Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования   
**Национальный исследовательский университет «МЭИ»**

**Курсовая работа**

по курсу«**WEB-технологии**»

**АРХИВ НАУЧНЫХ СТАТЕЙ**

Выполнила: Чалая Ю. М.

Группа: А-08-21

Вариант 18

Москва 2024

Оглавление

[Задание 3](#_Toc4002)

[Описание этапов разработки 3](#_Toc16347)

[Используемые версии программного обеспечения 3](#_Toc26900)

[Структура базы данных 3](#_Toc21100)

[Разработка на Node.js (JavaScript) 4](#_Toc28496)

[Описание работы сайта 7](#_Toc13837)

[Приложение А. Выбранный стек технологий. 13](#_Toc20741)

[Приложение Б. Листинг программы. 17](#_Toc12717)

# Задание

Разработать Web-приложение, представляющее из себя сайт - архив научных статей. Создать структуры базы данных под управлением СУБД PostgreSQL для хранения и обработки данных о пользователях (в том числе паролей от учетных записей). Вся текстовая и графическая информация также должна храниться в БД.

# Описание этапов разработки

Процесс разработки можно подразделить на следующие этапы:

1. Создание базы данных для хранения научных статей и персональной информации о пользователях, а также установка связей между таблицами базы данных.
2. Разработка клиентской части приложения: верстка страниц на HTML, создание каскадных стилей CSS, а так же подключение скриптов на JavaScript
3. Разработка серверной части приложения на языке JavaScript

# Используемые версии программного обеспечения

Windows 10 Pro версия 22H2

Node.js 20.11.1

Prisma 5.11.0

PostgreSQL 16.2

# Структура базы данных

Структура базы данных представляет из себя две таблицы (таблица \_prisma\_migrations была создана автоматически при использовании prisma).

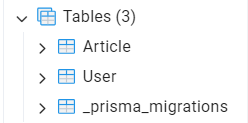


Рис. 1. Таблицы в базе данных.

Таблица Article создержит информацию о статье. Имеет 7 полей: id - идентификационный номер статьи, name - название статьи, text\_article - текст статьи, isPublished - опубликована статья или нет, isModerating - на модерации статья или нет, date\_published - дата публикации и authorId - идентификационный номер автора.

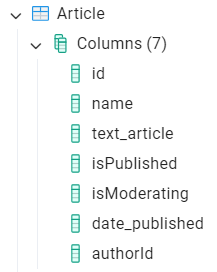


Рис. 2. Таблица Article.

Таблица User содержит информацию о пользователе. Имеет 4 поля: id - идентификационный номер пользователя, login - логин (имя) пользователя, password - пароль, isAdmin - является ли пользователь администратором.

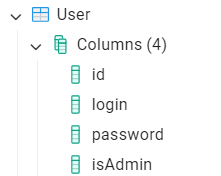


Рис. 3. Таблица User.

# Разработка на Node.js (JavaScript)

Вся серверная часть приложения находится в файле app.js. В самом начале файла происходит подключение различных модулей, используемых по мере разработки приложения.

Далее находятся две функции - is\_auth и is\_auth\_admin. Они нужны для того, чтобы определять, является ли посетитель сайта зарегистрированным пользователем или, во втором случае, администратором.

Ниже находятся методы Express js для обработки get и post запросов, описание которых приведено в таблице 1.

Таблица 1. app.get() и app.post() в app.js

|  |  |  |
| --- | --- | --- |
| Маршрут | Метод | Описание |
| /api/isadmin | get | Получает информацию о том, является ли пользователь администратором, из текущей сессии, и возвращает булево значение в формате json |
| / | get | Возвращает файл index.html. Главная страница сайта |
| /api/articleList | get | Возвращает список всех опубликованных статей в формате json. |
| /api/articleListModerate | get | Возвращает список статей, находящихся на модерации, в формате json. |
| /api/userArticles | get | Возвращает список статей конкретного пользователя (из текущей сессии) в формате json. |
| /article/accept | post | Публикация статьи, т. е. изменение в БД значение её полей isPublished, isModerating и date\_published |
| /article/reject | post | Отклонение статьи, т. е. изменение в БД значение её полей isPublished и isModerating |
| /article/del | post | Удаление статьи |
| /article/edit | get | Возвращает файл article\_create.html. Страница создания статьи. |
| /article/edit | post | Отправка статьи на модерацию (в БД создается новая статья с полями isPublished: false и isModerating: true |
| /article/sendOnModerate | get | Возвращает файл send\_on\_moderate.html. Уведомление пользователя об отправки его статьи на модерацию. |
| /article/moderate | get | Возвращает файл article\_moderate.html. Страница со статьями, находящимися на модерации |
| /article/:pk | get | Возвращает файл article.html. Страница с просмотром одной статьи. |
| /api/concretArticle/:pk | get | Возвращает одну статью по её id в формате json |
| /user/login | get | Возвращает файл login.html. Страница для входа пользователя в систему. |
| /user/login | post | Проверяет на то, верно ли были заполнены пользователем поля в форме логина. Если все верно и такой пользователь существует, обновляется информация о сессии |
| /user/registrate | get | Возвращает файл registrate.html. Страница для регистрации пользователя |
| /user/registrate | post | Регистрирует пользователя. “Засаливает” и хэширует пароль и заносит информацию о новом пользователе в БД |
| /user/logout | get | Выход пользователя из системы. Удаляет всю информацию из сессии. |
| /user | get | Возвращает файл user.html. Страница с профилем пользователя |
| /api/concretUser | get | Возвращает пользователя по его id (из сессии) в формате json |
| /api/author | post | Возвращает пользователя по его id (значение получается при помощи fetch с методом post) в формате json |

Все файлы .html находятся в папке templates, а .css в папке styles внутри проекта.

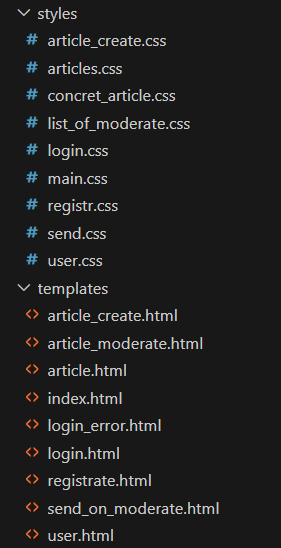


Рис. 4. Папки templates и styles.

# Описание работы сайта

Главная страница сайта выглядит следующим образом:

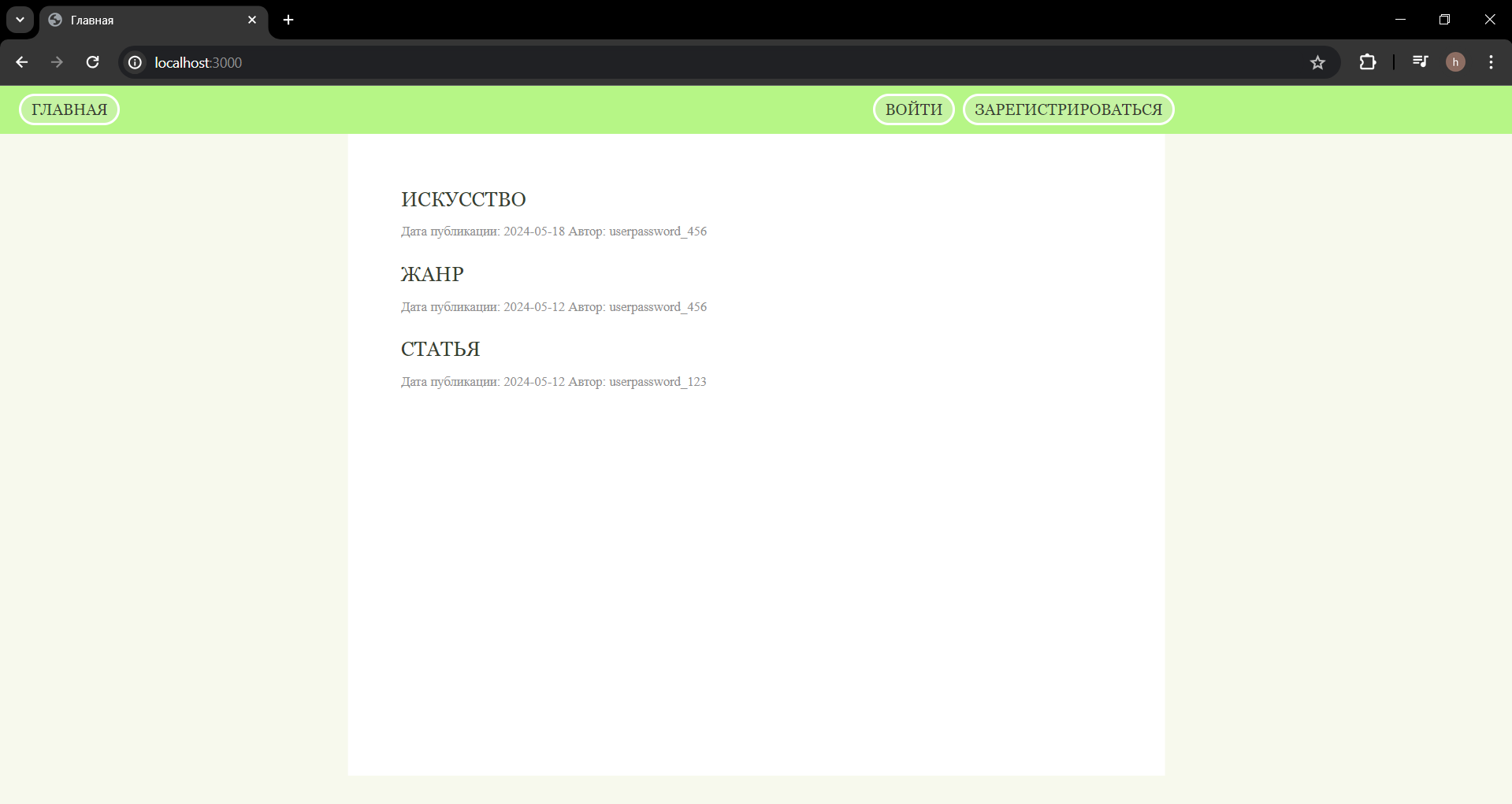


Рис. 5. Главная страница сайта.

Будучи незарегистрированным пользователем, можно только просматривать опубликованные статьи, нажимая на их название на главной странице.

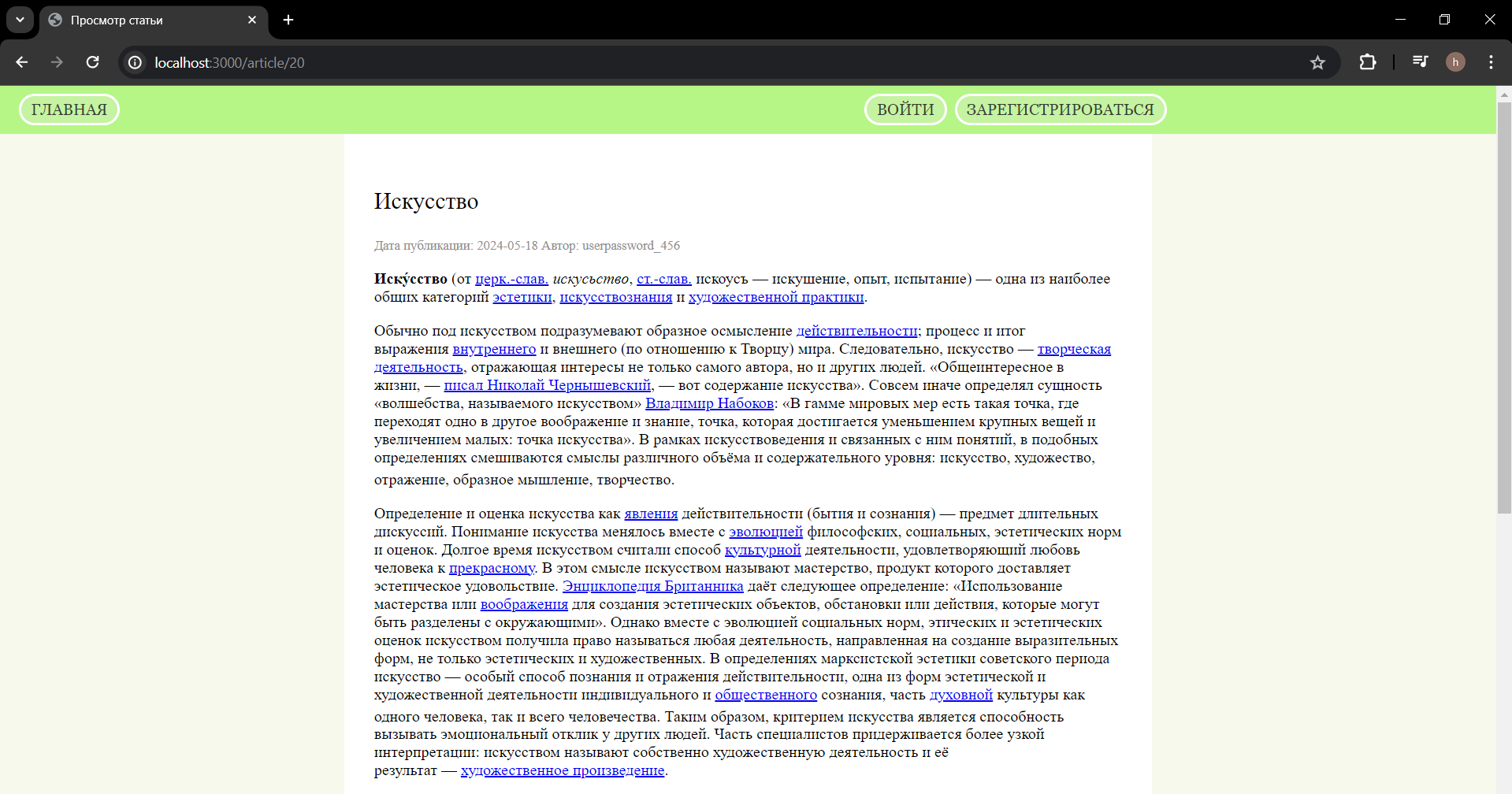


Рис. 6. Просмотр статьи.

Страницы входа и регистрации выглядят соответственно:

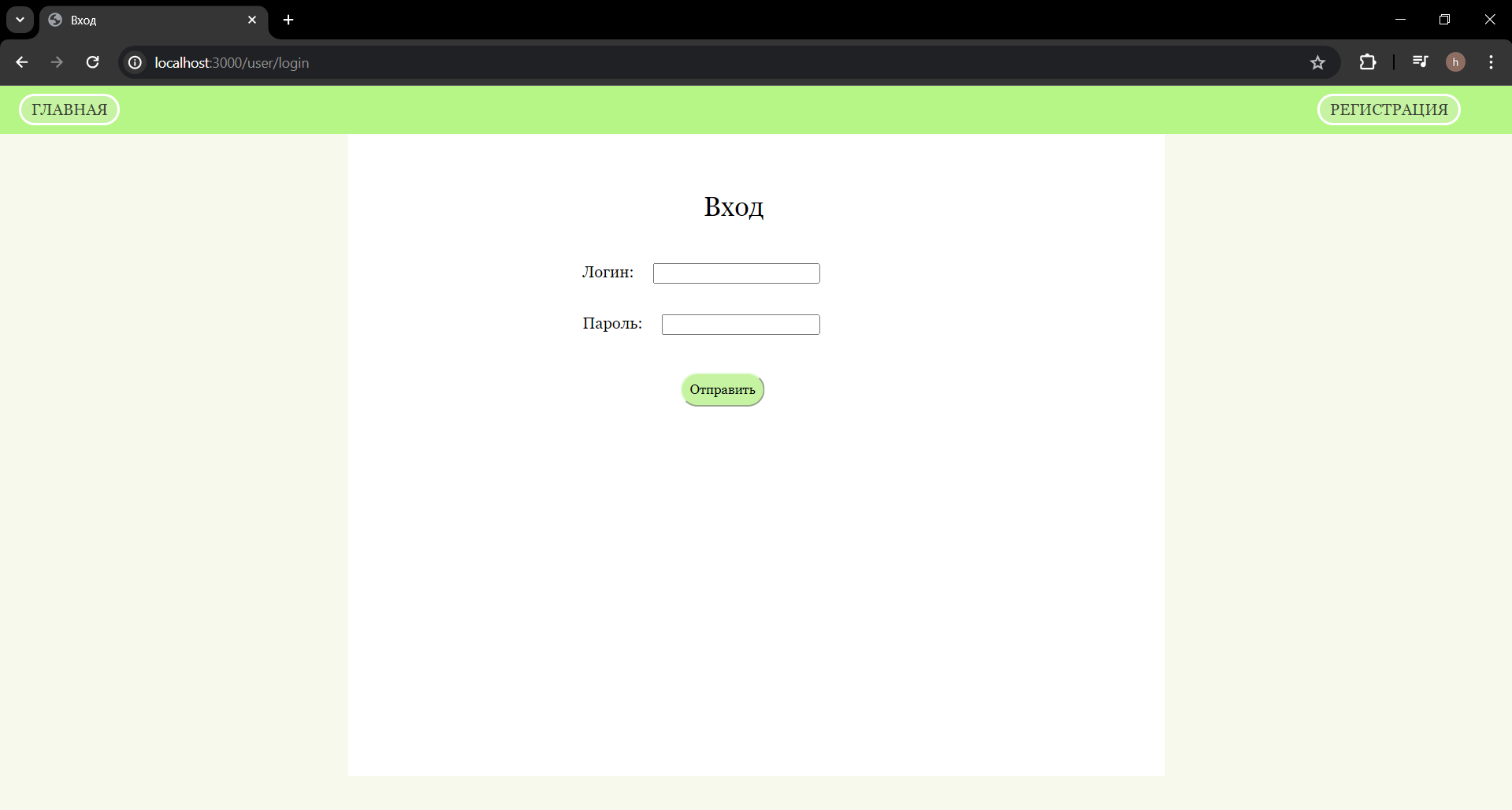


Рис. 7. Вход

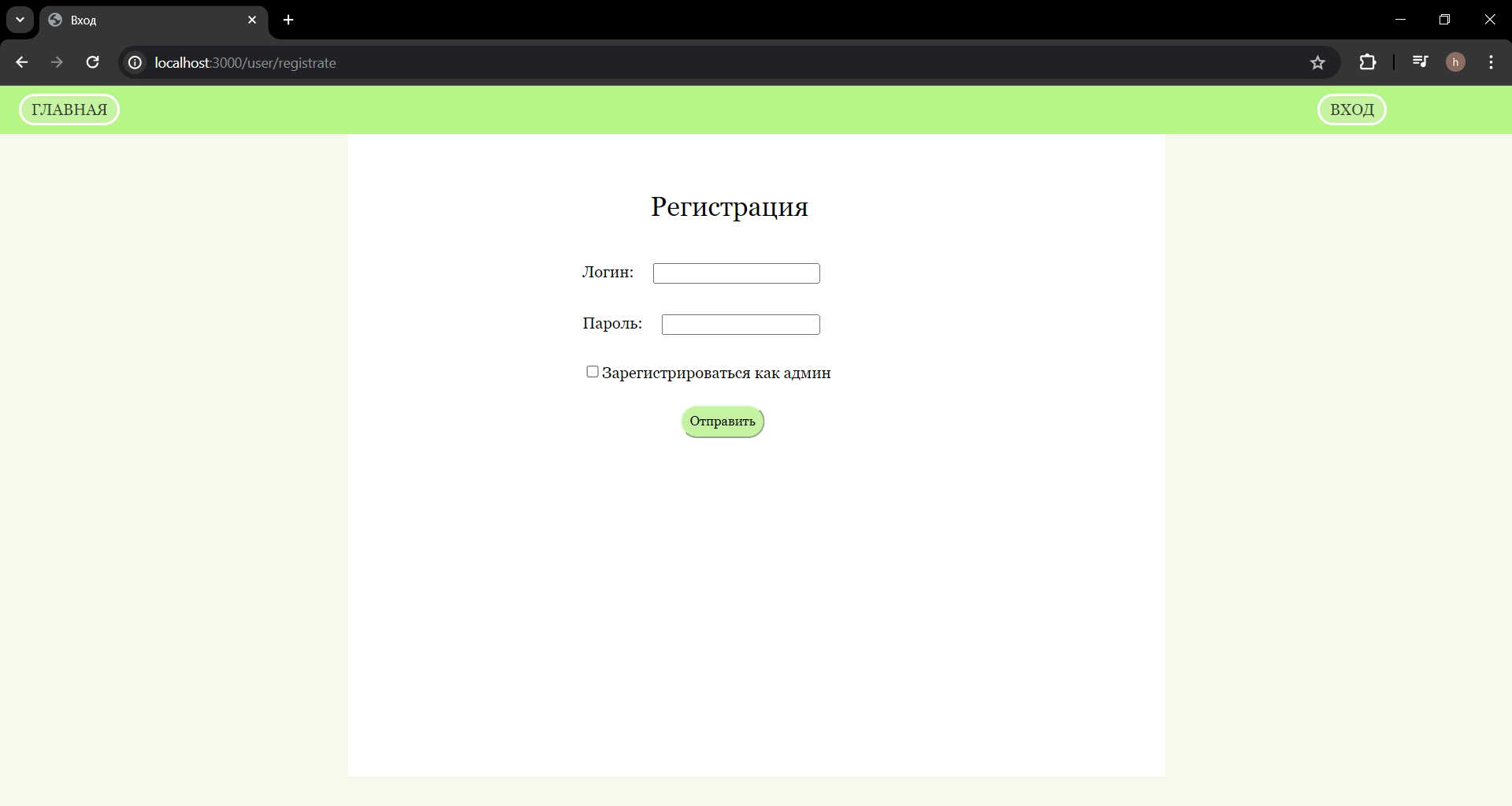


Рис. 8. Регистрация

Если при входе попробовать ввести неверный логин и/или пароль:

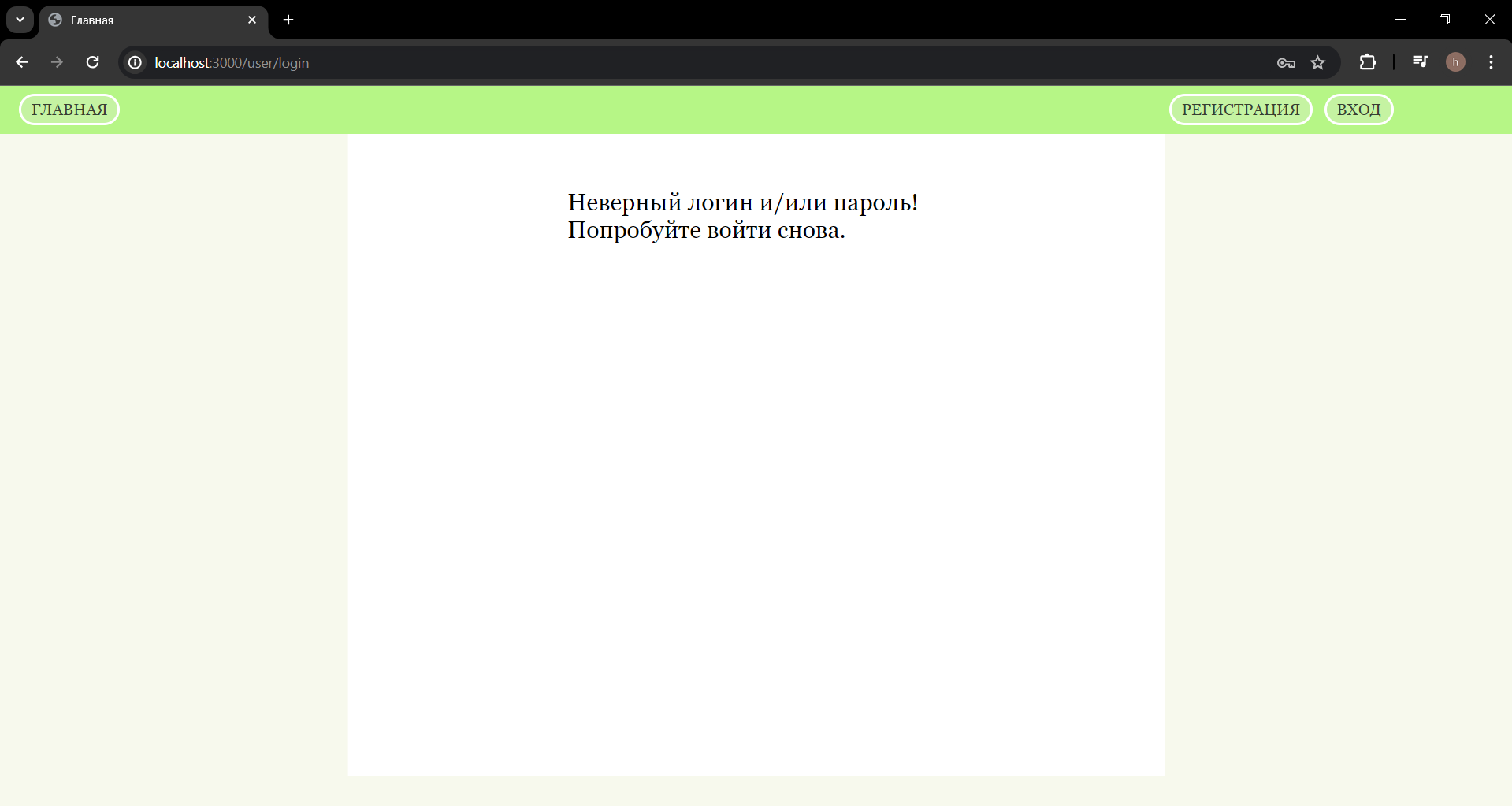


Рис. 9. Неудачная попытка входа.

Зарегистрированный пользователь имеет возможность предложить опубликовать свою статью.

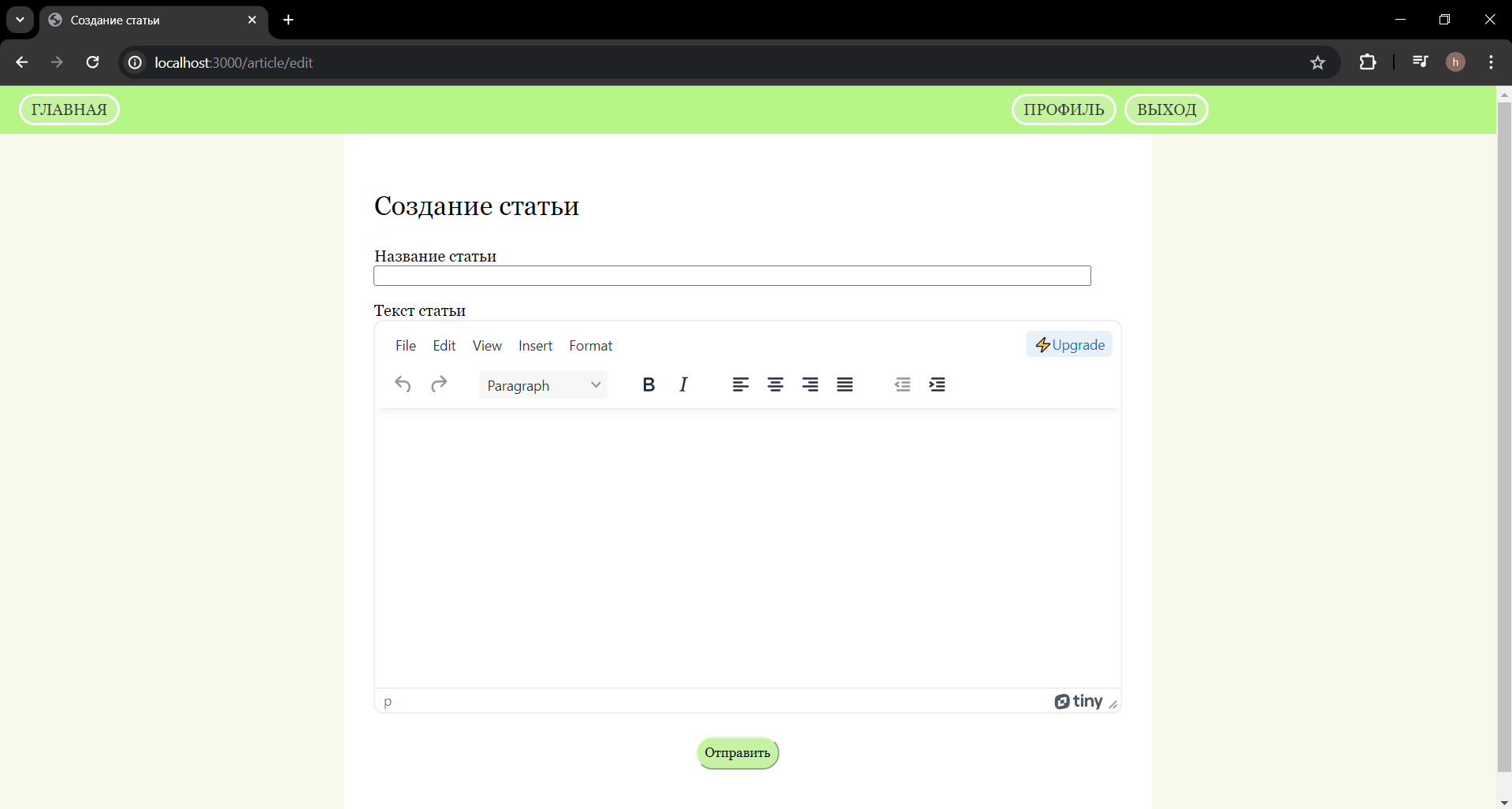


Рис. 10. Страница создания статьи.

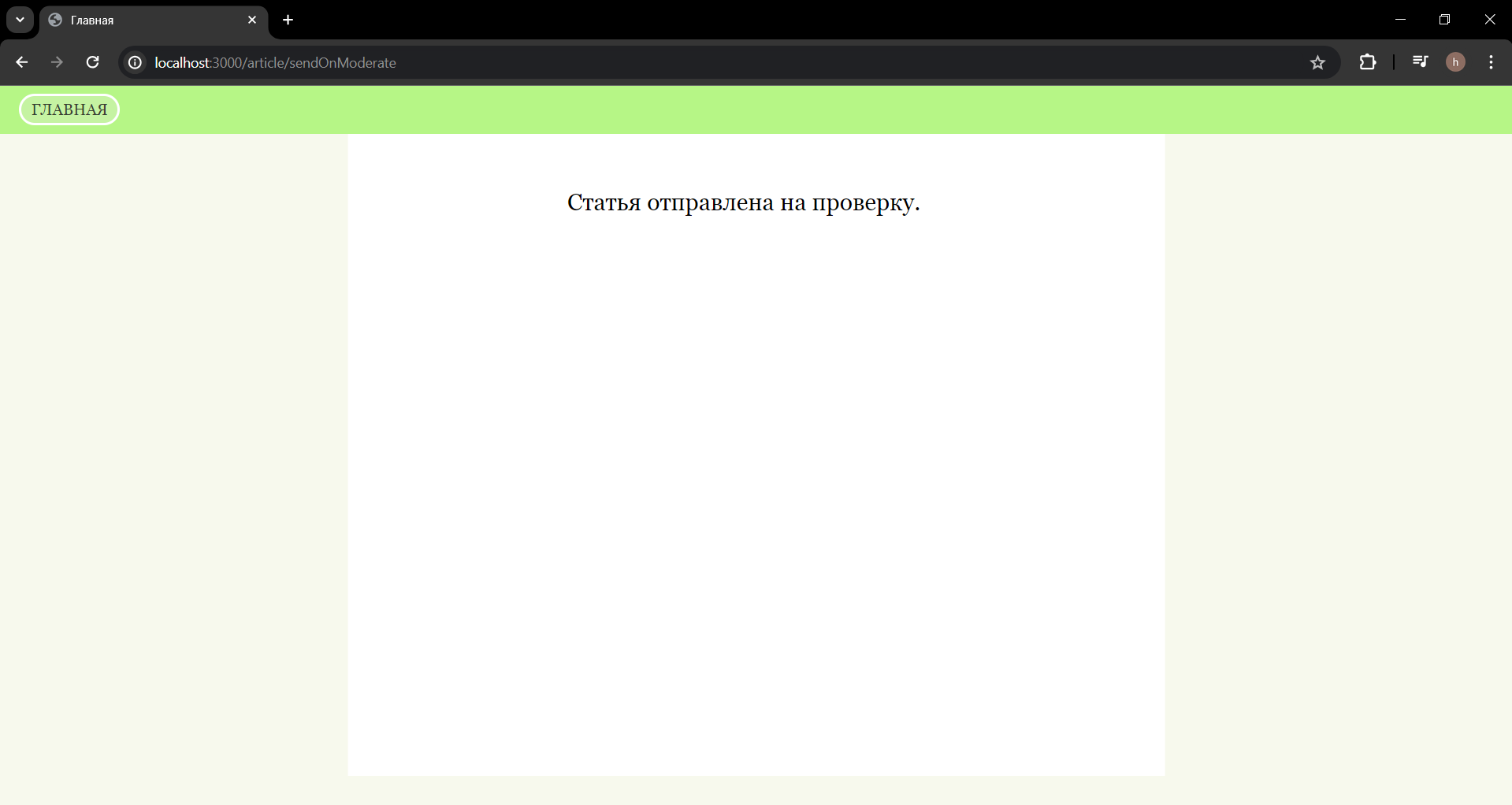


Рис. 11. Отправление статьи на модерацию.

Так же зарегистрированный пользователь может в профиле просмотреть все свои статьи и их статус: опубликована, на проверке или отклонена.

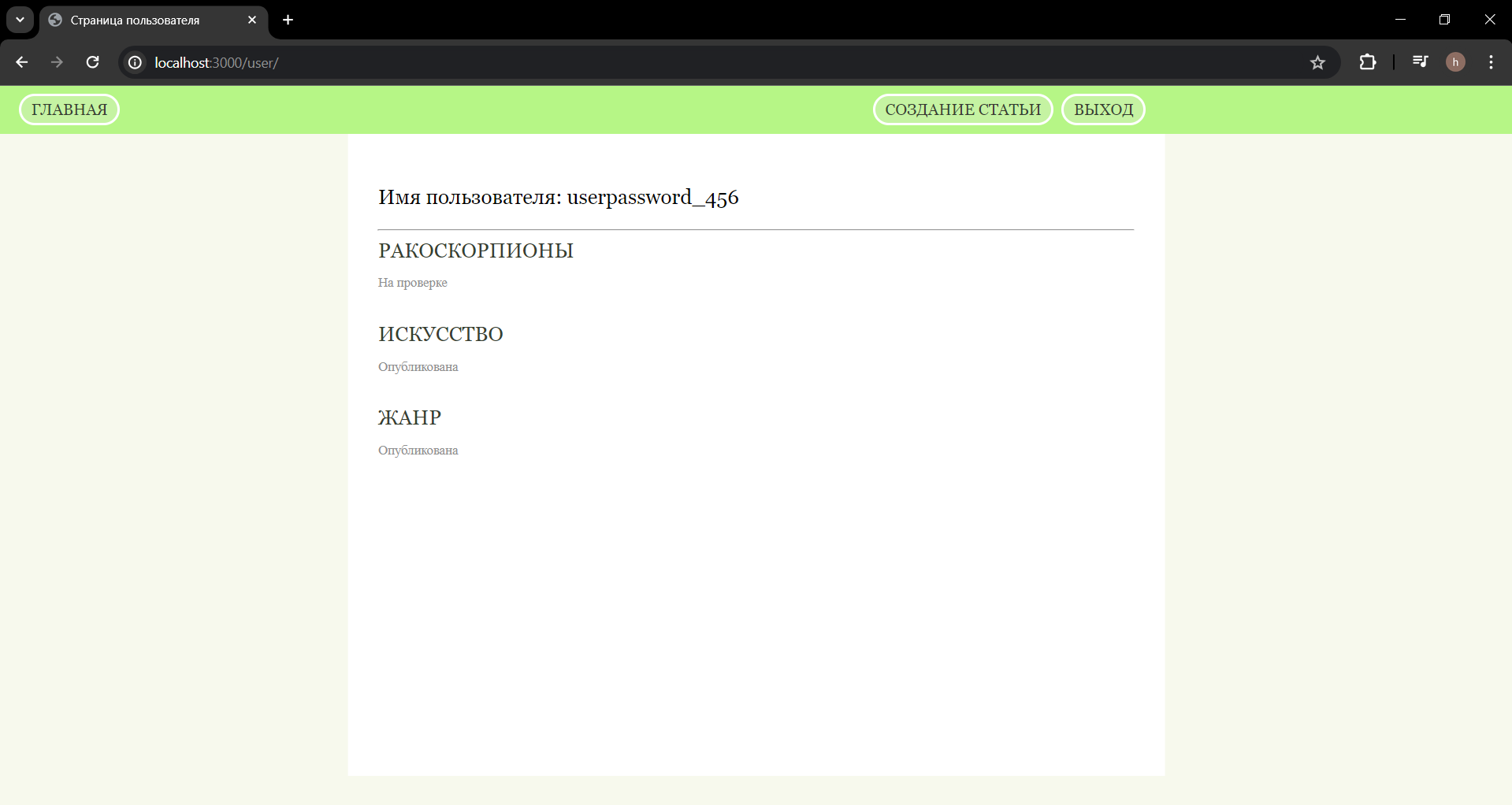


Рис. 12. Профиль пользователя.

Если пользователь является администратором, у него есть доступ к странице со списком статей, находящихся на модерации.

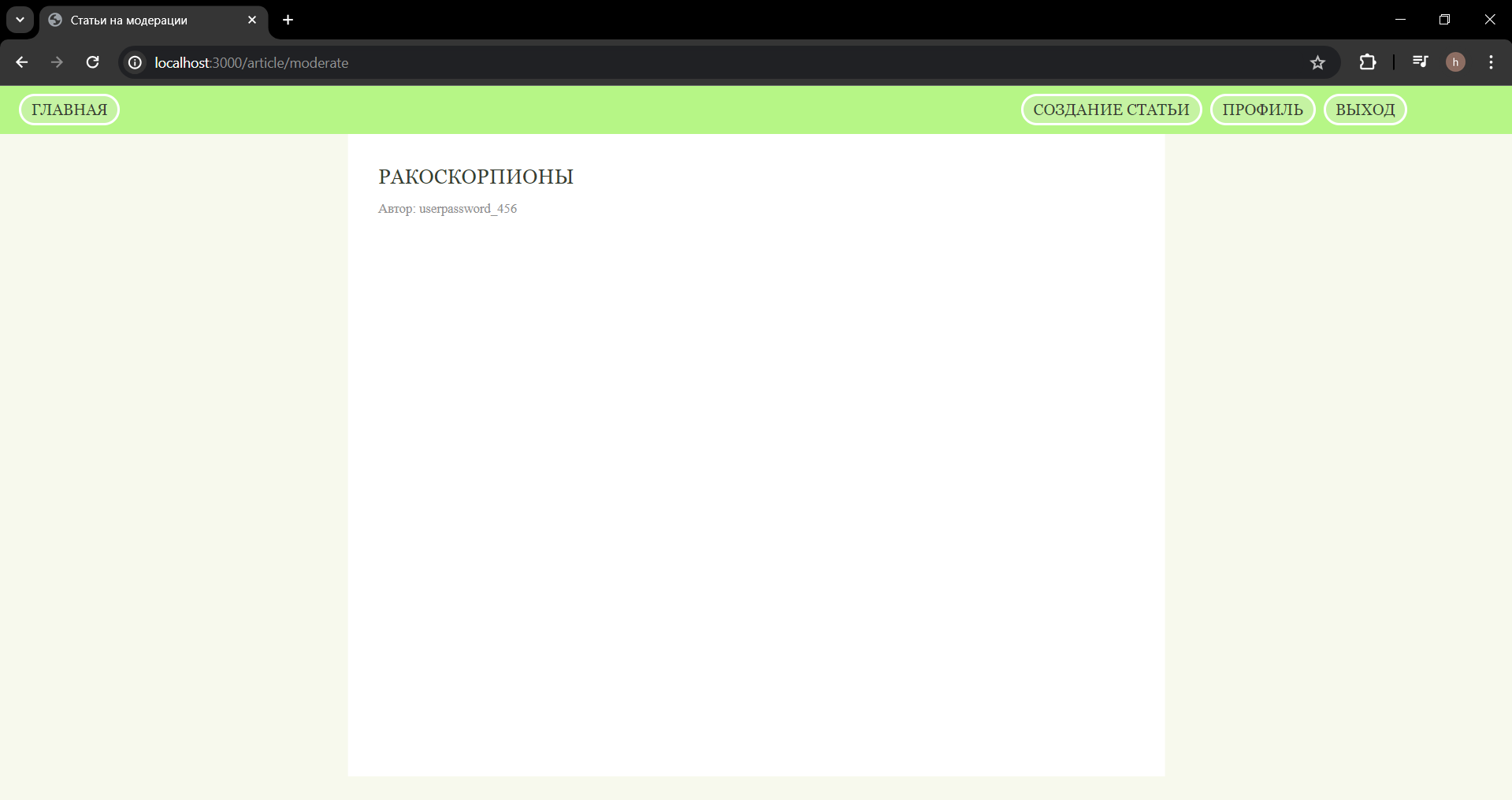


Рис. 13. Статьи, находящие на модерации.

Администратор может опубликовать статью или отклонить.

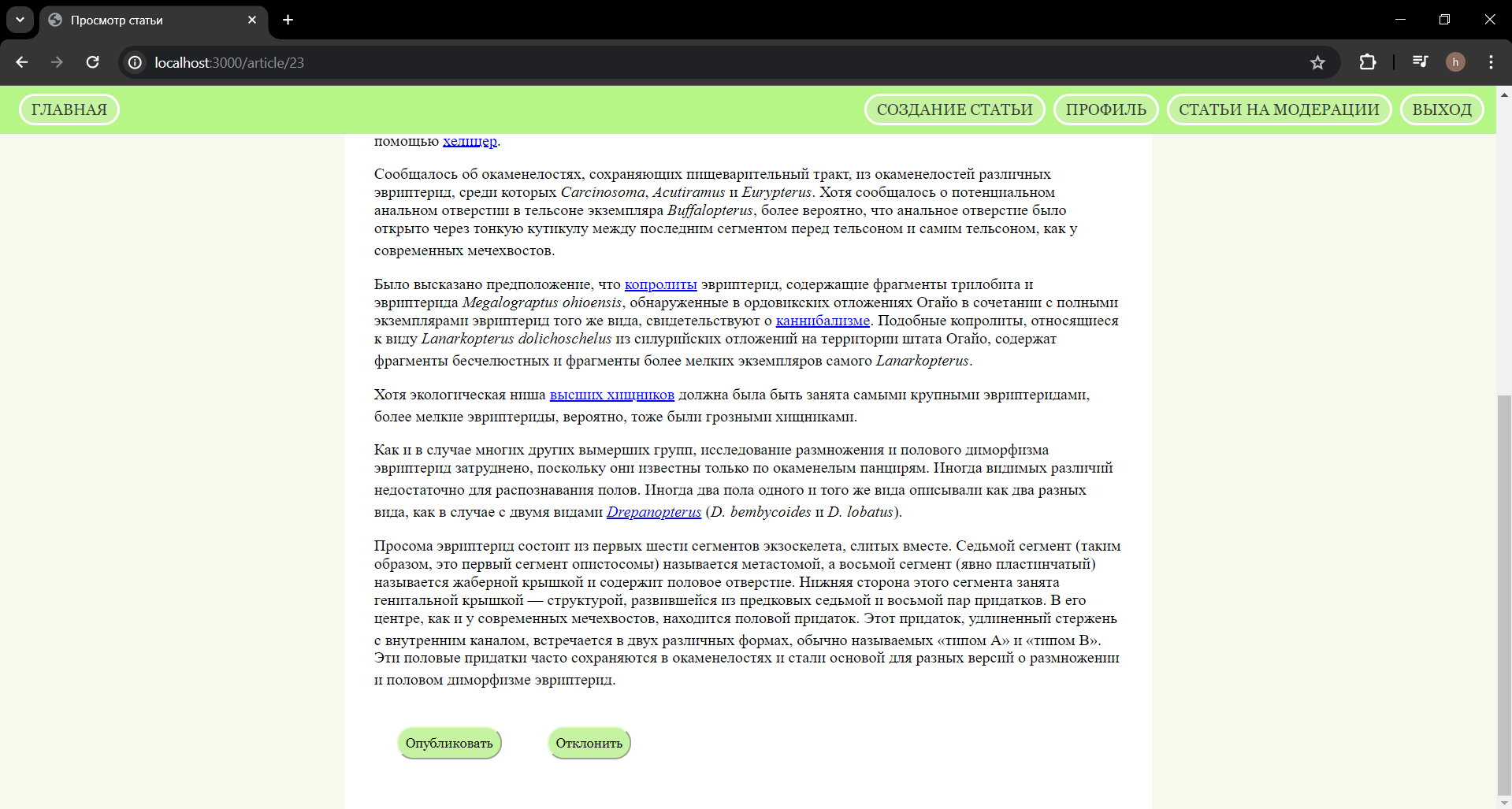


Рис. 14. Возможность опубликовать или отклонить статью.

Так же при просмотре опубликованной статьи администратор имеет возможность её удалить.

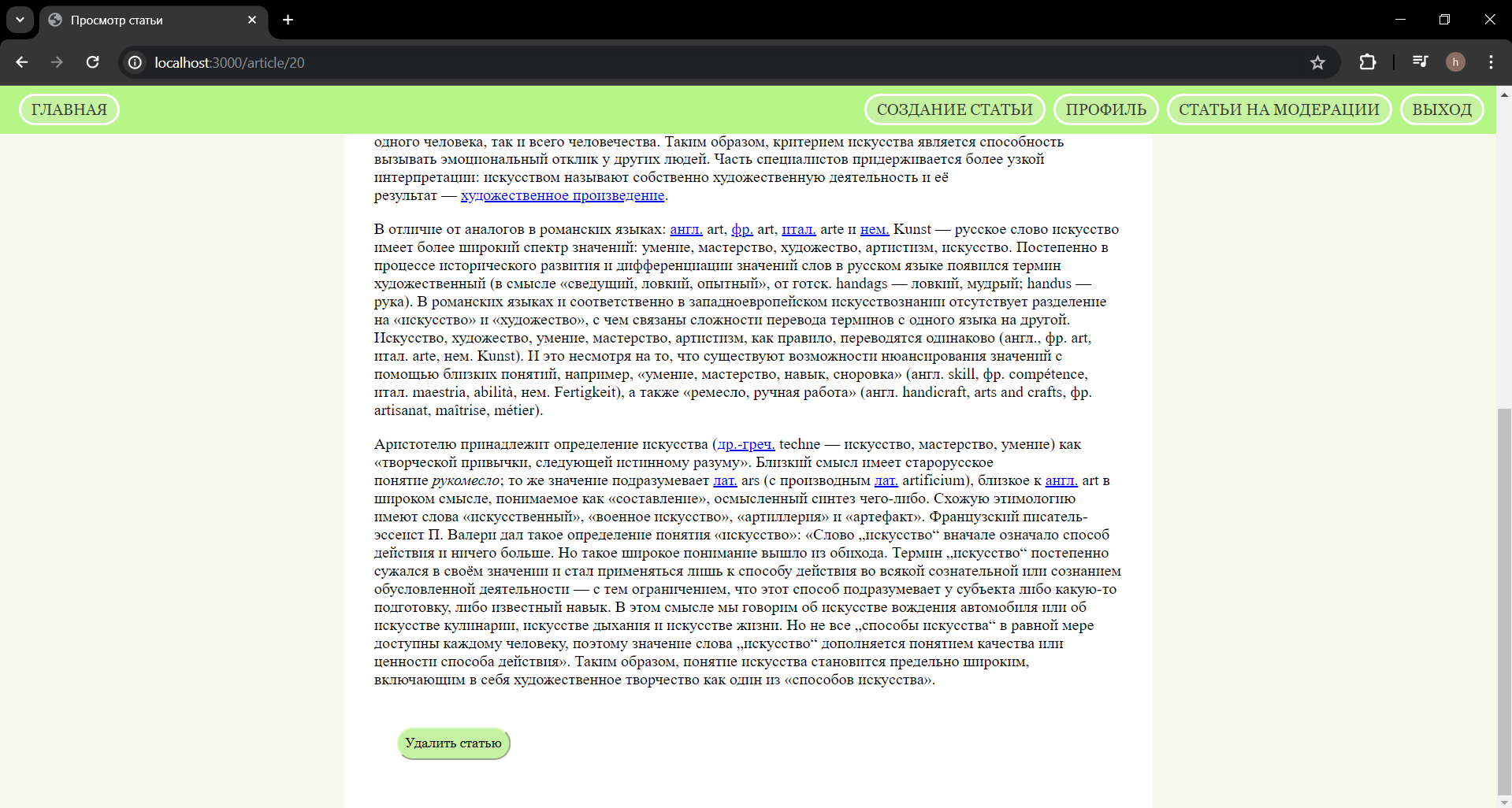


Рис. 15. Возможность удалить статью.

# Приложение А. Выбранный стек технологий.

В выбранный стек технологий входят:

1. Node.js
2. Express
3. Prisma
4. PostgreSQL

В качестве редактора кода будет использован Visual Studio Code. Для удобства работы с PostgreSQL используется приложение pgAdmin.

**Рассмотрим преимущества каждого аспекта выбранного стека технологий.**

1. Node.js

Node.js имеет высокую производительность благодаря использованию асинхронной модели ввода-вывода, что полезно для приложений с высокой нагрузкой. Одной из самых больших положительных черт является экосистема пакетов npm (Node Package Manager) - крупнейший репозиторий инструментов и библиотек для JavaScript, а JavaScript, в свою очередь, это универсальный язык программирования, используемый как на клиентской, так и на серверной части приложения.  
Node.js поощряет использование модульного подхода к разработке. Помимо этого, приложения на этой платформе могут быть легко масштабируемы благодаря асинхронной модели.

1. Express

Express является стандартным каркасом для Node.js при создании веб-приложений и API. Он эффективно использует ресурсы Node, благодаря чему имеет высокую производительность. Ключевой концепцией Express является middleware (связующее программное обесепчение), которое позволяет добавлять дополнительную функциональность к запросам и ответам.

1. Prisma

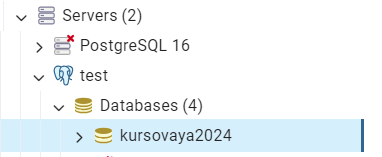
Prisma - это инструмент для работы с базами данных. Prisma отличается простотой в использовании и активным сообществом разработчиков. Кроме того, она поддерживает множество баз данных, таких как MySQL, SQLite, PostgreSQL и другие. Так же, Prisma обеспечивает безопасный доступ к данным благодаря генерации типизированных API на основе схемы базы данных.

1. PostgreSQL

PosgreSQL - объектно-реляционная СУБД, которая является полностью бесплатной с доступным исходным кодом. Обладает модульной архитектурой и обеспечивает надежность и целостность данных. pgAdmin, в свою очередь, сильно облегчает управление БД и мониторинг её промзводительности.

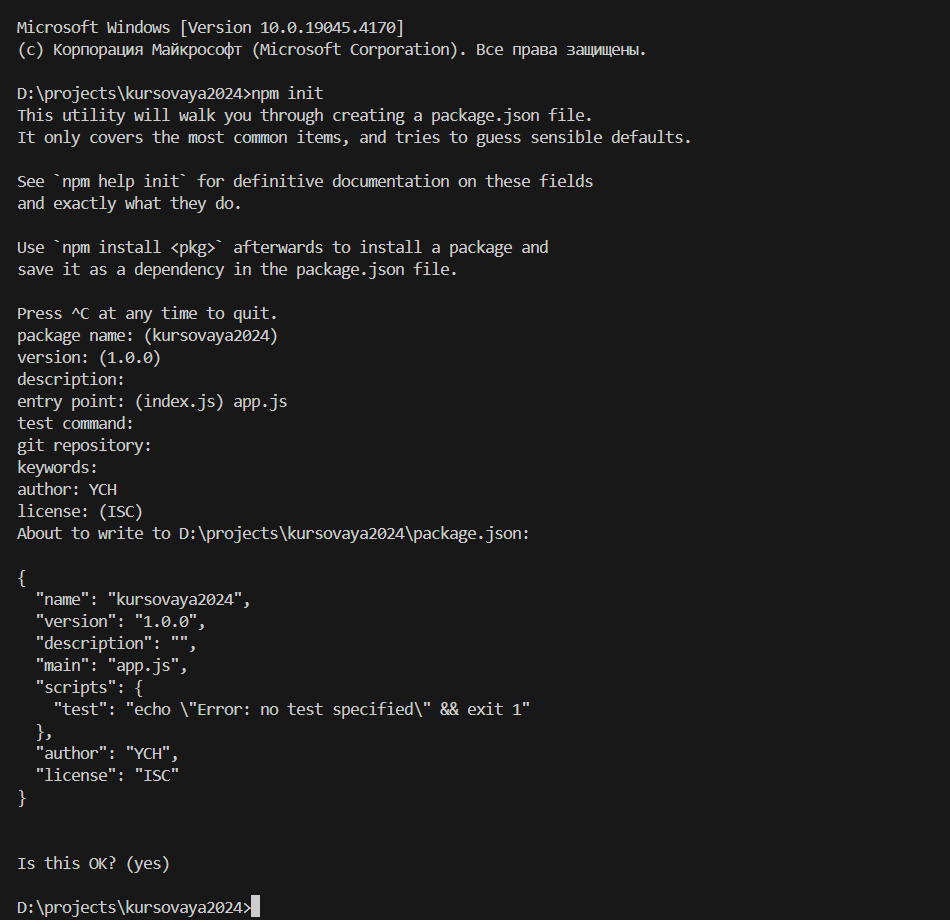
**Настройка выбранного стека.**

**Создание базы данных в pgAdmin**

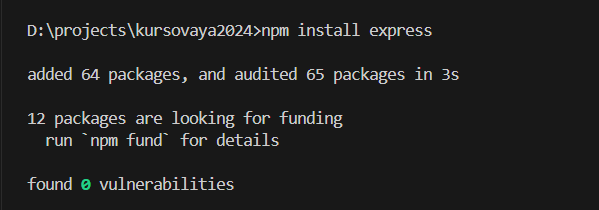


**Установка необходимых библиотек для Node js**

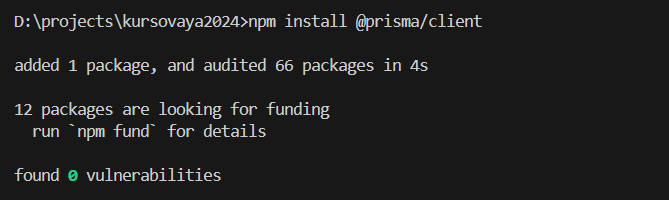
1. Создание проекта Node.js



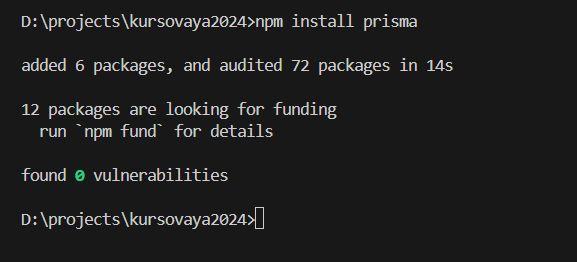
1. Установка Express



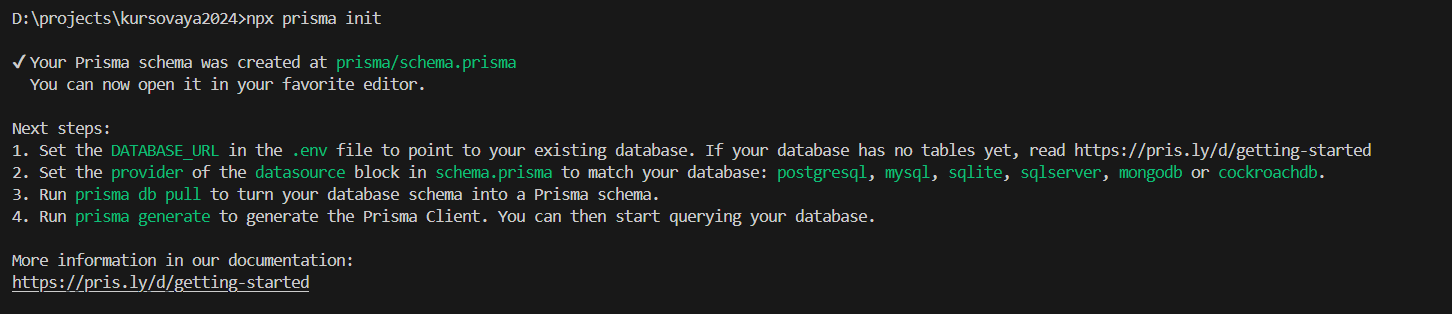
1. Установка клиента Prisma



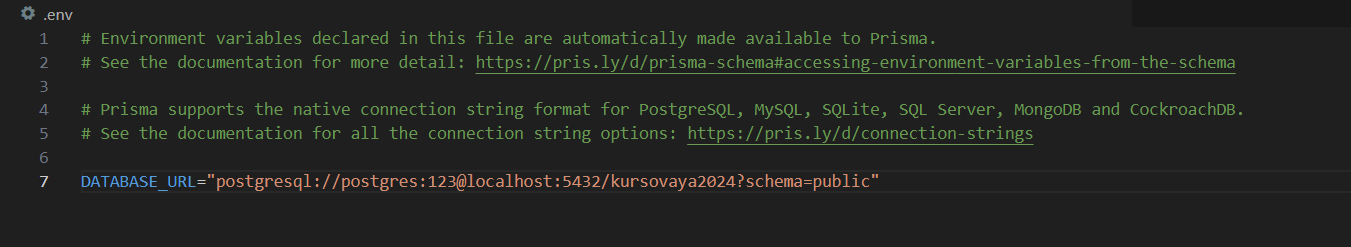
1. Установка Prisma CLI (command line interface)



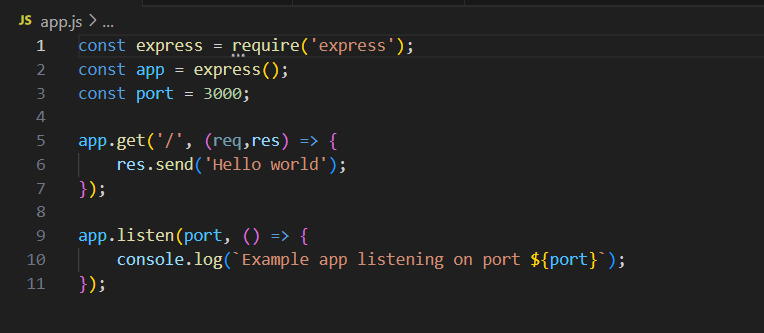
1. Инициализация файла ORM, который в будущем будет содержать набор моделей-таблиц базы данных



1. Настройка переменных окружения



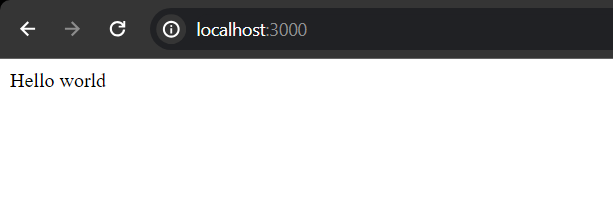
**Простое приложение для проверки работы сервера:**



Запуск приложения

9

Страница в браузере



# Приложение Б. Листинг программы.

**app.js**

const { PrismaClient } = require('@prisma/client');

const prisma = new PrismaClient();

const express = require('express');

const app = express();

const path = require('path');

const port = 3000;

const crypto = require('crypto');

const session = require('express-session');

app.use(express.json());

app.use(express.urlencoded({ extended: false }));

const SAULT = "koshechka";  // для засаливания паролей

app.use('/node\_modules', express.static('node\_modules'));

app.use('/styles', express.static('styles'));

app.use(        // настройка сессии

    session({

        secret: 'kartoshka',

        resave: true,

        saveUninitialized: true,

    })

);

function is\_auth\_admin (req, res, next) {

    let auth = req.session.loggedin;

    let is\_admin = req.session.isAdmin;

    if (auth && is\_admin) {

        next();

    } else {

        res.redirect('/');

    }

}

function is\_auth (req, res, next) {

    let auth = req.session.loggedin;

    if (auth == true) {

        next();

    } else {

        res.redirect('/');

    }

}

app.get('/api/isadmin', async (req, res) => {

    let is\_admin = req.session.isAdmin;

    if (is\_admin == undefined){

        is\_admin = null;

    }

    res.json(is\_admin);

});

app.get('/', (req,res) => {  //главная страница со статьями

    res.sendFile(path.join(\_\_dirname,'/templates') + '/index.html');

});

app.get('/api/articleList', async (req, res) => {

    const articles = await prisma.article.findMany({

        orderBy: { date\_published: 'desc'},

        where: { isPublished: true}

    });

    res.json(articles);

});

app.get('/api/articleListModerate', async (req, res) => {

    const articles = await prisma.article.findMany({

        orderBy: { date\_published: 'desc'},

        where: { isModerating: true}

    });

    res.json(articles);

});

app.get('/api/userArticles', async (req, res) => {

    const articles = await prisma.article.findMany({

        where: { authorId: req.session.userid },

        orderBy: { id: 'desc' }

    });

    res.json(articles);

});

app.post('/article/accept', is\_auth\_admin, async (req,res) => {  //публикация статьи

    let now = new Date();

    let update = await prisma.article.update({

        where: {

            id: req.body.id,

        },

        data: {

            isPublished: true,

            isModerating: false,

            date\_published: now,

        },

    })

    res.json('accept');

});

app.post('/article/reject', is\_auth\_admin, async (req,res) => {  //отклонение статьи

    let reject = await prisma.article.update({

        where: {

            id: req.body.id,

        },

        data: {

            isModerating: false,

            isPublished: false,

        },

    });

    res.json('reject');

});

app.post('/article/del', is\_auth\_admin, async (req,res) => {

    let del = await prisma.article.delete({

        where: {id: req.body.id}

    })

    res.json('delete');

})

app.get('/article/edit', is\_auth, (req,res) => { //создание статьи

    res.sendFile(path.join(\_\_dirname,'/templates') + '/article\_create.html');

});

app.post('/article/edit', is\_auth, async (req,res) => { //отправка на модерацию

    const new\_article = await prisma.article.create({

        data:{

            name: req.body.name\_article,

            text\_article: req.body.text\_article,

            isPublished: false,

            isModerating: true,

            authorId: req.session.userid,

        }

    })

    res.redirect('/article/sendOnModerate');

})

app.get('/article/sendOnModerate', is\_auth, (req, res) => {

    res.sendFile(path.join(\_\_dirname,'/templates') + '/send\_on\_moderate.html');

});

app.get('/article/moderate', is\_auth\_admin, (req,res) => {  //страница админа

    res.sendFile(path.join(\_\_dirname,'/templates') + '/article\_moderate.html');

});

app.get('/article/:pk', (req,res) => { //просмотр статьи + кнопки админа

    res.sendFile(path.join(\_\_dirname,'/templates') + '/article.html');

});

app.get('/api/concretArticle/:pk', async (req,res) => {

    let numOfArticle = req.params['pk'];

    const article = await prisma.article.findFirst({

        where: {id: Number(numOfArticle)}

    });

    res.json(article);

})

app.get('/user/login', (req,res) => {

    res.sendFile(path.join(\_\_dirname,'/templates') + '/login.html');

});

app.post('/user/login', async (req,res) => {

    let username = req.body.username;

    let pass = req.body.pass;

    if(username && pass) {

        let hash\_pass = crypto.createHash('sha256').update(pass + SAULT).digest('hex');

        const current\_user = await prisma.user.findFirst({

            where: {login: username, password: hash\_pass}

        });

        if (current\_user != null) {

            req.session.loggedin = true;

            req.session.isAdmin = current\_user.isAdmin;

            req.session.userid = current\_user.id;

            res.redirect('/');

        } else {

            res.sendFile(path.join(\_\_dirname,'/templates') + '/login\_error.html');

        }

    } else {

        res.redirect('/user/login');

    }

});

app.get('/user/registrate', (req,res) => {

    res.sendFile(path.join(\_\_dirname,'/templates') + '/registrate.html');

});

app.post('/user/registrate', async (req,res) => {

    const check\_name = await prisma.user.findFirst({

        where: {login: req.body.username}

    })

    console.log(check\_name);

    if (check\_name == null){

        let pass = req.body.pass + SAULT;

        let hash\_pass = crypto.createHash('sha256').update(pass).digest('hex');

        const user = await prisma.user.create({

            data:{

                login: req.body.username,

                password: hash\_pass,

                isAdmin: req.body.as\_admin == 'on'

            }

        })

    } else {

        throw "Такое имя пользователя уже существует.";

    }

    res.redirect('/user/login');

});

app.get('/user/logout', (req, res) =>{

    req.session.destroy();

    res.redirect('/');

})

app.get('/user', is\_auth, (req,res) => {  // страница пользователя

    res.sendFile(path.join(\_\_dirname,'/templates') + '/user.html');

});

app.get('/api/concretUser', async (req,res) => {

    let user\_id = req.session.userid;

    const user = await prisma.user.findFirst({

        where: {id: Number(user\_id)}

    });

    res.json(user);

})

app.post('/api/author', async (req, res) => {

    let author\_id = req.body.id;

    const author = await prisma.user.findFirst({

        where: { id: Number(author\_id)}

    });

    res.json(author);

});

app.listen(port, () => {

    console.log(`App listening on port ${port}`);

});

**index.html**

<!DOCTYPE html>

<html lang="ru">

    <head>

        <title>Главная</title>

        <meta charset="utf-8">

        <link rel="stylesheet" href="/styles/main.css">

        <link rel="stylesheet" href="/styles/articles.css">

    </head>

    <body>

        <div id="menu" class="top\_menu">

            <a href="/" class="main">Главная</a>

        </div>

        <div id="back" class="second\_back"></div>

        <script type="module">

            let res\_admin = await fetch(window.location.origin + '/api/isadmin');

            let is\_admin = await res\_admin.json();

            if(is\_admin != null) {

                let create\_article = create\_link('/article/edit', 'Создание статьи');

                menu.append(create\_article);

                let profile = create\_link('/user/', 'Профиль');

                menu.append(profile);

                if(is\_admin) {

                    let moderate = create\_link('/article/moderate', 'Статьи на модерации');

                    menu.append(moderate);

                }

                let logout = create\_link('/user/logout', 'Выход');

                menu.append(logout);

            } else {

                let login = create\_link('/user/login', 'Войти');

                menu.append(login);

                let registrate = create\_link('/user/registrate', 'Зарегистрироваться');

                menu.append(registrate);

            }

            let response = await fetch(window.location.origin + '/api/articleList');

            let listOfArticles = await response.json();

            for (let i=0; i<listOfArticles.length; i++){

                let author\_id = {

                    id: listOfArticles[i].authorId,

                };

                let res\_author = await fetch(window.location.origin + '/api/author', {

                    method: 'POST',

                    headers: {

                        'Content-Type': 'application/json;charset=utf-8'

                    },

                    body: JSON.stringify(author\_id),

                });

                let author = await res\_author.json();

                let dateOfArticle = listOfArticles[i].date\_published.split('T')

                let div = document.createElement('div');

                div.className = "article";

                div.innerHTML = `<a href="/article/${listOfArticles[i].id}" class="concret\_article">` + listOfArticles[i].name + '</a>'+ '<br>'

                 + '<p class="add">Дата публикации: ' + dateOfArticle[0] + ' Автор: ' + author.login + '</p>';

                back.append(div);

            }

            function create\_link (href, text) {

                let elem = document.createElement('a');

                elem.innerText = text;

                elem.className = "another\_links"

                elem.setAttribute('href', href);

                return elem;

            }

        </script>

    </body>

</html>

**article.html**

<!DOCTYPE html>

<html lang="ru">

    <head>

        <title>Просмотр статьи</title>

        <meta charset="utf-8">

        <link rel="stylesheet" href="/styles/main.css">

        <link rel="stylesheet" href="/styles/concret\_article.css">

    </head>

    <body>

        <div id="menu" class="top\_menu">

            <a href="/" class="main">Главная</a>

        </div>

        <div id="back" class="second\_back">

            <div id="buttons"></div>

        </div>

        <script type="module">

            let res\_admin = await fetch(window.location.origin + '/api/isadmin');

            let is\_admin = await res\_admin.json();

            if(is\_admin != null) {

                let create\_article = create\_link('/article/edit', 'Создание статьи');

                menu.append(create\_article);

                let profile = create\_link('/user/', 'Профиль');

                menu.append(profile);

                if(is\_admin) {

                    let moderate = create\_link('/article/moderate', 'Статьи на модерации');

                    menu.append(moderate);

                }

                let logout = create\_link('/user/logout', 'Выход');

                menu.append(logout);

            } else {

                let login = create\_link('/user/login', 'Войти');

                menu.append(login);

                let registrate = create\_link('/user/registrate', 'Зарегистрироваться');

                menu.append(registrate);

            }

            let currPath = window.location.pathname.split('/');

            let apiPath = window.location.origin + '/api/concretArticle';

            let response = await fetch(apiPath + '/' + currPath[2]);

            let concretArticle = await response.json();

            let author\_id = {

                    id: concretArticle.authorId,

                };

            let res\_author = await fetch(window.location.origin + '/api/author', {

                method: 'POST',

                headers: {

                    'Content-Type': 'application/json;charset=utf-8'

                },

                body: JSON.stringify(author\_id),

            });

            let author = await res\_author.json();

            let nameOfArticle = concretArticle.name;

            let dateOfArticle;

            if(concretArticle.date\_published != null) {

                dateOfArticle = concretArticle.date\_published.split('T');

            }

            let textOfArticle = concretArticle.text\_article;

            let addStr;

            if(concretArticle.isPublished == true) {

                addStr = 'Дата публикации: ' + dateOfArticle[0] + ' Автор: ' + author.login;

            } else {

                addStr = 'Статья находится на модерации.' + ' Автор: ' + author.login;

            }

            let div = document.createElement('div');

            div.innerHTML = '<p class="article\_name">' + nameOfArticle + '</p>' + '<p class="add">' + addStr + '</p>' + '<p>' + textOfArticle + '</p>';

            back.prepend(div);

            if(is\_admin && concretArticle.isPublished) {

                let del = document.createElement('button');

                del.innerText = 'Удалить статью';

                del.className = "del";

                del.onclick = del\_article;

                buttons.append(del);

            }

            if (is\_admin && concretArticle.isModerating) {

                let accept = document.createElement('button');

                let reject = document.createElement('button');

                accept.innerText = 'Опубликовать';

                accept.className = 'accept';

                reject.innerText = 'Отклонить';

                reject.className = 'reject';

                accept.onclick = accept\_article;

                reject.onclick = reject\_article;

                buttons.append(accept);

                buttons.append(reject);

            }

            function create\_link (href, text) {

                let elem = document.createElement('a');

                elem.innerText = text;

                elem.className = "another\_links";

                elem.setAttribute('href', href);

                return elem;

            }

            async function accept\_article() {

                let change\_article = {

                    id: concretArticle.id,

                    isPublished: true,

                    isModerating: false

                };

                let res = await fetch(window.location.origin + '/article/accept', {

                    method: 'POST',

                    headers: {

                        'Content-Type': 'application/json;charset=utf-8'

                    },

                    body: JSON.stringify(change\_article),

                });

                window.location.pathname = '/article/moderate';

            }

            async function reject\_article() {

                let change\_article = {

                    id: concretArticle.id

                }

                let res = await fetch(window.location.origin + '/article/reject', {

                    method: 'POST',

                    headers: {

                        'Content-Type': 'application/json;charset=utf-8'

                    },

                    body: JSON.stringify(change\_article),

                });

                window.location.pathname = '/article/moderate';

            }

            async function del\_article() {

                let del\_article = {

                    id: concretArticle.id

                }

                let res = await fetch(window.location.origin + '/article/del', {

                    method: 'POST',

                    headers: {

                        'Content-Type': 'application/json;charset=utf-8'

                    },

                    body: JSON.stringify(del\_article),

                });

                window.location.pathname = '/';

            }

        </script>

    </body>

</html>

**article\_create.html**

<!DOCTYPE html>

<html lang="ru">

    <head>

        <title>Создание статьи</title>

        <meta charset="utf-8">

        <script src="/node\_modules/tinymce/tinymce.min.js" referrerpolicy="origin"></script>

        <link rel="stylesheet" href="/styles/main.css">

        <link rel="stylesheet" href="/styles/article\_create.css">

    </head>

    <body>

        <div id="menu" class="top\_menu">

            <a href="/" class="main">Главная</a>

        </div>

        <div id="back" class="second\_back">

            <p class="title">Создание статьи</p>

            <form method="post">

                <label>Название статьи<br><input type="text" name="name\_article"></label><br>

                <p>

                    <label>Текст статьи<br><textarea id="textarticle" name="text\_article"></textarea></label>

                    <input type="submit" name="Отправить на модерацию" class="button">

                </p>

            </form>

        </div>

        <script type="module">

            tinymce.init({

                selector: 'textarea#textarticle',

                license\_key: 'gpl'

            });

            let res\_admin = await fetch(window.location.origin + '/api/isadmin');

            let is\_admin = await res\_admin.json();

            if(is\_admin != null) {

                let profile = create\_link('/user/', 'Профиль');

                menu.append(profile);

                if(is\_admin) {

                    let moderate = create\_link('/article/moderate', 'Статьи на модерации');

                    menu.append(moderate);

                }

                let logout = create\_link('/user/logout', 'Выход');

                menu.append(logout);

            } else {

                let login = create\_link('/user/login', 'Войти');

                menu.append(login);

                let registrate = create\_link('/user/registrate', 'Зарегистрироваться');

                menu.append(registrate);

            }

            function create\_link (href, text) {

                let elem = document.createElement('a');

                elem.innerText = text;

                elem.className = "another\_links";

                elem.setAttribute('href', href);

                return elem;

            }

        </script>

    </body>

</html>

**send\_on\_moderate.html**

<!DOCTYPE html>

<html lang="ru">

    <head>

        <title>Главная</title>

        <meta charset="utf-8">

        <link rel="stylesheet" href="/styles/main.css">

        <link rel="stylesheet" href="/styles/send.css">

    </head>

    <body>

        <div id="menu" class="top\_menu">

            <a href="/" class="main">Главная</a>

        </div>

        <div id="back" class="second\_back">

            <p>Статья отправлена на проверку.</p>

        </div>

    </body>

</html>

**user.html**

<!DOCTYPE html>

<html lang="ru">

    <head>

        <title>Страница пользователя</title>

        <meta charset="utf-8">

        <link rel="stylesheet" href="/styles/main.css">

        <link rel="stylesheet" href="/styles/user.css">

    </head>

    <body>

        <div id="menu" class="top\_menu">

            <a href="/" class="main">Главная</a>

        </div>

        <div id="back" class="second\_back">

            <!-- <p>Список статей: </p> -->

        </div>

        <script type="module">

            let res\_admin = await fetch(window.location.origin + '/api/isadmin');

            let is\_admin = await res\_admin.json();

            let create\_article = create\_link('/article/edit', 'Создание статьи');

            menu.append(create\_article);

            if(is\_admin) {

                let moderate = create\_link('/article/moderate', 'Статьи на модерации');

                menu.append(moderate);

            }

            let logout = create\_link('/user/logout', 'Выход');

            menu.append(logout);

            let apiPath = window.location.origin + '/api/concretUser';

            let response = await fetch(apiPath);

            let user = await response.json();

            let username = user.login;

            let div = document.createElement('div');

            div.innerHTML = '<p class="name">Имя пользователя: ' + username + '</p><hr>';

            back.prepend(div);

            let res\_articles = await fetch(window.location.origin + '/api/userArticles');

            let listOfArticles = await res\_articles.json();

            console.log(listOfArticles);

            for (let i=0; i<listOfArticles.length; i++){

                let addstr = "";

                if(listOfArticles[i].isPublished){

                    addstr = 'Опубликована';

                }

                if(listOfArticles[i].isModerating){

                    addstr = 'На проверке';

                }

                if(!(listOfArticles[i].isPublished || listOfArticles[i].isModerating)){

                    addstr = 'Отклонена';

                }

                let user\_article = document.createElement('div');

                user\_article.className = "article";

                user\_article.innerHTML = `<a href="/article/${listOfArticles[i].id}" class="concret\_article">` + listOfArticles[i].name + '</a>'+ '<br>' + '<p class="add">' + addstr + '</p>' + '<br>' ;

                back.append(user\_article);

            }

            function create\_link (href, text) {

                let elem = document.createElement('a');

                elem.innerText = text;

                elem.className = "another\_links";

                elem.setAttribute('href', href);

                return elem;

            }

        </script>

    </body>

</html>

**article\_moderate.html**

<!DOCTYPE html>

<html lang="ru">

    <head>

        <title>Статьи на модерации</title>

        <meta charset="utf-8">

        <link rel="stylesheet" href="/styles/main.css">

        <link rel="stylesheet" href="/styles/list\_of\_moderate.css">

    </head>

    <body>

        <div id="menu" class="top\_menu">

            <a href="/" class="main">Главная</a>

        </div>

        <div id="back" class="second\_back"></div>

        <script type="module">

            let res\_admin = await fetch(window.location.origin + '/api/isadmin');

            let is\_admin = await res\_admin.json();

            if(is\_admin != null) {

                let create\_article = create\_link('/article/edit', 'Создание статьи');

                menu.append(create\_article);

                let profile = create\_link('/user/', 'Профиль');

                menu.append(profile);

                let logout = create\_link('/user/logout', 'Выход');

                menu.append(logout);

            } else {

                let login = create\_link('/user/login', 'Войти');

                console.log(login);

                menu.append(login);

                let registrate = create\_link('/user/registrate', 'Зарегистрироваться');

                menu.append(registrate);

            }

            let response = await fetch(window.location.origin + '/api/articleListModerate');

            let listOfArticles = await response.json();

            console.log(listOfArticles);

            if(listOfArticles.length == 0){

                let p = document.createElement('p');

                p.className = 'none';

                p.innerHTML = 'Отсутствуют статьи на модерации';

                back.append(p);

            } else {

                for (let i=0; i<listOfArticles.length; i++){

                    let author\_id = {

                        id: listOfArticles[i].authorId,

                    };

                    let res\_author = await fetch(window.location.origin + '/api/author', {

                        method: 'POST',

                        headers: {

                            'Content-Type': 'application/json;charset=utf-8'

                        },

                        body: JSON.stringify(author\_id),

                    });

                    let author = await res\_author.json();

                    let div = document.createElement('div');

                    div.className = "article";

                    div.innerHTML = `<a href="/article/${listOfArticles[i].id}" class="concret\_article">` + listOfArticles[i].name + '</a>'+ '<br>' + '<p class="add">Автор: ' + author.login + '</p>';

                    back.append(div);

                }

            }

            function create\_link (href, text) {

                let elem = document.createElement('a');

                elem.innerText = text;

                elem.className = 'another\_links';

                elem.setAttribute('href', href);

                return elem;

            }

        </script>

    </body>

</html>

**login\_error.html**

<!DOCTYPE html>

<html lang="ru">

    <head>

        <title>Главная</title>

        <meta charset="utf-8">

        <link rel="stylesheet" href="/styles/main.css">

        <link rel="stylesheet" href="/styles/send.css">

    </head>

    <body>

        <div id="menu" class="top\_menu">

            <a href="/" class="main">Главная</a>

            <a href="/user/registrate" class="another\_links">Регистрация</a>

            <a href="/user/login" class="another\_links">Вход</a>

        </div>

        <div id="back" class="second\_back">

            <p>Неверный логин и/или пароль!<br>

                Попробуйте войти снова.</p>

        </div>

    </body>

</html>

**article\_create.css**

.another\_links {

    position: relative;

    right: -60%;

}

label {

    font-family:Georgia, 'Times New Roman', Times, serif;

}

input[type="text"] {

    width: 95%;

}

.button {

    position: relative;

    left: 40%;

    margin: 3%;

    padding: 1%;

    border-radius: 20px;

    border-color: rgb(229, 250, 215);

    background-color: rgb(197, 243, 162);

    font-family:Georgia, 'Times New Roman', Times, serif;

}

.title {

    font-size: 170%;

    font-family:Georgia, 'Times New Roman', Times, serif;

}

**articles.css**

.another\_links {

    position: relative;

    right: -50%;

}

.concret\_article {

    text-decoration: none;

    font-family:Georgia, 'Times New Roman', Times, serif;

    font-size: 130%;

    text-transform: uppercase;

}

.article {

    margin: 3%;

}

.add {

    font-size: smaller;

    color:rgb(144, 144, 144);

}

**concret\_article.css**

.another\_links {

    position: relative;

    right: -50%;

}

.del, .accept, .reject {

    margin: 3%;

    padding: 1%;

    border-radius: 20px;

    border-color: rgb(229, 250, 215);

    background-color: rgb(197, 243, 162);

    font-family:Georgia, 'Times New Roman', Times, serif;

}

.add {

    font-size: smaller;

    color:rgb(144, 144, 144);

}

.article\_name {

    font-size: 150%;

}

**list\_of\_moderate.css**

.another\_links {

    position: relative;

    right: -60%;

}

.concret\_article {

    text-decoration: none;

    font-family:Georgia, 'Times New Roman', Times, serif;

    font-size: 130%;

    text-transform: uppercase;

}

.add {

    font-size: smaller;

    color:rgb(144, 144, 144);

}

.none {

    font-family:Georgia, 'Times New Roman', Times, serif;

    font-size: 150%;

    position: relative;

    left: 25%;

}

**login.css**

.another\_links {

    position: relative;

    right: -80%;

}

form {

    position: relative;

    left: 27%;

}

.field {

    margin: 2%;

}

.button {

    position: relative;

    left: 10%;

    margin: 3%;

    padding: 1%;

    border-radius: 20px;

    border-color: rgb(229, 250, 215);

    background-color: rgb(197, 243, 162);

    font-family:Georgia, 'Times New Roman', Times, serif;

}

label, p {

    font-family:Georgia, 'Times New Roman', Times, serif;

}

p {

    position: relative;

    left: 16%;

    font-size: 170%;

}

input[type="password"] {

    width: 20%;

}

**main.css**

.top\_menu {

    padding: 1%;

    background-color: rgb(182, 246, 134);

    position: sticky;

    top: 0;

    z-index: 2;

}

body {

    margin: 0%;

    background-color: rgb(247, 249, 237);

}

html, body {

    height: 100%;

}

.main, .another\_links {

    text-decoration: none;

    font-family:Georgia, 'Times New Roman', Times, serif;

    padding: 5px;

    padding-left: 10px;

    padding-right: 10px;

    border: solid 3px rgb(255, 255, 255);

    border-radius: 20px;

    background-color: rgb(197, 243, 162);

    margin: 4px;

    text-transform: uppercase;

}

.main:link, .another\_links:link, .concret\_article:link {

    color:rgb(56, 63, 50);

}

.main:visited, .another\_links:visited, .concret\_article:visited {

    color:rgb(56, 63, 50);

}

.second\_back {

    background-color: white;

    width: 50%;

    padding: 2%;

    position: relative;

    left: 50%;

    transform: translate(-50%, 0);

    min-height: 80%;

    z-index: 1;

}

**registr.css**

p {

    position: relative;

    left: 9%;

    font-size: 170%;

}

input[type="checkbox"] {

    margin-top: 2%;

}

**send.css**

p {

    font-family:Georgia, 'Times New Roman', Times, serif;

    font-size: 150%;

    position: relative;

    left: 25%;

}

.another\_links {

    position: relative;

    right: -70%;

}

**user.css**

.another\_links {

    position: relative;

    right: -50%;

}

.concret\_article {

    text-decoration: none;

    font-family:Georgia, 'Times New Roman', Times, serif;

    font-size: 130%;

    text-transform: uppercase;

}

.add {

    font-size: smaller;

    color:rgb(144, 144, 144);

}

.name {

    font-family:Georgia, 'Times New Roman', Times, serif;

    font-size: 130%;

}