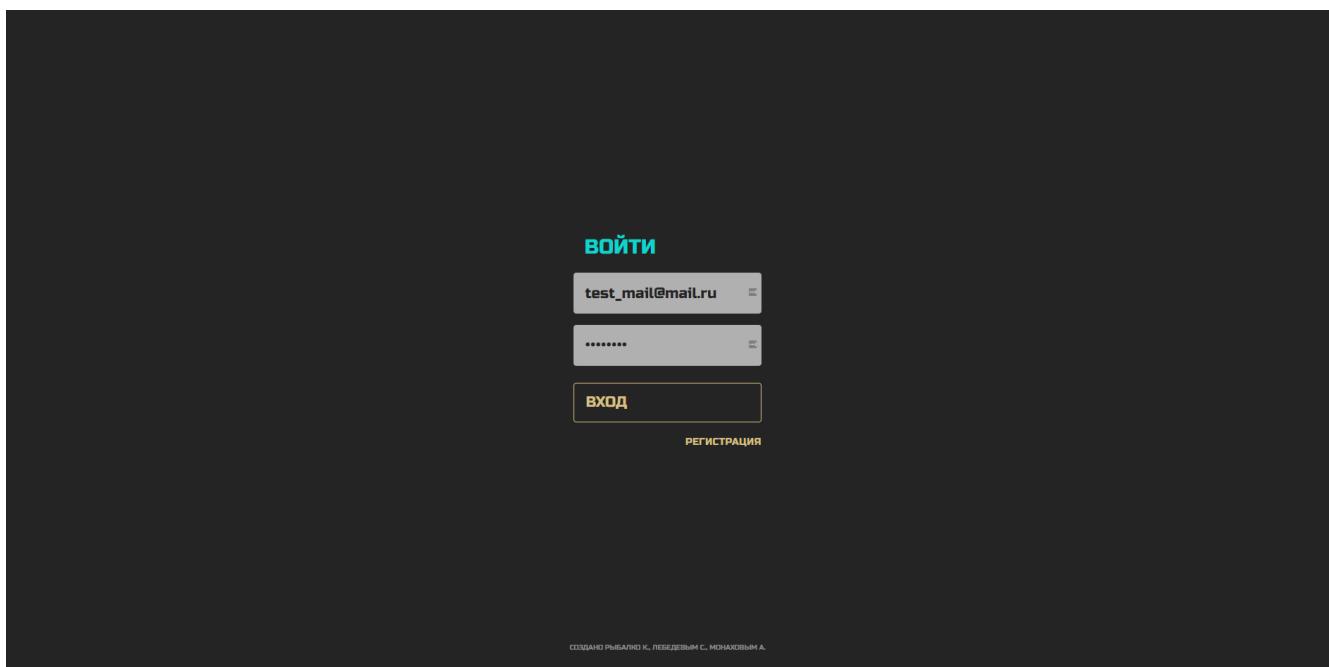
Окно настроек:



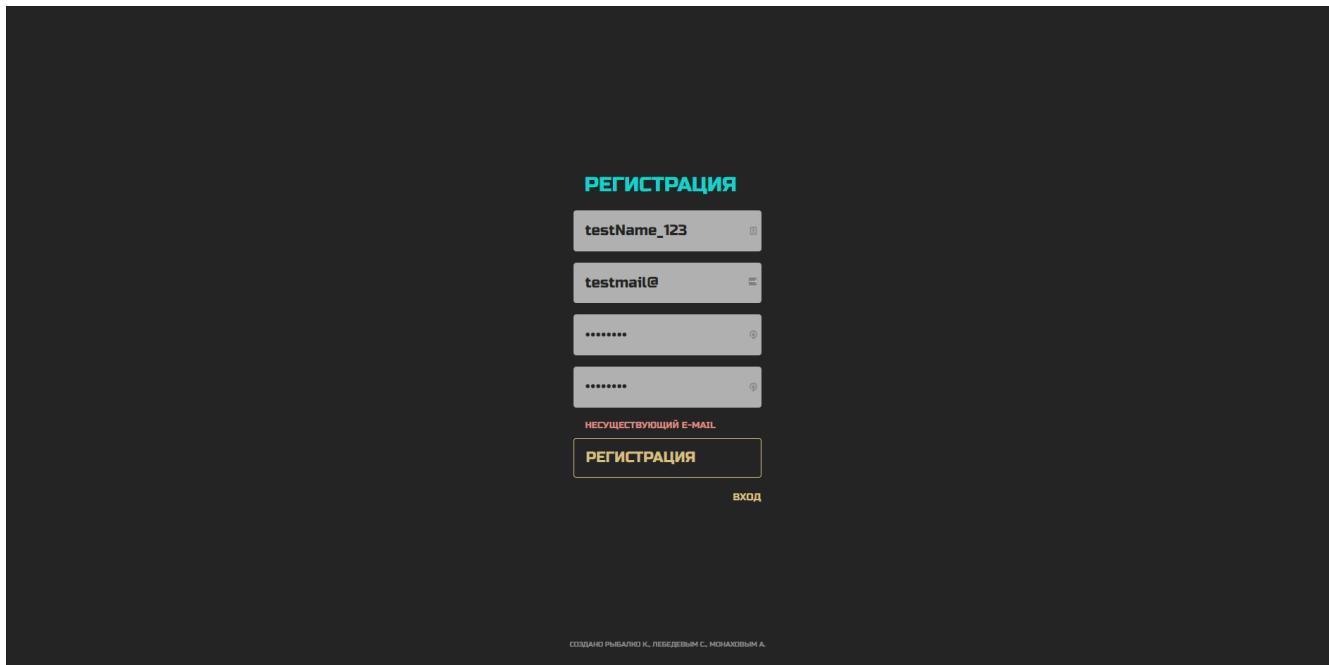
Страница приветствия:



Страница аутентификации:



Страница регистрации:



РЕГИСТРАЦИЯ

testName_123

testmail@

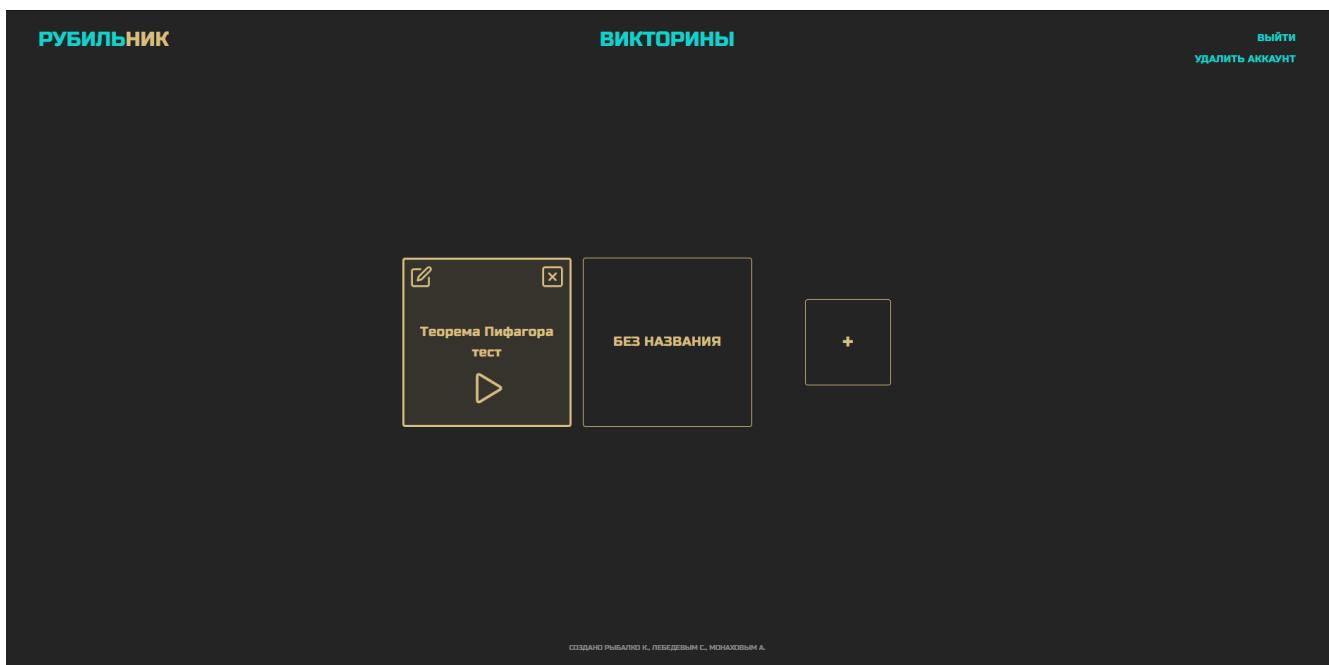
НЕСУЩЕСТВУЮЩИЙ Е-MAIL

РЕГИСТРАЦИЯ

ВХОД

СОЗДАНО РЫБАЛКО К., ЛЕБЕДЕВЫМ С., МОНАХОВЫМ А.

Страница викторин:



РУБИЛЬНИК ВИКТОРИНЫ ВЫЙТИ
УДАЛИТЬ АККАУНТ

Теорема Пифагора тест

БЕЗ НАЗВАНИЯ

+

СОЗДАНО РЫБАЛКО К., ЛЕБЕДЕВЫМ С., МОНАХОВЫМ А.

Страница создания и редактирования викторины:

The screenshot shows a dark-themed web application for creating quizzes. At the top left is the logo 'РУБИЛЬНИК'. In the center, there's a title 'Теорема Пифагора' (Pythagorean theorem). On the left, a sidebar has a heading 'СОЗДАТЬ ВОПРОС' (Create Question) and a dropdown menu with options: 'Кто придумал теорему Пифагора?' (Who invented the Pythagorean theorem?), 'Не пифагор' (Not Pythagoras), and 'Пифагор' (Pythagoras). Below this is a section 'ВАРИАНТ ОТВЕТА' (Answer Variant) with a note 'Заполните все поля!' (Fill all fields!) and two buttons: '-' and '+'. A 'Добавить вопрос' (Add question) button is at the bottom. To the right, there are three question cards: 'Назовите столицу Франции' (Name the capital of France) with answer 'Париж' (Paris); 'Чему равно $2+2$?' (What is $2+2$ equal to?) with answer 'Четыре' (Four); and a third card with a question 'Сколько лучше варить яйцо вкрутую?' (How much better to boil an egg? - a riddle) with answer 'Лучше' (Better). A 'Сохранить' (Save) button is at the bottom right of the card area. At the very bottom, it says 'СОЗДАНО РЫБАЛКО К., ЛЕБЕДЕВЫМ С., МОНАХОВЫМ А.'.

Страница ожидания викторины:

The screenshot shows a landing page for a quiz. It features a large number '2' and the text 'AAAAA - код подключения' (AAAAA - connection code). Below this are two buttons: 'MOBILE1' and 'MOBILE2'. On the left, a vertical bar has the text 'ВИКТОРИНА TEST'. At the bottom is a 'СТАРТ' (Start) button. To the right, there's a smartphone icon with the text 'Room AAAA' above it and a small screen icon below it.

Страницы вопросов:

Смартфон показывает мобильную версию интерфейса игры. Виджет на экране смартфона отображает вопрос: «Назовите столицу Франции» и 4 варианта ответа: А - Париж, Б - Токио, В - Москва. Виджет внизу имеет кнопку «РУБИЛЬНИК». Слева на смартфоне виден текст «ВИКТОРИНА TEST».

Назовите столицу Франции

1/4

ВИКТОРИНА TEST

А Париж

Б Токио

В Москва

Г

РУБИЛЬНИК

QUESTION 1

Назовите столицу Франции

A B C

Смартфон показывает мобильную версию интерфейса игры. Виджет на экране смартфона отображает вопрос: «Чему равно $2+2$?» и 4 варианта ответа: А - четыре, Б - four, В - 4, Г - Все перечисленное выше. Виджет внизу имеет кнопку «РУБИЛЬНИК». Слева на смартфоне виден текст «ВИКТОРИНА TEST».

Чему равно $2+2$?

2/4

ВИКТОРИНА TEST

А четыре

Б four

В 4

Г Все перечисленное выше

Г

РУБИЛЬНИК

QUESTION 2

Чему равно $2+2$?

A B C D

Страница итога викторины:

