

Министерство образования Республики Беларусь  
Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей  
Кафедра информатики  
Дисциплина «Основы алгоритмизации и программирования»

«К защите допустить»  
Руководитель курсового проекта  
преподаватель  
\_\_\_\_\_ Е.Г. Крупенич  
\_\_\_\_\_.\_\_\_\_\_.2025

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
к курсовому проекту  
на тему:  
**«ИГРА "PIXEL WARS"»**

БГУИР КП 6-05-0612-02 025 ПЗ

Выполнил студент группы  
453501  
СКРОБОТ Денис Алексеевич

\_\_\_\_\_  
(подпись студента)

Курсовой проект представлен  
на проверку \_\_\_\_\_.\_\_\_\_\_.2025

\_\_\_\_\_  
(подпись студента)

Минск 2025

# СОДЕРЖАНИЕ

Введение . . . . .	4
1 Обзор существующих аналогов . . . . .	5
1.1 Оригинальный r/Place . . . . .	5
1.2 PixelPlace.io . . . . .	6
1.3 PixelCanvas.io . . . . .	7
2 Архитектура приложения . . . . .	9
3 Технологии программирования, используемые для решения поставленных задач . . . . .	10
3.1 Язык программирования Java . . . . .	10
3.2 Фреймворк Spring . . . . .	11
3.3 База данных PostgreSQL . . . . .	12
3.4 Google OAuth 2.0 API . . . . .	13
3.5 Интерфейс . . . . .	14
3.6 Среда разработки IntelliJ IDEA . . . . .	15
3.7 Среда разработки Visual Studio Code . . . . .	16
3.8 Хостинг и развёртывание приложения . . . . .	18
4 Технические особенности реализации . . . . .	20
4.1 Обновление холста . . . . .	20
4.2 Установка пикселя на холст . . . . .	21
4.3 Система модерации и панель администратора . . . . .	22
5 Тестирование . . . . .	24
Заключение . . . . .	26
Список использованных источников . . . . .	27
Приложение А (обязательное) Листинг программного кода . . . . .	28

## **ВВЕДЕНИЕ**

В последние десятилетия концепция «пользовательского контента» приобрела широкое распространение, а с развитием интернет-технологий появились проекты, в которых пользователи могут активно взаимодействовать друг с другом в рамках виртуальных платформ. Одним из ярких примеров таких проектов является r/Place — социальный эксперимент, проведённый на платформе Reddit, в котором пользователи могли размещать пиксели на общей картине, создавая таким образом коллективное произведение искусства. Этот проект стал не только местом для творческого самовыражения, но и уникальной возможностью для наблюдения за коллективным поведением и взаимодействием пользователей в реальном времени. Вдохновленная этим экспериментом, игра Pixel Wars представляет собой аналог r/Place, но с усовершенствованиями в механике взаимодействия, такие как система рейтингов и должностей.

За последние несколько лет разработка многопользовательских онлайн-игр и платформ, которые обеспечивают взаимодействие пользователей в реальном времени, стала важной частью исследований в области геймдизайна и программирования.

Целью данной курсовой работы является разработка многопользовательской игры Pixel Wars, аналогичной r/Place, с уникальной игровой механикой и интерфейсом, предоставляющим пользователю возможность взаимодействовать с другими игроками в реальном времени. В процессе работы будут исследованы аспекты проектирования интерфейсов, работы с базами данных, а также реализована система взаимодействия между игроками, поддерживающая динамичные изменения в игровом мире.

При проектировании игры использовались принципы модульности, гибкости интерфейса и оптимизации работы серверной части приложения. Основное внимание было уделено разработке механики взаимодействия пользователей с картой.

Для достижения данной цели предусмотрены следующие задачи:

- Анализ аналогов
- Проектирование архитектуры
- Реализация серверной части
- Разработка пользовательского интерфейса
- Введение системы модерации
- Тестирование

# 1 ОБЗОР СУЩЕСТВУЮЩИХ АНАЛОГОВ

## 1.1 Оригинальный r/Place

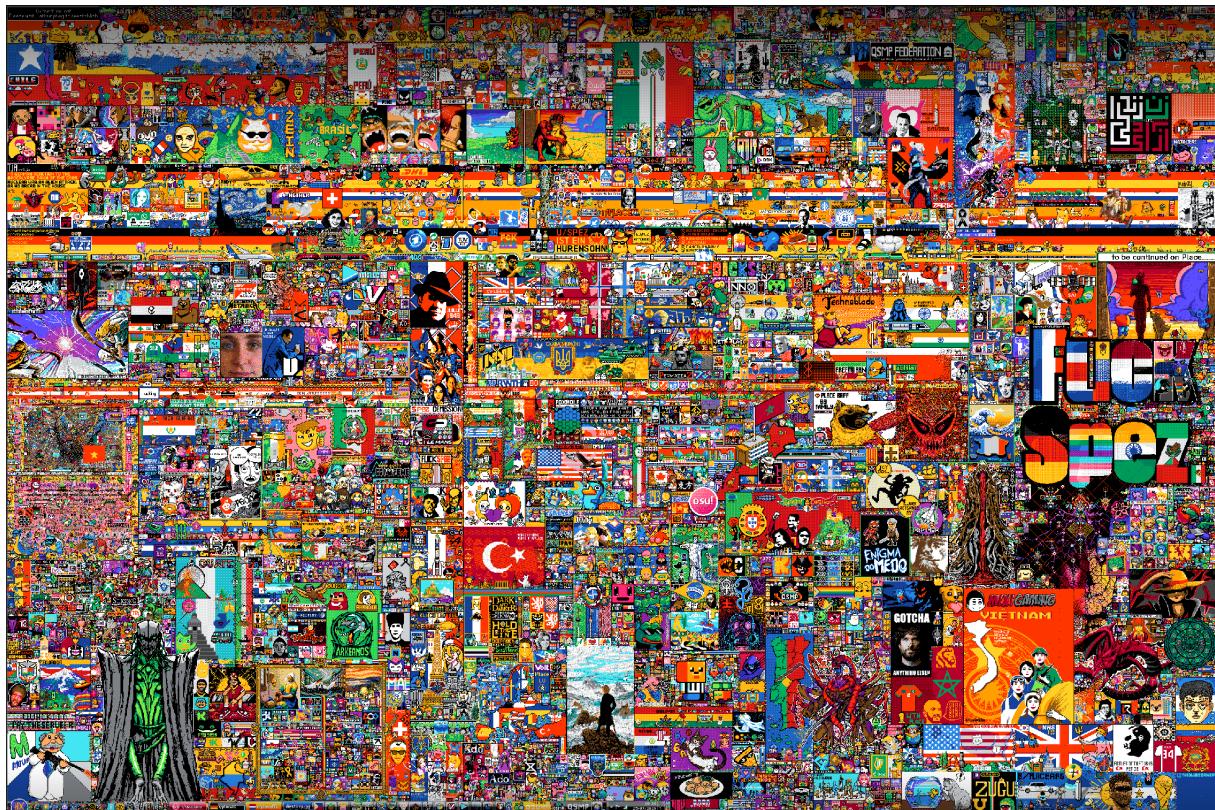


Рисунок 1 – Финальный холст r/Place 2023

r/Place — это социальный эксперимент, проведённый платформой Reddit в 2017 году, который позволил пользователям размещать пиксели на огромной сетке размером 1000x1000 пикселей [1]. Каждому участнику предоставлялся ограниченный выбор цветов для одного пикселя, который можно было разместить на картине. Однако, через некоторое время, любой другой пользователь мог изменить или удалить размещённый пиксель. Это создавалось условия для динамичного взаимодействия, конкуренции и коллективного творчества, так как участники должны были защищать свои изображения от «вандалов» и сражаться за территорию на картине.

Основной целью r/Place было изучение коллективного поведения и взаимодействия пользователей в условиях ограниченных ресурсов. Проект позволил наблюдать, как участники организуют коллективные усилия для создания и сохранения изображений, а также как возникают временные альянсы и конфликты. Созданная карта стала живым примером коллективного разума, где каждый пиксель был важен для общего результата.

Социальный эксперимент стал значимой вехой в изучении цифровых сообществ, показав, как большое количество людей может взаимодействовать в реальном времени, создавая совместное произведение искусства. В 2022 году был проведён повторный запуск r/Place, который использовал улучшенную механику и привлек ещё больше участников.

## 1.2 PixelPlace.io



Рисунок 2 – Фрагмент холста и интерфейса PixelPlace.io

PixelPlace.io — это многопользовательская онлайн-игра, вдохновлённая концепцией r/Place, в которой пользователи могут размещать пиксели на общем холсте в реальном времени [2]. В отличие от оригинального проекта Reddit, данное приложение включает элементы геймификации и более развитую социальную инфраструктуру.

На холсте PixelPlace.io пользователи могут рисовать изображения, координируясь с другими игроками через встроенный чат. Уникальной особенностью платформы является наличие системы прокачки: за размещение пикселей игрок получает опыт, увеличивает уровень и может открывать новые возможности. Интерфейс позволяет отслеживать свою статистику и рейтинг, что стимулирует участие в долгосрочной игровой активности.

Игра предлагает несколько серверов (регионов), каждый из которых имеет свой отдельный холст. Введены механизмы защиты от вандализма — игроки могут "замораживать" пиксели, а также использовать шаблоны, чтобы совместными усилиями воспроизводить

изображения. Администрация поддерживает систему модерации и следит за соблюдением правил.

В результате, PixelPlace.io развивает идеи r/Place, превращая коллективное рисование в полноценную соревновательную игру с элементами RPG, рейтингов и кооперативных миссий. Благодаря своей гибкой системе взаимодействия, прогрессии и визуальной составляющей, этот проект стал заметным явлением в жанре многопользовательских пиксельных игр.

### 1.3 PixelCanvas.io

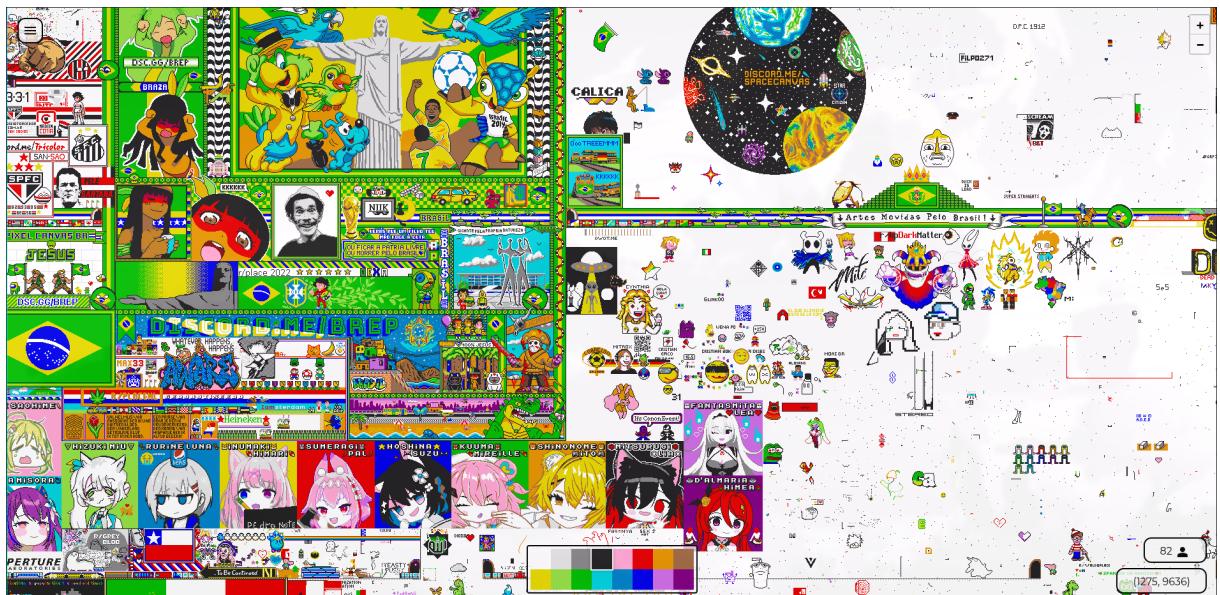


Рисунок 3 – Фрагмент холста PixelCanvas.io

PixelCanvas.io — это онлайн-платформа для коллективного рисования, во многом вдохновлённая оригинальным r/Place, но реализованная в более открытом и гибком формате [3]. Данный проект предоставляет пользователям возможность закрашивать пиксели на практически бесконечном холсте, поддерживая масштабные рисунки и долговременные коллaborации.

Ключевой особенностью PixelCanvas.io является масштабность: пользователи могут свободно перемещаться по огромной карте, создавая изображения в любом месте. Система координат позволяет точно размещать шаблоны, что делает платформу удобной для реализации сложных изображений и коллективных проектов. Каждый пиксель можно установить с заданной задержкой, что формирует основу для соревновательного взаимодействия между группами игроков.

Также в игре поддерживается загрузка и наложение шаблонов изображений, что облегчает организацию крупных проектов.

Пользователи часто координируются через внешние чаты и форумы. Несмотря на отсутствие формальной системы уровней или рейтингов, активные сообщества создают неофициальные альянсы, соревнуются за территорию и организуют защиту своих работ.

## 2 АРХИТЕКТУРА ПРИЛОЖЕНИЯ

Приложение Pixel Wars построено по клиент-серверной архитектуре, что позволяет реализовать многопользовательское взаимодействие в реальном времени. Данная архитектура обеспечивает централизованное управление игрой, синхронизацию состояния холста, а также упрощает реализацию модерации и системы рейтингов.

Приложение состоит из следующих логических компонентов:

**1. Клиентская часть.** Отвечает за отображение интерфейса пользователю, приём его команд (выбор цвета, установка пикселя), а также получение данных с сервера (текущее состояние холста, обновления, сообщения, рейтинг и т.д.). Все действия игрока отправляются на сервер, где проходят проверку и обработку.

**2. Серверная часть.** Выполняет обработку всех действий, поступающих от клиентов. Основные функции сервера:

- обновление состояния общей карты;
- ведение логов, хранение статистики;
- поддержка сессий игроков;
- управление таймером;
- выполнение модераторских команд (изменение статуса пользователя).

**3. База данных.** Сервер взаимодействует с базой данных, в которой хранятся:

- информация о пользователях (логины, пароли, статус);
- текущее состояние холста (цвета и координаты всех пикселей);
- история действий.

### **3 Технологии программирования, используемые для решения поставленных задач**

Здесь мы рассмотрим, какие технологии программирования были выбраны и по какой причине выбор был именно таким.

#### **3.1 Язык программирования Java**

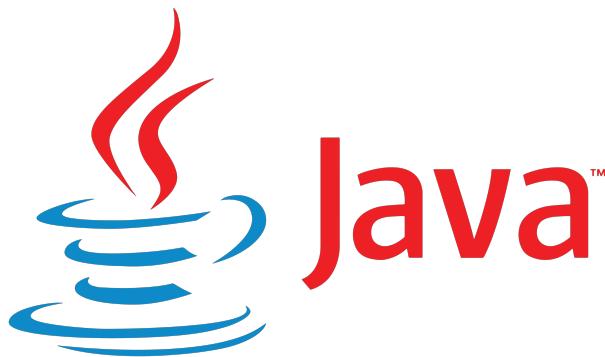


Рисунок 4 – Java

Язык программирования Java является объектно-ориентированным языком, разработанным компанией Sun Microsystems и в настоящее время поддерживаемым корпорацией Oracle. Java обеспечивает высокий уровень абстракции, платформенную независимость и безопасность, что делает его одним из самых популярных языков в мире.

Одним из основных преимуществ Java является его независимость от платформы, достигаемая благодаря использованию виртуальной машины Java (JVM), которая исполняет байт-код, одинаковый для всех операционных систем. Это свойство позволяет создавать переносимые приложения без необходимости модификации исходного кода.

Java поддерживает основные принципы объектно-ориентированного программирования, такие как наследование, инкапсуляция, полиморфизм и абстракция. Эти особенности обеспечивают высокую модульность и повторное использование кода. [4]

В рамках разработки игры Pixel Wars язык Java был выбран как основной инструмент программирования благодаря следующим преимуществам:

- наличие обширной стандартной библиотеки, включающей средства для сетевого взаимодействия, работы с потоками, графическим интерфейсом и структурами данных;
- развитая экосистема, включающая библиотеки и фреймворки, ускоряющие процесс разработки;

- встроенная система управления памятью с автоматическим сборщиком мусора, упрощающая разработку и снижает вероятность ошибок, связанных с управлением ресурсами;
- высокая читаемость и строгая типизация, что способствует написанию надёжного и легко сопровождаемого кода.

В процессе разработки проекта с использованием Java были реализованы следующие компоненты: взаимодействие с серверной частью через RESTful API, обработка пользовательских действий и реализация механизма обновления состояния в реальном времени.

Таким образом, использование языка Java в проекте позволило обеспечить кроссплатформенность, надёжность и структурированность разрабатываемого программного продукта.

### 3.2 Фреймворк Spring



Рисунок 5 – Spring Framework

Spring — это мощный и гибкий фреймворк с открытым исходным кодом для разработки корпоративных и веб-приложений на языке Java. Благодаря своей модульной структуре, высокой производительности и удобству масштабирования, Spring широко применяется при создании распределённых и высоконагруженных систем, включая веб-сервисы и многопользовательские онлайн-платформы.

Одной из ключевых особенностей Spring является внедрение зависимостей (Dependency Injection), которое способствует слабой связанности компонентов приложения и облегчает их тестирование, повторное использование и расширение. Фреймворк построен по принципу инверсии управления (Inversion of Control), где управление жизненным циклом объектов осуществляется специальным контейнером.

В проекте Pixel Wars фреймворк Spring используется для реализации серверной части приложения, обеспечивающей работу с пользователями, обработку запросов, взаимодействие с базой данных и реализацию защитных механизмов.

В процессе разработки были использованы следующие модули Spring:

- Spring Boot — модуль, предназначенный для упрощения конфигурации и быстрой инициализации проекта. Он предоставляет встроенный сервер и автоматическую настройку компонентов, что позволило сосредоточиться на реализации логики приложения;
- Spring Web — модуль, реализующий обработку HTTP-запросов, маршрутизацию, и построение RESTful API. С его помощью реализовано взаимодействие клиентской и серверной частей, включая передачу данных о пикселях, рейтингах и действиях игроков;
- Spring Data — модуль для работы с базой данных с использованием объектно-реляционного отображения. Он позволяет создавать репозитории и взаимодействовать с таблицами базы данных через Java-классы, что упрощает реализацию слоя доступа к данным;
- Spring Security — модуль, обеспечивающий защиту приложения. В проекте он используется для реализации регистрации, аутентификации и авторизации пользователей, а также ограничения доступа к защищённым ресурсам в зависимости от роли (обычный игрок, модератор и т.д.).

Использование Spring позволило добиться высокой скорости обработки запросов, надёжности и масштабируемости серверной части. Благодаря модульной архитектуре фреймворка, разработка велась по принципу разделения ответственности, что улучшило структуру проекта и упростило поддержку и развитие функционала. [5]

### 3.3 База данных PostgreSQL



Рисунок 6 – PostgreSQL

Для хранения данных в проекте Pixel Wars используется система управления базами данных PostgreSQL. Это современное, высокопроизводительное и надёжное решение, широко применяемое для создания отказоустойчивых и масштабируемых серверных приложений.

Структура базы данных сформирована в соответствии с предметной областью проекта и охватывает три основные сущности: цвет, пиксель

и пользователь. Связи между таблицами организованы через внешние ключи, обеспечивая логическую целостность и согласованность данных.

Структура базы данных включает следующие таблицы:

**color:**

- `id` — уникальный идентификатор цвета (тип: `integer`);
- `color` — строковое представление цвета в шестнадцатеричном формате (тип: `text`).

**pixel:**

- `id` — уникальный идентификатор пикселя (тип: `integer`);
- `color` — цвет пикселя в формате RGB (тип: `text`);
- `timestamp` — время установки пикселя (тип: `bigint`);
- `x`, `y` — координаты пикселя на холсте (тип: `integer`);
- `user_id` — внешний ключ, ссылающийся на таблицу `users` (тип: `integer`).

**users:**

- `id` — уникальный идентификатор пользователя (тип: `integer`);
- `email` — адрес электронной почты (тип: `text`);
- `name` — имя пользователя (тип: `text`);
- `picture` — URL-адрес изображения профиля (тип: `text`);
- `status` — статус пользователя, например `owner` (тип: `text`);
- `subject` — идентификатор в внешней системе аутентификации (тип: `bigint`).

Связи между таблицами:

- Каждый пиксель связан с конкретным пользователем через `user_id`.
  - Цвета пикселей хранятся как текстовые значения RGB, при этом в отдельной таблице `color` хранится справочник с основными цветами в HEX-формате.

Взаимодействие приложения с PostgreSQL реализовано через Spring Data JPA, что обеспечивает надёжный и гибкий доступ к данным без необходимости явного написания SQL-запросов. Конфигурация подключения к базе данных задаётся через `application.properties`.

Выбор PostgreSQL обусловлен её устойчивостью к сбоям, поддержкой транзакций, возможностями масштабирования и глубокой интеграцией с инструментами Java и Spring.

### 3.4 Google OAuth 2.0 API

Для реализации аутентификации пользователей в приложении Pixel Wars использовалась технология Google OAuth 2.0 API, интегрированная с помощью фреймворка Spring Security. OAuth 2.0 представляет собой широко используемый протокол авторизации, позволяющий сторонним приложениям безопасно получать ограниченный доступ к

пользовательским данным без необходимости запрашивать пароль.

С помощью данной технологии пользователи получают возможность входа в систему через свою учётную запись Google. Это повышает уровень безопасности приложения и упрощает процесс аутентификации, поскольку пользователь не вводит логин и пароль вручную.

После успешной авторизации Google перенаправляет пользователя обратно в приложение с временным авторизационным кодом. Этот код обменивается на access token, после чего приложение может получить базовую информацию о пользователе.

Использование Google OAuth 2.0 API в сочетании со Spring Security позволило реализовать следующие преимущества:

- безопасная авторизация без хранения паролей пользователей;
- удобство и быстрота входа в систему;
- снижение барьера входа для новых пользователей.

### 3.5 Интерфейс

Во внешнем интерфейсе приложения Pixel Wars использовались стандартные веб-технологии — HTML, CSS и JavaScript [6], а также шаблонизатор Thymeleaf, встроенный в стек Spring Boot.

HTML (HyperText Markup Language) применялся для создания структуры веб-страниц и определения размещения элементов пользовательского интерфейса. Все страницы проекта были реализованы с учётом семантической разметки, что облегчает их поддержку и улучшает восприятие контента пользователями.

CSS (Cascading Style Sheets) использовался для стилизации HTML-элементов и создания визуально привлекательного интерфейса. Основное внимание уделялось адаптивной верстке и обеспечению читаемости интерфейса на различных устройствах.

JavaScript применялся для реализации интерактивности на клиентской стороне. С его помощью реализованы такие функции, как динамическое обновление содержимого без перезагрузки страницы (через асинхронные запросы), обработка событий пользователя и взаимодействие с серверной частью приложения через REST API.

Особое внимание в проекте уделялось интеграции с Thymeleaf — серверным Java-шаблонизатором, тесно интегрированным с Spring Boot. С его помощью осуществлялся рендеринг HTML-страниц с динамическими данными, передаваемыми с сервера. Использование Thymeleaf позволило избежать избыточной ручной генерации HTML и упростить передачу данных от сервера к клиенту, благодаря поддержке выражений и условной логики в шаблонах.

### 3.6 Среда разработки IntelliJ IDEA

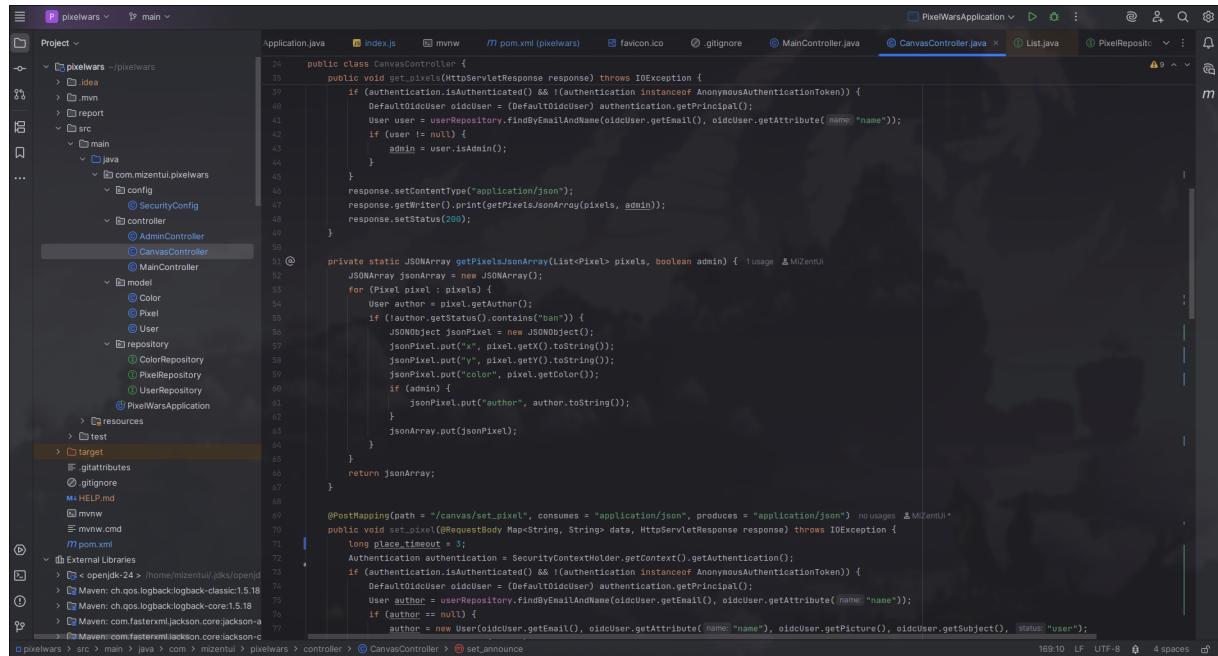


Рисунок 7 – Экран редактирования кода IntelliJ IDEA

IntelliJ IDEA — это интегрированная среда разработки (IDE), созданная компанией JetBrains для разработки программного обеспечения на языке Java и других языках платформы JVM. С момента своего появления в 2001 году, IntelliJ IDEA зарекомендовала себя как одна из наиболее мощных и удобных сред разработки, широко используемая в промышленной и академической практике. [7]

IntelliJ IDEA предоставляет полный набор инструментов, необходимых для создания современных Java-приложений, включая интеллектуальную навигацию по коду, рефакторинг, отладку, анализ производительности, тестирование истроенную поддержку систем контроля версий. Среда активно поддерживает фреймворки, такие как Spring, Hibernate, Maven, Gradle и многие другие, что делает её особенно удобной при разработке серверной части сложных приложений.

В рамках курсового проекта IntelliJ IDEA использовалась для разработки серверной части системы Pixel Wars. Она обеспечила удобную интеграцию с фреймворком Spring Boot, возможность работы с базой данных PostgreSQL через встроенные плагины, а также автоматическую генерацию и управление зависимостями через систему сборки Gradle.

Причины выбора данной среды разработки включают:

1. Глубокая интеграция с Java и Spring Boot. IntelliJ IDEA предоставляет интеллектуальную поддержку аннотаций, конфигурации и кода, специфичного для Spring, что значительно ускоряет процесс разработки.

2. Автоматизация и интеллектуальные подсказки. Среда активно подсказывает доступные методы, проверяет корректность конфигураций, предлагает варианты рефакторинга и автоматически управляет импортами.

3. Удобная работа с базой данных. Встроенные средства позволяют подключаться к PostgreSQL напрямую из среды, выполнять запросы, просматривать структуру таблиц и редактировать содержимое.

4. Инструменты отладки и тестирования. IntelliJ IDEA содержит мощные инструменты для локальной отладки, трассировки выполнения и написания модульных тестов с использованием JUnit.

Таким образом, использование IntelliJ IDEA обеспечило стабильную и эффективную среду для разработки серверной части проекта. Её функциональные возможности и адаптация под современные технологии Java сделали процесс создания приложения удобным, гибким и соответствующим промышленным стандартам.

## 3.7 Среда разработки Visual Studio Code

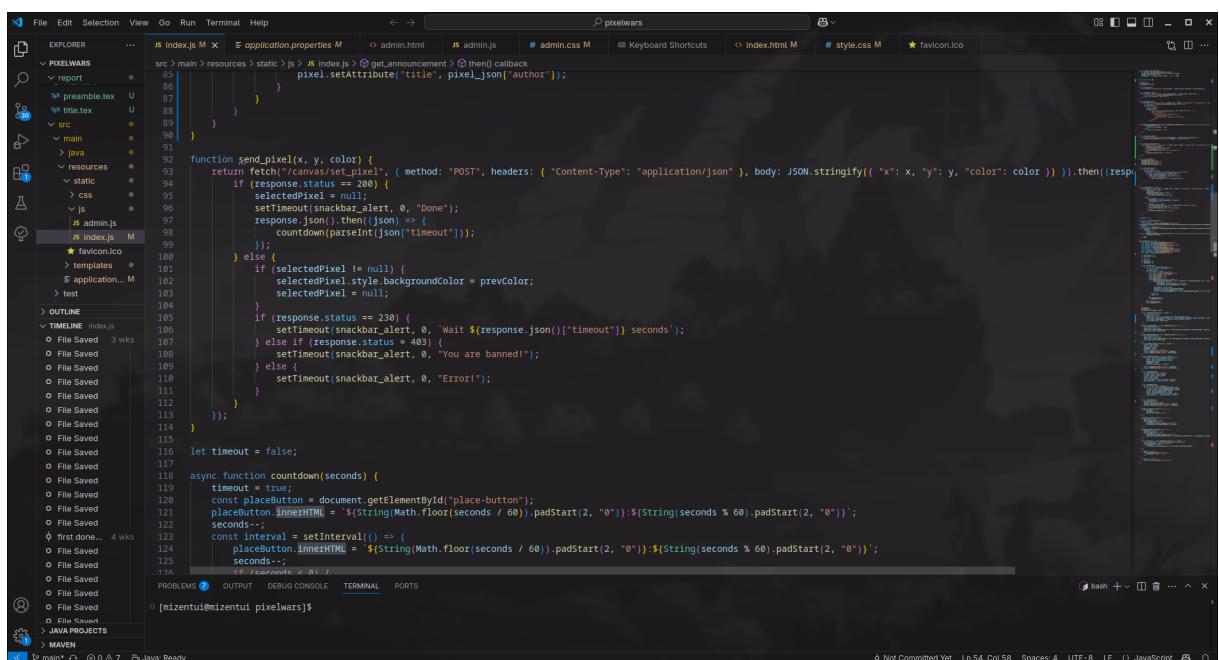


Рисунок 8 – Экран редактирования кода Visual Studio Code

Visual Studio Code (VS Code) — это лёгкая, но мощная интегрированная среда разработки, разработанная компанией Microsoft. С момента своего выхода в 2015 году она завоевала огромную популярность благодаря простоте, быстроте и гибкости. VS Code является кроссплатформенным инструментом, доступным на Windows, macOS и

Linux, что делает его универсальным решением для разработчиков по всему миру. [8]

VS Code не перегружена лишними функциями, но в то же время предоставляет всё необходимое для разработки: интеллектуальную подсветку синтаксиса, автодополнение, отладку, поддержку Git, систему контроля версий и встроенный терминал. Это позволяет разрабатывать и тестировать код быстро и эффективно, без лишних задержек.

В проекте Pixel Wars Visual Studio Code использовалась для разработки фронтенда. Среда идеально подошла для создания клиентской части приложения, поскольку она поддерживает работу с JavaScript, HTML и CSS, а также позволяет интегрировать популярные фреймворки и библиотеки.

Основные причины выбора VS Code включают:

1. Лёгкость и быстродействие. VS Code быстро запускается и работает с минимальными задержками, что делает её идеальной для разработки фронтенд-части и быстрого прототипирования.

2. Расширяемость. Благодаря множеству доступных расширений, VS Code легко адаптируется под любые задачи. В проекте использовались расширения для работы с JavaScript, CSS, HTML, а также плагины для отладки и тестирования кода.

3. Интеграция с Git и другими системами контроля версий. VS Code имеет встроенную поддержку Git, что упрощает работу с репозиториями и позволяет эффективно управлять версиями проекта.

Visual Studio Code идеально подошла для разработки фронтенда Pixel Wars, предоставив удобные инструменты для работы с кодом, расширяемость и простоту в освоении. Весь процесс разработки клиентской части был значительно ускорен благодаря этой мощной и гибкой среде разработки. Даже этот документ написан в VS Code с использованием L<sup>A</sup>T<sub>E</sub>X.

### 3.8 Хостинг и развёртывание приложения

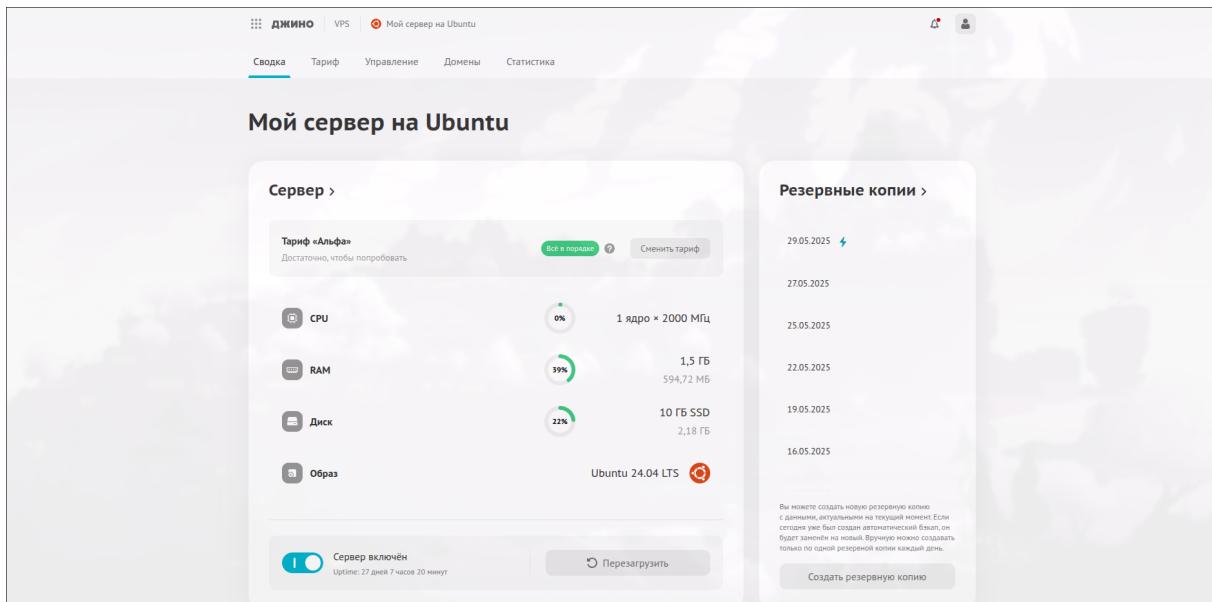


Рисунок 9 – Страница сервера на jino.ru

Для размещения серверной части проекта Pixel Wars использовался платный хостинг-провайдер Jino.ru. Выбор данного провайдера был обусловлен возможностью полного управления сервером через SSH-доступ, поддержкой системных сервисов, а также интеграцией доменов и SSL-сертификатов. [9]

Развёртывание приложения осуществляется на выделенный виртуальный сервер (VPS), работающий под управлением операционной системы Linux. Доступ к серверу выполняется с использованием безопасного протокола SSH, что позволяет выполнять все действия удалённо: загрузку файлов, настройку служб и управление серверной частью.

Сборка проекта осуществляется с использованием системы Maven. После успешной сборки на локальной машине создаётся JAR-файл, который загружается на сервер с помощью SCP. Развёртывание приложения производится вручную, без использования CI/CD, с контролем всех этапов.

Для автоматического запуска и надёжной поддержки работоспособности серверной части был создан отдельный systemd-сервис, зарегистрированный в системе как юнит. Этот сервис обеспечивает:

- автоматический запуск приложения при загрузке сервера;
- перезапуск в случае сбоев;
- логирование через системный журнал.

Конфигурация systemd-юнита включает в себя путь к исполняемому JAR-файлу, указание рабочей директории, параметров Java-машины и стандартных параметров управления службой.

Для доступа к приложению по HTTPS был зарегистрирован собственный домен, приобретённый через панель управления Jino. К домену подключён wildcard-сертификат, позволяющий защищать не только основной домен, но и все его поддомены. Сертификат установлен на стороне веб-сервера, который при необходимости может проксировать запросы к приложению, обеспечивая безопасную и надёжную маршрутизацию трафика.

Использование Jino в качестве хостинг-платформы позволило обеспечить:

- полную автономность и контроль над серверной средой;
- надёжную работу приложения в режиме 24/7;
- безопасный доступ пользователей по защищённому протоколу HTTPS;
- удобную интеграцию домена и сертификатов.

Развёртывание проекта в реальной среде подтвердило его работоспособность вне среды разработки и позволило провести полноценное тестирование функциональности в условиях, приближенных к промышленной эксплуатации.

## 4 ТЕХНИЧЕСКИЕ ОСОБЕННОСТИ РЕАЛИЗАЦИИ

Данная глава посвящена описанию ключевых технических решений, реализованных в проекте.

### 4.1 Обновление холста

Механизм обновления холста в проекте Pixel Wars реализован на основе архитектуры клиент–сервер. Обновление инициируется на клиентской стороне с помощью HTTP-запроса к серверу, который возвращает актуальное состояние холста в формате JSON. Данные обрабатываются и визуализируются без необходимости перезагрузки страницы, обеспечивая плавную и интерактивную работу интерфейса.

Клиентская часть периодически отправляет GET-запрос на REST-эндпоинт, предназначенный для получения текущего состояния холста. В ответ сервер возвращает массив пикселей, каждый из которых содержит координаты и цвет. После получения ответа данные передаются в обработчик, который обновляет внешний вид холста на странице пользователя.

Серверная часть принимает запрос и формирует ответ, основываясь на данных, хранящихся в базе. Сначала извлекаются последние установленные пиксели для каждой координаты. Затем, при наличии активной авторизации, проверяется роль пользователя: если это администратор, в ответ дополнительно включается информация об авторах пикселей. Пиксели, установленные пользователями со статусом блокировки, фильтруются и не включаются в выдачу.

Ответ формируется в формате JSON и отправляется клиенту с указанием соответствующего типа контента.

После получения ответа клиентская часть обрабатывает каждый пиксель: проверяется его позиция и текущее состояние на экране. Если пиксель ранее не был обновлён, ему присваивается цвет, полученный от сервера. В случае если пользователь имеет административные права, к пикселю добавляется всплывающая подсказка с именем автора.

В результате реализованная система позволяет:

- отображать актуальное состояние холста всем пользователям;
- разграничивать доступ к дополнительной информации (например, авторам пикселей) в зависимости от роли;
- динамически обновлять интерфейс без полной перезагрузки страницы.

## 4.2 Установка пикселя на холст

Механизм установки пикселя реализован с использованием REST-запросов. Пользователь, взаимодействуя с холстом, выбирает координаты и цвет, после чего инициирует отправку запроса на сервер для фиксации результата. Это обеспечивает динамичное взаимодействие без полной перезагрузки страницы и позволяет достичь высокой отзывчивости пользовательского интерфейса.

На стороне клиента формируется POST-запрос, содержащий следующие данные:

- координаты пикселя (x, y);
- цвет в формате RGB или HEX;
- тело запроса в формате JSON.

Запрос направляется на соответствующий серверный эндпоинт, где происходит последовательная обработка.

Во-первых, осуществляется проверка аутентификации пользователя. Затем проверяется статус доступа. В случае блокировки (например, статус `ban`) сервер отклоняет запрос с кодом состояния 403 (`Forbidden`).

Во-вторых, применяется ограничение по времени между действиями. Сервер анализирует историю пикселей, размещенных пользователем, и вычисляет, сколько времени прошло с момента последнего действия. Если минимальный интервал ещё не истёк, сервер возвращает специальный код 230 и значение оставшегося времени ожидания.

Если все условия соблюdenы (пользователь авторизован, не заблокирован и не нарушает временные ограничения), сервер сохраняет пиксель в базу данных.

Вносятся следующие параметры:

- координаты: x, y;
- цвет пикселя;
- временная метка;
- идентификатор пользователя.

В ответ клиент получает JSON-объект, содержащий:

- статус выполнения (успешно или с ошибкой);
- значение таймера `timeout`, определяющее задержку до следующего действия.

После получения ответа клиентская часть:

- запускает таймер обратного отсчёта;
- временно блокирует интерфейс взаимодействия с холстом;
- в случае ошибки выводит уведомление о причине отказа (ожидание, запрет доступа, общая ошибка).

Таким образом, установка пикселя реализована как контролируемая операция с учётом аутентификации и ролевой модели. Данная

архитектура позволяет обеспечить стабильную и безопасную механику многопользовательского взаимодействия.

### 4.3 Система модерации и панель администратора



Рисунок 10 – Интерфейс панели администратора

Для обеспечения контроля за поведением пользователей и соблюдением игровых правил в проекте Pixel Wars реализована система модерации, доступ к которой имеет только администратор. Административные функции включают просмотр всех зарегистрированных пользователей, изменение их статусов и блокировку нарушителей.

Модерация реализована через отдельный REST-контроллер серверной части, доступ к которому ограничен на уровне авторизации. При входе в административный интерфейс, если пользователь обладает статусом администратора, он получает доступ к следующему функционалу:

- просмотр списка всех зарегистрированных пользователей;
- отображение основной информации о каждом пользователе (имя, email, subject, статус, изображение профиля, количество установленных пикселей);
- возможность изменить статус пользователя.

На стороне клиента административная панель реализована в виде отдельного интерфейсного блока, который динамически запрашивает список пользователей с сервера. Для этого используется периодическая отправка GET-запросов на серверный эндпоинт, обрабатываемый

соответствующим контроллером. При получении ответа в формате JSON информация обновляется на странице без перезагрузки.

При изменении статуса конкретного пользователя (например, перевод в статус `ban` или `admin`) отправляется POST-запрос с указанием идентификатора пользователя и нового статуса. Сервер проверяет авторизацию текущего пользователя, наличие прав администратора и допустимость изменения (в частности, невозможно изменить статус пользователей со статусом «`owner`»). После успешного обновления информации клиент получает подтверждение и обновлённый список пользователей.

Для визуальной обратной связи используется система уведомлений, реализованная в виде всплывающего блока. Пользователь получает сообщение о результате операции: успешно ли изменён статус или произошла ошибка.

Список возможных статусов:

- `owner` — статус владельца — привилегии как у `admin`, но не возможно изменить статус владельцу через панель администратора;
- `admin` — статус администратора — может менять статус другим пользователям, кроме владельца, также заходить в панель администратора и просматривать данные пользователей;
- `creator` — статус художника — может ставить пиксели без временного ограничения;
- `user` — статус пользователя;
- `ban` — статус блокировки — пользователь не может рисовать и поставленные ранее пиксели не отображаются на общем холсте.

Система модерации предоставляет администратору инструменты для управления пользовательской активностью, контроля за соблюдением правил, блокировки нарушителей на платформе. Простота реализации и интуитивный интерфейс позволяют эффективно использовать административные функции даже без технической подготовки.

## 5 ТЕСТИРОВАНИЕ

Тестирование проекта Pixel Wars проводилось поэтапно с постепенным расширением аудитории. Такой подход позволил выявить ключевые проблемы на ранних стадиях и обеспечить стабильную работу приложения при высокой нагрузке.

**Ручное тестирование.** На первом этапе тестирование выполнялось в ручном режиме непосредственно разработчиком. Проверялись базовые функции: установка пикселя, обновление холста, авторизация, система ограничения по времени, работа административной панели. Выявленные ошибки включали некорректную обработку исключений, нарушения в логике работы с таймерами, а также визуальные баги в интерфейсе.

**Тестирование с участием небольшой группы пользователей.** На втором этапе была подключена небольшая группа из 4 человек, которые моделировали реальное поведение пользователей: взаимодействовали с холстом, пробовали нарушить ограничения, отправляли частые запросы с разных устройств и браузеров. Это позволило проверить устойчивость системы при одновременных действиях и протестировать серверную нагрузку в условиях близких к реальным.

**Публикация проекта для широкой аудитории.** Финальный этап тестирования проходил в условиях реального использования. Проект был размещён на удалённом сервере и открыт для студентов БГУИР — общее количество пользователей составило около 50 человек. За 5 дней активного взаимодействия с холстом участники оставили множество изображений и надписей. Это позволило оценить, как приложение работает при нагрузке, а также получить обратную связь от реальных пользователей.

В ходе тестирования были выявлены и устраниены следующие проблемы:

- ошибки, связанные с блокировкой пользователей и логикой смены статуса;
- нестабильная работа timeout-механизма;
- высокая нагрузка на клиент при масштабном обновлении холста;
- недоработки в панели администратора.

В результате был собран полноценный холст, демонстрирующий коллективную активность. На нём присутствуют рисунки и другие элементы, созданные пользователями за время публичного тестирования.

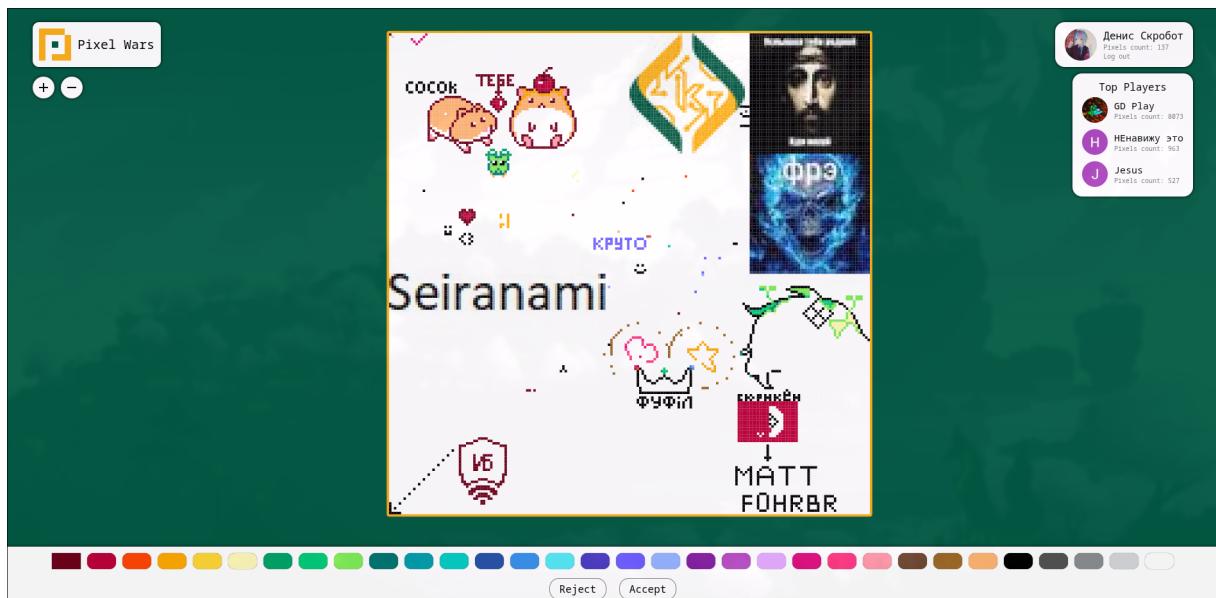


Рисунок 11 – Интерфейс и холст Pixel Wars

# ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта была разработана многопользовательская онлайн-игра Pixel Wars, представляющая собой цифровой холст для коллективного рисования в реальном времени. Основной целью проекта было создание функционального, устойчивого и масштабируемого приложения, способного объединить большое количество пользователей в общем интерактивном пространстве.

Основные результаты и выводы:

**1. Реализация клиент-серверной архитектуры.** Приложение построено по принципу разделения клиентской и серверной частей. Серверная часть реализована на Java с использованием Spring Boot, PostgreSQL. Клиент взаимодействует с сервером через REST-интерфейс, обеспечивая передачу данных о пикселях и состояниях пользователей.

**2. Механизм управления холстом.** Разработан механизм установки пикселей с учётом ограничения по времени, фильтрации действий заблокированных пользователей и отображения авторов рисунков для администраторов.

**3. Панель администратора и система модерации.** В системе реализован модуль управления пользователями. Администратор может просматривать информацию о пользователях, изменять их статус, а также отслеживать активность на холсте.

**4. Полноценное тестирование и развёртывание.** Проект прошёл три этапа тестирования: ручное, с привлечением небольшой фокус-группы, и широкое публичное тестирование с участием около 50 студентов. В результате удалось собрать обратную связь, выявить и устранить ряд недочётов.

Разработка данного проекта позволила углубить знания в области веб-технологий, серверной архитектуры, работы с базами данных, а также познакомиться с практикой проектирования и защиты распределённых систем. Были освоены современные подходы к реализации авторизации, разграничению прав доступа и синхронизации клиентской и серверной логики.

Таким образом, проект Pixel Wars стал не только учебным, но и практически значимым решением, подтвердившим свою работоспособность в реальной среде. Он заложил основу для дальнейшего развития и масштабирования: расширения функциональности, внедрения новых игровых механик, улучшения производительности и визуальной составляющей. Полученный мною опыт может быть использован в будущих проектах, связанных с созданием интерактивных многопользовательских приложений.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

- [1] r/Place [Электронный ресурс]. – Режим доступа : <https://www.reddit.com/r/place/>.
- [2] PixelPlace.io [Электронный ресурс]. – Режим доступа : <https://pixelplace.io/>.
- [3] PixelCanvas.io [Электронный ресурс]. – Режим доступа : <https://pixelcanvas.io/>.
- [4] Schildt, H. Java: The Complete Reference, Ninth Edition / H. Schildt. — McGraw-Hill Education, 2014. — с. 17 – 23.
- [5] Spring Guides [Электронный ресурс]. – Режим доступа : <https://spring.io/guides>.
- [6] Resources for Developers, by Developers [Электронный ресурс]. – Режим доступа : <https://developer.mozilla.org/en-US/>.
- [7] IntelliJ IDEA: The IDE for Professional Development in Java and Kotlin [Электронный ресурс]. – Режим доступа : <https://www.jetbrains.com/idea/>.
- [8] Visual Studio Code [Электронный ресурс]. – Режим доступа : <https://code.visualstudio.com/>.
- [9] Джино: Виртуальный хостинг [Электронный ресурс]. – Режим доступа : <https://jino.ru/>.

# ПРИЛОЖЕНИЕ А

## (обязательное)

### Листинг программного кода

#### AdminController.java

```
package com.mizentui.pixelwars.controller;

import com.mizentui.pixelwars.model.User;
import com.mizentui.pixelwars.repository.UserRepository;
import jakarta.servlet.http.HttpServletResponse;
import org.json.JSONArray;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.
AnonymousAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.oauth2.core.oidc.user.DefaultOidcUser
;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import java.io.IOException;
import java.util.Map;
import java.util.Optional;

@RestController
public class AdminController {

    @Autowired
    private UserRepository userRepository;

    @GetMapping("/admin/get_users")
    public void get_users(HttpServletRequest response) throws IOException {
        Authentication authentication = SecurityContextHolder.getContext()
            .getAuthentication();
        if (authentication.isAuthenticated() && !(authentication
instanceof AnonymousAuthenticationToken)) {
            DefaultOidcUser oidcUser = (DefaultOidcUser) authentication.
                getPrincipal();
            User user = userRepository.findByEmailAndName(oidcUser.getEmail
                (), oidcUser.getAttribute("name"));
            if (user != null) {
                if (user.isAdmin()) {
                    Iterable<User> users = userRepository.findAll();
                    response.setContentType("application/json");
                }
            }
        }
    }
}
```

```

            response.getWriter().print(getUsersJsonArray(users));
            response.setStatus(200);
        }
    }
} else {
    response.setStatus(403);
}
}

private static JSONArray getUsersJsonArray(Iterable<User> users) {
    JSONArray jsonArray = new JSONArray();
    for (User currrentUser : users) {
        JSONObject jsonUser = new JSONObject();
        jsonUser.put("id", currrentUser.getId());
        jsonUser.put("name", currrentUser.getName());
        jsonUser.put("email", currrentUser.getEmail());
        jsonUser.put("subject", currrentUser.getSubject());
        jsonUser.put("picture", currrentUser.getPicture());
        jsonUser.put("status", currrentUser.getStatus());
        jsonUser.put("pixels_count", currrentUser.getPixelsCount());
        jsonArray.put(jsonUser);
    }
    return jsonArray;
}

@PostMapping(path = "/admin/set_user_status", consumes = "application/
    json", produces = "application/json")
public void set_user_status(@RequestBody Map<String, String> data,
    HttpServletResponse response) {
    Authentication authentication = SecurityContextHolder.getContext()
        .getAuthentication();
    if (authentication.isAuthenticated() && !(authentication
        instanceof AnonymousAuthenticationToken)) {
        DefaultOidcUser oidcUser = (DefaultOidcUser) authentication.
            getPrincipal();
        User user = userRepository.findByEmailAndName(oidcUser.getEmail
            (), oidcUser.getAttribute("name"));
        if (user != null) {
            if (user.isAdmin()) {
                Optional<User> optionalUser = userRepository.findById(
                    Long.parseLong(data.get("id")));
                if (optionalUser.isPresent() && !optionalUser.get().
                    getStatus().equals("owner")) {
                    User changeableUser = optionalUser.get();
                    changeableUser.setStatus(data.get("status"));
                    userRepository.save(changeableUser);
                }
                response.setStatus(200);
            }
        }
    }
}
}
}
```

```
}
```

## CanvasController.java

```
package com.mizentui.pixelwars.controller;

import com.mizentui.pixelwars.model.Pixel;
import com.mizentui.pixelwars.model.User;
import com.mizentui.pixelwars.repository.PixelRepository;
import com.mizentui.pixelwars.repository.UserRepository;
import jakarta.servlet.http.HttpServletResponse;
import org.json.JSONArray;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.
    AnonymousAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.oauth2.core.oidc.user.DefaultOidcUser
    ;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import java.io.IOException;
import java.util.*;

@RestController
public class CanvasController {

    @Autowired
    private PixelRepository pixelRepository;

    @Autowired
    private UserRepository userRepository;

    private String announce = "";

    @GetMapping("/canvas/get_pixels")
    public void get_pixels(HttpServletResponse response) throws
        IOException {
        List<Pixel> pixels = pixelRepository.getLastPixels();
        Authentication authentication = SecurityContextHolder.getContext()
            .getAuthentication();
        boolean admin = false;
        if (authentication.isAuthenticated() && !(authentication
            instanceof AnonymousAuthenticationToken)) {
            DefaultOidcUser oidcUser = (DefaultOidcUser) authentication.
                getPrincipal();
            User user = userRepository.findByName(oidcUser.getEmail)
```

```

        (), oidcUser.getAttribute("name"));
    if (user != null) {
        admin = user.isAdmin();
    }
}
response.setContentType("application/json");
response.getWriter().print(getPixelsJsonArray(pixels, admin));
response.setStatus(200);
}

private static JSONArray getPixelsJsonArray(List<Pixel> pixels,
    boolean admin) {
    JSONArray jsonArray = new JSONArray();
    for (Pixel pixel : pixels) {
        User author = pixel.getAuthor();
        if (!author.getStatus().contains("ban")) {
            JSONObject jsonPixel = new JSONObject();
            jsonPixel.put("x", pixel.getX().toString());
            jsonPixel.put("y", pixel.getY().toString());
            jsonPixel.put("color", pixel.getColor());
            if (admin) {
                jsonPixel.put("author", author.toString());
            }
            jsonArray.put(jsonPixel);
        }
    }
    return jsonArray;
}

@PostMapping(path = "/canvas/set_pixel", consumes = "application/json"
    , produces = "application/json")
public void set_pixel(@RequestBody Map<String, String> data,
    HttpServletResponse response) throws IOException {
    long place_timeout = 3;
    Authentication authentication = SecurityContextHolder.getContext()
        .getAuthentication();
    if (authentication.isAuthenticated() && !(authentication
        instanceof AnonymousAuthenticationToken)) {
        DefaultOidcUser oidcUser = (DefaultOidcUser) authentication.
            getPrincipal();
        User author = userRepository.findByEmailAndName(oidcUser.
            getEmail(), oidcUser.getAttribute("name"));
        if (author == null) {
            author = new User(oidcUser.getEmail(), oidcUser.
               getAttribute("name"), oidcUser.getPicture(), oidcUser.
                getSubject(), "user");
            userRepository.save(author);
        }
        JSONObject json = new JSONObject();
        if (author.isAdmin() || author.getStatus().toLowerCase().
            contains("creator")) {
            place_timeout = 0;

```

```

    }
    json.put("timeout", place_timeout);
    response.getWriter().print(json);
    List<Pixel> pixels = author.getPixels();
    if (pixels != null) {
        pixels.sort(Comparator.comparing(Pixel::getTimestamp).
            reversed());
    }
    if (author.getStatus().toLowerCase().contains("ban")) {
        response.setStatus(403);
    } else if (pixels == null || (pixels.isEmpty() || pixels.
        getLast() != null && Math.abs(System.currentTimeMillis() -
        pixels.getLast().getTimestamp()) > place_timeout * 1000)) {
        pixelRepository.save(new Pixel(Integer.parseInt(data.get("x"
            )), Integer.parseInt(data.get("y")), data.get("color"),
            System.currentTimeMillis(), author));
        response.setStatus(200);
    } else {
        response.setStatus(230);
    }
} else {
    response.setStatus(401);
}
}

@PostMapping(path = "/canvas/get_top_users", consumes = "application/
    json", produces = "application/json")
public void get_top_users(@RequestBody Map<String, String> data,
    HttpServletResponse response) throws IOException {
    Authentication authentication = SecurityContextHolder.getContext()
        .getAuthentication();
    boolean admin = false;
    if (authentication.isAuthenticated() && !(authentication
        instanceof AnonymousAuthenticationToken)) {
        DefaultOidcUser oidcUser = (DefaultOidcUser) authentication.
            getPrincipal();
        User user = userRepository.findByEmailAndName(oidcUser.getEmail
            (), oidcUser.getAttribute("name"));
        if (user != null) {
            admin = user.isAdmin();
        }
    }
    JSONArray jsonArray = new JSONArray();
    List<User> topUsers = new ArrayList<>();
    userRepository.findAllById(pixelRepository.getTopAuthorsId(Long.
        parseLong(data.get("count")))).forEach(topUsers::add);
    topUsers.sort(Comparator.comparing(User::getPixelsCount).reversed
        ());
    for (User user : topUsers) {
        JSONObject jsonUser = new JSONObject();
        jsonUser.put("name", user.getName());
        jsonUser.put("picture", user.getPicture());
        jsonArray.add(jsonUser);
    }
}
}

```

```

        jsonUser.put("pixel_count", user.getPixelsCount());
        if (admin) {
            jsonUser.put("user_info", user.toString());
        }
        jsonArray.put(jsonUser);
    }
    response.setContentType("application/json");
    response.getWriter().print(jsonArray);
    response.setStatus(200);
}

@GetMapping("/canvas/get_pixels_count")
public void get_pixels_count(HttpServletRequest response) throws
IOException {
    Authentication authentication = SecurityContextHolder.getContext()
        .getAuthentication();
    if (authentication.isAuthenticated() && !(authentication
        instanceof AnonymousAuthenticationToken)) {
        DefaultOidcUser oidcUser = (DefaultOidcUser) authentication.
            getPrincipal();
        response.setContentType("application/json");
        User user = userRepository.findByEmailAndName(oidcUser.getEmail
            (), oidcUser.getAttribute("name"));
        if (user != null) {
            response.getWriter().print("{\"count\": " + user.
                getPixelsCount() + '}');
        } else {
            response.getWriter().print("{\"count\": 0}");
        }
        response.setStatus(200);
    }
}

@GetMapping("/canvas/get_announcement")
public void get_announce(HttpServletRequest response) throws
IOException {
    response.setContentType("application/json");
    response.getWriter().print("{\"announcement\": \"\" + announce + "
        +"\"");
    response.setStatus(200);
}

@PostMapping(path = "/canvas/set_announcement", consumes =
    "application/json", produces = "application/json")
public void set_announce(@RequestBody Map<String, String> data,
    HttpServletResponse response) {
    Authentication authentication = SecurityContextHolder.getContext()
        .getAuthentication();
    if (authentication.isAuthenticated() && !(authentication
        instanceof AnonymousAuthenticationToken)) {
        DefaultOidcUser oidcUser = (DefaultOidcUser) authentication.
            getPrincipal();

```

```

        User user = userRepository.findByEmailAndName(oidcUser.getEmail
            (), oidcUser.getAttribute("name"));
        if (user != null && user.isAdmin()) {
            announce = data.get("announcement");
            response.setStatus(200);
        }
        else {
            response.setStatus(403);
        }
    }
}

```

### MainController.java

```

package com.mizentui.pixelwars.controller;

import com.mizentui.pixelwars.model.Color;
import com.mizentui.pixelwars.model.User;
import com.mizentui.pixelwars.repository.ColorRepository;
import com.mizentui.pixelwars.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.
    AnonymousAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.oauth2.core.oidc.user.DefaultOidcUser
    ;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class MainController {

    @Autowired
    private ColorRepository colorRepository;

    @Autowired
    private UserRepository userRepository;

    @GetMapping("/")
    public String index(Model model) {
        Authentication authentication = SecurityContextHolder.getContext()
            .getAuthentication();
        Iterable<Color> colors = colorRepository.findAll();
        model.addAttribute("colors", colors);
        if (authentication.isAuthenticated() && !(authentication
            instanceof AnonymousAuthenticationToken)) {
            DefaultOidcUser user = (DefaultOidcUser) authentication.
                getPrincipal();

```

```

        model.addAttribute("authorized", "true");
        model.addAttribute("username", user.getAttribute("name"));
        model.addAttribute("picture_url", user.getPicture());
    } else {
        model.addAttribute("authorized", "");
        model.addAttribute("username", "Log in");
    }
    return "index";
}

@GetMapping("/admin")
public String admin(Model model) {
    Authentication authentication = SecurityContextHolder.getContext()
        .getAuthentication();
    if (authentication.isAuthenticated() && !(authentication
        instanceof AnonymousAuthenticationToken)) {
        DefaultOidcUser oidcUser = (DefaultOidcUser) authentication.
            getPrincipal();
        User user = userRepository.findByEmailAndName(oidcUser.getEmail
            (), oidcUser.getAttribute("name"));
        if (user != null) {
            if (user.isAdmin()) {
                model.addAttribute("admin", "true");
                model.addAttribute("users", userRepository.findAll());
            }
        }
    }
    return "admin";
}
}

```

### PixelWarsApplication.java

```

package com.mizentui.pixelwars;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class PixelWarsApplication {

    public static void main(String[] args) {
        SpringApplication.run(PixelWarsApplication.class, args);
    }
}

```