

Аннотация

В данной пояснительной записке содержится проектная документация веб-сайта для многопользовательского игрового проекта с возможностью регистрации личного аккаунта, разработанного в рамках курсового проекта по дисциплине «Веб-технологии».

Во введении обозначена тема курсового проекта, проблема, которая должна быть решена в результате разработки, определены цели и задачи.

В аналитической части производится исследование предметной области, выделение сущностей и связей между ними.

В основной части документа дается описательная проектных и технических решений поставленных задач, в том числе, структура базы данных, архитектура серверной и клиентской частей веб-сайта. После описания проекта, дается информация о деталях реализации, а также приводятся результаты тестирования разработанной системы.

В заключении делается вывод о результатах проделанной работы и дается оценка разработанного проекта.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ	5
ВВЕДЕНИЕ.....	6
1 АНАЛИТИЧЕСКАЯ ЧАСТЬ	8
1.1 Анализ предметной области	8
1.2 Анализ системы прав доступа «LuckPerms»	9
2 РАЗРАБОТКА БАЗЫ ДАННЫХ.....	11
2.1 Разработка схемы базы данных	11
2.2 Интеграция базы данных «LuckPerms»	12
3 РАЗРАБОТКА ВЕБ-САЙТА.....	15
3.1 Функция регистрации пользователей	15
3.2 Функция аутентификации	16
3.3 Функция восстановления пароля.....	17
3.4 Функция подтверждения e-mail.....	18
3.5 Реализация компонентного подхода Vue.js на основе переводимых шаблонов Twig.....	20
3.6 Реализация WYSISWYG-редактора новостной записи	22
3.7 Интеграция веб-ориентированного редактора «LuckPermsWeb»	22
3.8 Интеграция системы прав доступа в систему авторизации Symfony	23
4 ТЕСТИРОВАНИЕ ВЕБ-САЙТА	26
4.1 Главная страница	26
4.2 Регистрация пользователя.....	28
4.3 Аутентификация пользователя	31

4.4 Восстановление пароля	33
4.5 Редактор прав доступа.....	34
4.6 Редактор новостей.....	35
ЗАКЛЮЧЕНИЕ	37
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....	38
ПРИЛОЖЕНИЕ А	39
ПРИЛОЖЕНИЕ Б.....	42

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

GUID	– Globally Unique Identifier (рус. Глобальный уникальный идентификатор)
IDE	– Integrated development environment (рус. Интегрированная среда разработки)
LAMP	– Linux, Apache, MySQL, PHP
MVC	– Model-View-Controller (рус. Модель-Представление-Контроллер)
ORM	– Object-Relational Mapping (рус. Объектно-реляционное отображение)
SPA	– Single-Page Application (рус. Одностраничное приложение)
UUID	– Universally Unique Identifier (рус. Универсальный уникальный идентификатор)
XAMPP	– X(cross-platform), Apache, MySQL, PHP, Perl
ПО	– программное обеспечение
ЯП	– язык программирования

ВВЕДЕНИЕ

Актуальность темы. Веб-сайт является неотъемлемой частью современного проекта в любой сфере деятельности. Для многопользовательских игр веб-сайт может выполнять различные функции, в частности, новостную. Разработка и освещение обновлений является важным методом удержания игроков, а также привлечения старых игроков, потерявших интерес к проекту по какой-либо причине.

Кроме этого, крупные игровые системы требуют все большего количества инструментов для их администрирования. Это особенно заметно в случае горизонтального масштабирования серверной части. Существенно упростить процесс администрирования могут позволить современный веб-ориентированные системы с графическим интерфейсом, которые могут быть встроены в административную часть веб-сайта.

Цель и задачи работы. Целью данного курсового проекта является проектирование и разработка новостного веб-сайта с возможностью регистрации личного аккаунта для игрового проекта «Cerebus Network».

Для достижения поставленной цели необходимо выполнить следующие задачи:

- 1) проанализировать предметную область и выделить в ней сущности и связи между ними;
- 2) построить логическую модель данных и разработать схему БД;
- 3) разработать новостной раздел веб-сайта и личный кабинет;
- 4) интегрировать подсистему управления правами доступа «LuckPerms» в систему аккаунтов веб-сайта.

Практическое значение работы. Новостной раздел веб-сайта позволит публиковать последние изменения и обновления на проекте для своевременного ознакомления с этой информацией конечными пользователями.

Раздел личного кабинета в совокупности с гибкой системой управления правами доступа позволит в дальнейшем реализовать дополнительный функционал, непосредственно связанный с внутриигровыми механиками.

Административный раздел даст возможность управления компонентами игровой платформы и контроля за показателями активности. Модульная структура позволит реализовывать дополнительный функционал по мере появления новых потребностей.

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Анализ предметной области

В рамках данного курсового проекта ставится задача разработки новостного раздела веб-сайта и системы аккаунтов. Так как добавление новостных записей подразумевается только от лица администраторов, то объекты подсистемы новостей будут связаны с объектом пользователя только с помощью идентификатора (связь один ко многим, один администратор может создавать множество новостей), поэтому рассмотрим эти две подсистемы отдельно.

Сущности новостной записи (`blog_post`) идентифицируется порядковым номером. Свойствами новости являются:

- идентификатор автора (связь с сущностью пользователя);
- заголовок;
- содержание (`content`);
- заглавное изображение;
- состояние (черновик, опубликована, скрыта);
- дата создания;
- дата обновления;
- дата публикации.

Также объект новости должен быть связан с изображениями, которые использованы в его содержании, на уровне модели. Этого можно достичь путем введения сущности использованного загруженного ресурса (`referenced_upload`). Сущности `blog_post` и `referenced_upload` связаны отношение многие ко многим – одно изображение может быть использовано в нескольких новостях и в одной новости может быть использовано несколько изображений.

Раздел личного кабинета подразумевает наличие сущности пользователь (`user`). Для идентификации объектов используется GUID (UUID версии 4).

Данный тип идентификатора выбран с целью упрощения интеграции с существующей игровой платформой.

Кроме уникального идентификатора, для объектов сущности user введено два альтернативных ключа – никнейм (внутриигровое имя пользователя) и e-mail.

Сущность пользователя содержит следующие свойства:

- уникальный идентификатор;
- никнейм;
- e-mail;
- пароль (в виде хэша);
- дата регистрации;
- дата подтверждения e-mail;
- уровень лицензии (свойство игровой платформы).

1.2 Анализ системы прав доступа «LuckPerms»

Система LuckPerms вводит следующие сущности: группа прав доступа (роль), права доступа пользователя, пользователь (игрок, player), трек (track).

Логика определение прав конкретного пользователя строится на основе идентификаторов прав доступа. Идентификатором является строка заданного формата (<раздел>[.<раздел>], где раздел – наименование раздела в функциональности системы или *, обозначающая любой раздел), определяющая какое-либо действие.

Каждому идентификатору соответствует набор полей:

- значение (истина – разрешено, ложь – запрещено);
- срок истечения;
- контексты;
- сервер (специальный контекст);
- мир (специальный контекст).

Наличие свойства контекстов позволяет задавать различные значения прав доступа в зависимости от внешних свойств пользователя.

Также, существует специальный раздел прав доступа – «group». Его использование позволяет назначить пользователю права какой-либо роли (группы прав доступа). Например, наличие права group.admin назначает пользователю все права группы с именем admin.

Сущность track определяет иерархию ролей, назначение которых зависит от внешних факторов. Например, повышение игрового уровня пользователя.

Сущность пользователя в рамках подсистемы «LuckPerms» и сущность пользователя в основной предметной области представляют описание одного и того же реального объекта, поэтому целесообразно их объединить. Техническое решение этой проблемы представлено в разделе 3.2.

2 РАЗРАБОТКА БАЗЫ ДАННЫХ

2.1 Разработка схемы базы данных

При разработке схемы базы данных был использован подход последовательного создания миграций с применением инструмента «Liquibase Community». Данный подход позволяет хранить все изменения схемы базы данных в стандартных системах контроля версий для исходного кода. За счет обеспечения централизованного источника информации о текущей схеме БД значительно упрощается её изменение в процессе эволюции программной системы.

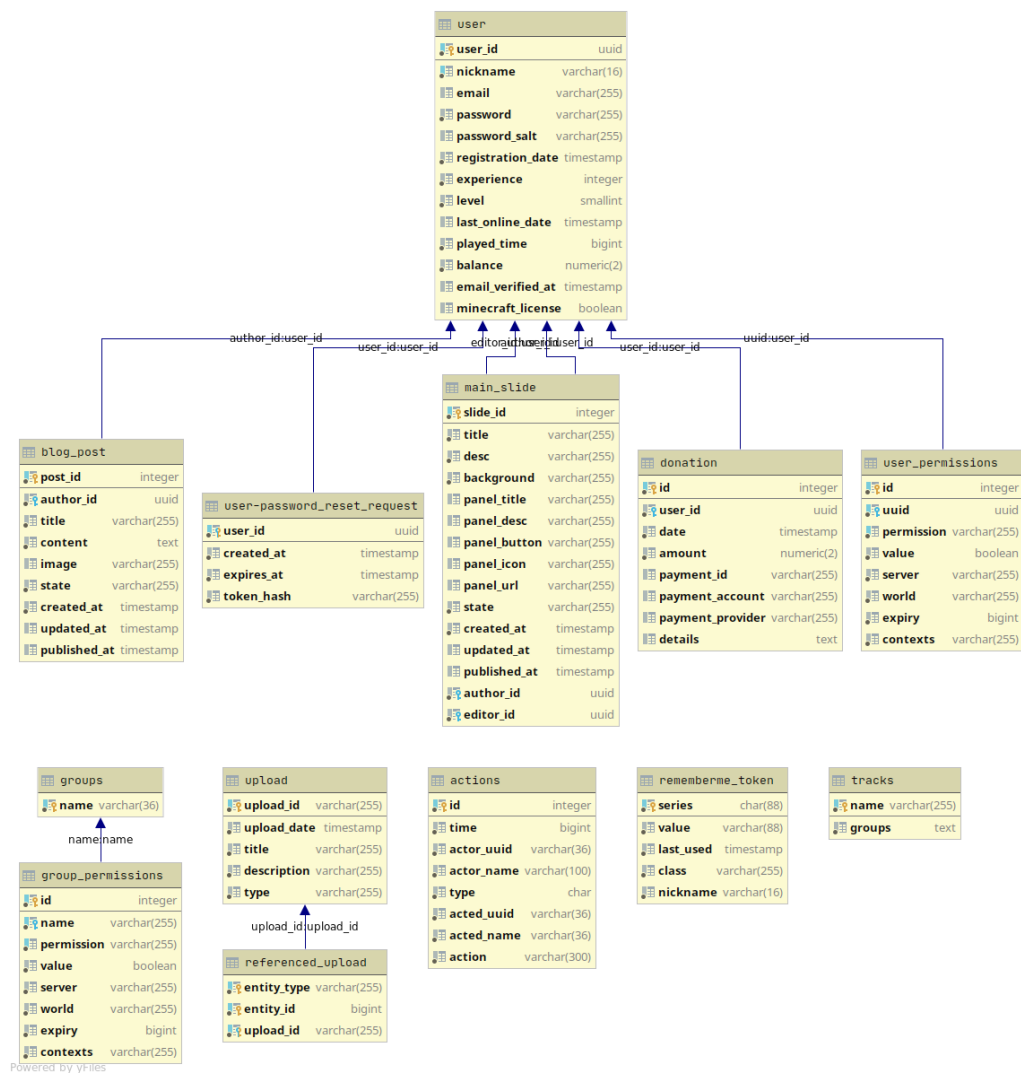


Рисунок 2.1 – Схема базы данных веб-сайта

В процессе разработки активно использовались возможности по преобразовке данных на уровне БД, а также различные виды ограничений целостности, ключей и индексов. В частности, были использованы внешние ключи для связанных таблиц, чтобы обеспечить невозможность удаления независимой сущности при наличии ссылок на нее в зависимых или для обеспечения консистентного удаления всех связанных сущностей, в случаях, когда это обусловлено ограничениями предметной области. Также были использованы индексы на основе вычисляемых свойств для обеспечения логической уникальности регистронезависимых полей (никнейм и e-mail).

Схема разработанной базы данных представлена на рисунке 2.1.

2.2 Интеграция базы данных «LuckPerms»

В рамках данной работы была поставлена задача интегрировать разрабатываемую систему с расширением контроля прав доступа игровой платформы «LuckPerms». Интеграция осуществлялась в рамках подхода использования общей базы данных (shared database). Исходная структура базы данных расширения изображена на рисунке 2.2.

Для объединения исходных сущностей player и user была создана проекция (view) таблицы user со следующим сопоставлением полей:

1. user.user_id – players.uuid;
2. user.nickname – players.nickname;
3. const “default” – players.primary_group.

Запрос для создания такой проекции приведен в листинге 2.1. Так как для изменения данных о правах доступа был разработан модуль, заменяющий исходные процедуры «LuckPerms», выполняемые в контексте Java-расширения для игровой платформы, то для предотвращения ошибочного изменения данных этим расширением были созданы соответствующие правила для проекции “players”.

```
--changeset klyshko.n:202008132325 runOnChange:true
CREATE OR REPLACE VIEW "players"("uuid", "username", "primary_group") AS
  SELECT "user_id", "nickname", 'default'::varchar(255) FROM "user";

--changeset klyshko.n:202008132352 runOnChange:true
CREATE OR REPLACE RULE "players_insert_fake" AS ON INSERT TO "players" DO
  INSTEAD NOTHING;

--changeset klyshko.n:202008132353 runOnChange:true
CREATE OR REPLACE RULE "players_update_fake" AS ON UPDATE TO "players" DO
  INSTEAD NOTHING;
```

Листинг 2.1 – Миграции для создания проекции “players”.

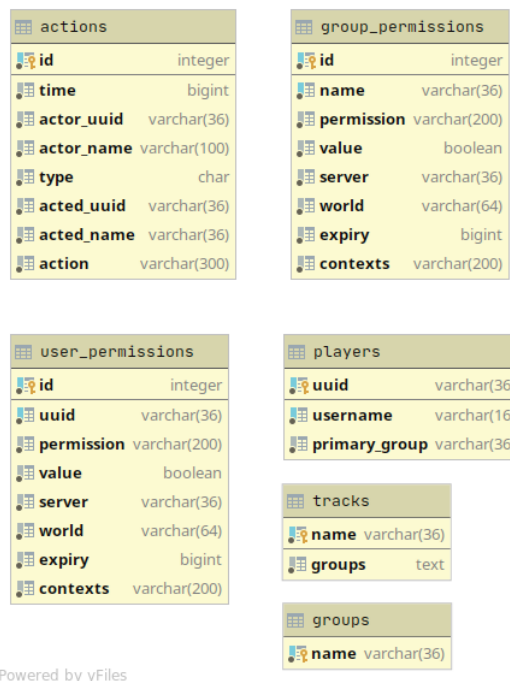


Рисунок 2.2 – Схема базы данных «LuckPerms»

При необходимости извлечения полного списка прав доступа конкретного пользователя будет возникать проблема N+1 запроса, потому что для получения списка прав каждой роли пользователя (заданной через специальный идентификатор “group.<имя роли>”) будет необходимо выполнить отдельный запрос. Запрос, решающий данную проблему, представлен в листинге 2.2.

```
SELECT permission FROM user_permissions WHERE server = \'global\' AND uuid =  
:userId  
      UNION  
      SELECT permission FROM group_permissions WHERE server = \'global\'  
AND name IN  
      (SELECT regexp_replace(permission, \'group\\.\', \'\\') FROM  
user_permissions WHERE uuid = :userId)
```

Листинг 2.2. – Запрос на выборку всех прав пользователя.

3 РАЗРАБОТКА ВЕБ-САЙТА

3.1 Функция регистрации пользователей

Регистрация нового пользователя в системе заключается в том, чтобы создать новый объект пользователя и добавить его в базу данных, при этом соблюдая заданные ограничения уникальности полей. Как уже было отмечено, на уровне базы данных использовались продвинутое ограничения целостности, поэтому при попытке добавления новой записи, нарушающей ограничения уникальности никнейма или e-mail, будет вызвано исключение `UniqueConstraintViolationException`. При этом важно показать пользователю какое именно поле необходимо исправить. Так как библиотеки PDO и Doctrine не предоставляют интерфейса для определения поля, по которому было нарушено ограничение, то был разработан класс, решающий эту задачу. Реализация класса для базы данных PostgreSQL приведена в листинге 3.1.

```
class PostgresConstraintParser implements ConstraintParser
{
    public function parseConstraint(Exception $e)
    {
        $array = explode('DETAIL: ', $e->getMessage());
        if (count($array) < 2) {
            throw $e;
        }
        $details = $array[1];
        if (preg_match('/\(((^|)+)\)/', $details, $m)) {
            $parts = explode(':', $m[1]);
            return $parts[0];
        } else {
            throw $e;
        }
    }
}
```

Листинг 3.1 – Реализация класса для определения поля, по которому было нарушено ограничение целостности в БД.

3.2 Функция аутентификации

Аутентификация – это процедура проверки подлинности. Аутентификация пользователя заключается в том, чтобы проверить на соответствие введенного им имени и пароля тем данным, которые указаны в БД. Фреймворк Symfony позволяет реализовать этот функционал путем расширения абстрактного класса `AbstractFormLoginAuthenticator`.

В техническом задании также указано требование к показу уведомлений о необходимости добавления или подтверждения e-mail пользователю после аутентификации. Так как класс `FormLoginAuthenticator` задействуется только при аутентификации через форму, то есть в случае, когда пользователь не был аутентифицирован ранее с параметром “Запомнить меня”, то необходимо было реализовать обработку события, которое вызывается как при создании новой сессии, так и при ее восстановлении (при перезаходе в браузер). В листинге 3.2 приведена реализация слушателя события аутентификации.

```
class UserVerificationWatcher implements EventSubscriberInterface
{
    private $logger;
    private $trans;
    private $urlGenerator;

    public function __construct(LoggerInterface $logger, TranslatorInterface
    $trans, UrlGeneratorInterface $urlGenerator)
    {
        $this->logger = $logger;
        $this->trans = $trans;
        $this->urlGenerator = $urlGenerator;
    }

    public function onLoginSuccess(InteractiveLoginEvent $event) : void
    {
        $user = $event->getAuthenticationToken()->getUser();
        if ($user instanceof User) {
            if ($user->getEmail() == null) {
                $event->getRequest()->getSession()->getFlashBag()->add('er-
                ror',
                    $this->trans->trans('user.alert.bind_email',
                        ['%url%' => $this->urlGenerator->generate('secu-
                rity_email_bind'))));
            } else
                if ($user->getEmailVerifiedAt() == null) {
```



```

        $event->getRequest()->getSession()->getFlashBag()->add('warn-
ing',
        $this->trans->trans('user.alert.verify_email',
        ['%url%' => $this->urlGenerator->generate('secu-
rity_email_verify'))));
    }
}

public static function getSubscribedEvents()
{
    return [
        InteractiveLoginEvent::class => 'onLoginSuccess'
    ];
}
}

```

Листинг 3.2. – Слушатель события InteractiveLoginEvent.

3.3 Функция восстановления пароля

Функция восстановления доступа к аккаунту является важной частью системы, обеспечивающей возможность вернуть контроль над аккаунтом пользователю в случае его взлома третьими лицами. Причем, если пользователь подтвердил e-mail адрес, то восстановление может быть выполнено в автоматическом режиме.

В целях повышения безопасности процесса автоматического восстановления, в базе данных хранится криптоустойчивый хэш жетона вместе со сроком его действия, который ограничен одним часом с момента его создания. Таким образом, даже в случае компроментации базы данных, злоумышленник не сможет получить доступ к аккаунту пользователя. Короткий срок действия исключает возможность подбора жетона путем полного перебора.

Для предотвращения создания множества запросов на восстановление пароля используется запрос, приведенный в листинге 3.3. В рамках данного запроса проверяется наличие активного запроса, созданного в течение последних трёх минут.

```

INSERT INTO "user-password_reset_request" AS r (user_id, expires_at, token_hash) VALUES (:id, :expires_at, :token_hash)
ON CONFLICT (user_id) DO UPDATE
SET "created_at" = CURRENT_TIMESTAMP, "token_hash" = excluded."token_hash", "expires_at" = excluded."expires_at"
WHERE EXTRACT(epoch FROM (CURRENT_TIMESTAMP - r."created_at")) > :throttle_time

```

Листинг 3.3 – Запрос для создания или обновления запроса восстановления пароля

3.4 Функция подтверждения e-mail

Задача подтверждения e-mail схожа с задачей восстановления пароля. И она состоит в том, чтобы удостовериться, что пользователь действительно владеет (имеет доступ) аккаунтом электронной почты, адрес которого указал при регистрации.

В соответствии с техническим заданием, подтверждением e-mail считается переход пользователя по ссылке, которую он получит в электронном письме. Данная ссылка должна быть подписана электронной подписью, что позволяет закодировать в ней публичные данные, подлинность которых необходимо обеспечить. В частности, идентификатор пользователя, подтверждаемый e-mail адрес, срок действия ссылки. Обработчик http-запросов на создание и проверку таких ссылок приведен в листинге 3.4.

```

public function verify(Request $request, MailerInterface $mailer, UrlGeneratorInterface $urlGenerator,
                        UrlSigner $urlSigner, TranslatorInterface $trans) :
Response
{
    if ($request->get('email') != null && $request->get('id') != null) {
        $user = $this->getDoctrine()->getManager()->getRepository(User::class)->find($request->get('id'));
        if ($user->getEmail() != $request->get('email') || $user->getEmailVerifiedAt() != null) {
            return $this->redirectToRoute('main');
        }
        if ($urlSigner->validate($request->getUri())) {
            $user->setEmailVerifiedAt(new DateTime());
            $this->getDoctrine()->getManager()->flush();
        }
    }
}

```

```

        $request->getSession()->getFlashBag()->add('success', $trans-
>trans('user.alert.email_verified'));
    } else {
        $request->getSession()->getFlashBag()->add('error',
            $trans->trans('user.alert.email_verification_error',
                ['%url%' => $urlGenerator->generate('secu-
rity_email_verify')]));
    }
    return $this->redirectToRoute('main');
}

$this->denyAccessUnlessGranted('IS_AUTHENTICATED_REMEMBERED');

$user = $this->getUser();
if (!$user instanceof User) {
    return $this->redirectToRoute('main');
}
if ($user->getEmail() == null) {
    return $this->redirectToRoute('security_email_bind');
}
if ($user->getEmailVerifiedAt() != null) {
    return $this->redirectToRoute('main');
}

//параметры должны быть расположены в лексикографическом порядке
$url = $urlGenerator->generate('security_email_verify', [
    'email' => $user->getEmail(),
    'id' => $user->getId()
], UrlGeneratorInterface::ABSOLUTE_URL);
$url = $urlSigner->sign($url, 1);

$email = (new TemplatedEmail())
    ->from($this->emailSenderAddress)
    ->to($user->getEmail())
    ->subject($trans->trans('email.verify.subject'))
    ->htmlTemplate('emails/email_verification.html.twig')
    ->context([
        'username' => $user->getUsername(),
        'url' => $url
    ]);
$mailer->send($email);
return $this->render('security/email_verification_sent.html.twig');
}

```

Листинг 3.4 – Обработчик запросов подтверждения e-mail

3.5 Реализация компонентного подхода Vue.js на основе переводимых шаблонов Twig

В случае использования библиотеки Vue.js для реализации отдельных компонентов web-страниц при необходимости обеспечения перевода пользовательского интерфейса на несколько языков, возникает задача внедрения переводов строк в компоненты Vue. При разработке SPA целесообразно загрузить полные карты переводов для всего приложения и использовать их в дальнейшем без перезагрузки страницы браузера, но в случае многостраничного приложения (multi-page application), делать отдельный запрос на каждый Vue-компонент при каждой перезагрузке страницы не оптимально.

Для решение указанной проблемы в данной работе применялись X-Templates, которые включаются в основной поток html документа (шаблона) тем самым получают доступ к директивам Twig, в том числе директиве “trans”. При этом описание шаблона компонента и его объявление (в файле .js) располагается в директории templates проекта. На рисунке 3.1 изображена иерархия файлов Vue-компонентов. Такая файловая структура решает проблему физического разделения шаблона компонента и его объявления, упоминаемую в документации Vue.

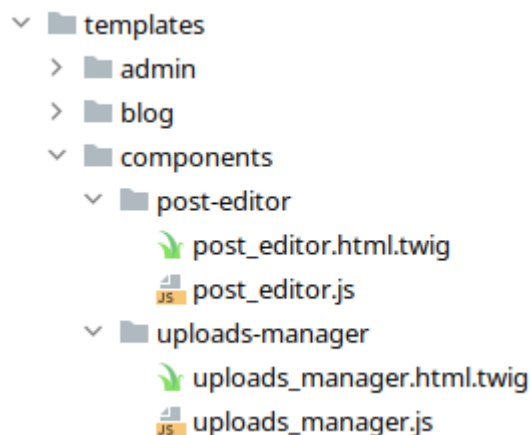


Рисунок 3.1 – Структура директории компонентов Vue

Пример использования директив шаблонизатора Twig в шаблоне компонента приведен в листинге 3.5. Пример объявления компонента с X-шаблоном приведен в листинге 3.6.

```
<script type="text/x-template" id="post-editor-template">
<div class="container">
  {{ form_start(createForm) }}
  {{ form_errors(createForm) }}
  {{ form_row(createForm.title, { 'attr': {'autocomplete': 'off'} }) }}
  {{ form_row(createForm.content) }}

  <ul class="row px-3">
    <li class="col-sm-4 col-md-2 col-lg-2 px-2 my-1" v-for="image in
images">
      <div>
        <i class="fa fa-share-square-o fa-2x" aria-hidden="true"
v-on:click="insertImage(image)"></i>
        <input type="radio" name="create_post_form[image]"
class="post-image"
v-bind:id="`radio${image.url}`"
v-bind:value="image.url">
        <label v-bind:for="`radio${image.url}`" class="fa ml-20">
</label>
        
      </div>
    </li>
    <li class="col-sm-4 col-md-2 col-lg-2 px-2 my-1">
      <button type="button" data-toggle="modal" data-target="#up-
loads-modal" class="img-thumbnail w-100 h-100 text-center">
        <i class="fa fa-plus-circle fa-5x align-middle"></i>
      </button>
    </li>
  </ul>

  <button class="btn btn__secondary btn__hover3 btn__block mt-3"
type="submit">{% trans %}blog.post.create.submit{% endtrans %}</button>

  {{ form_row(createForm._token) }}
  {{ form_end(createForm, {'render_rest': false}) }}
</div>
</script>
```

Листинг 3.5 – Шаблон компонента post-editor

```
Vue.component('post-editor', {
  template: '#post-editor-template',
  delimiters: ['${', '}'],
  // ...
})
```

Листинг 3.6 – Объявление компонента post-editor

3.6 Реализация WYSISWYG-редактора новостной записи

В качестве WYSISWYG-редактора был использован компонент js-библиотеки «EasyMDE». Это простой встраиваемый редактор, использующий в качестве текстового синтаксиса легковесный язык разметки Markdown.

Реализация библиотеки не включает функции загрузки изображений и вставки ссылок на них в редакторе. Недостающий функционал был реализован с помощью библиотеки Vue на клиентской стороне и контроллера загружаемых файлов на серверной стороне. В соответствии с техническим заданием, также реализован запрос с сервера предзагруженных изображений при открытии редактора.

3.7 Интеграция веб-ориентированного редактора «LuckPermsWeb»

Система авторизации данного проекта строится на основе системы прав доступа «LuckPerms», поэтому целесообразно интегрировать совместимый с ней веб-ориентированный редактор. Данный редактор является single-page Vue приложением, рассчитанным на автономное (standalone) развертывание. Как было упомянуто в 2.2, оригинальное расширение LuckPerms обновляет данные в БД с помощью запросов из Java-приложения игровой платформы, используя при этом JSON-копию данных, сформированных веб-редактором.

В рамках интеграции целесообразно обеспечить непосредственное взаимодействие редактора с серверной частью веб-сайта для выполнения обновления данных в БД.

Архитектура решения, позволяющего запрашивать и обновлять данные прав доступа посредством web-api запросов в формате JSON представлена на рисунке 3.2. На рисунке видно, что приложение LuckPermsWeb разворачива-

ется в поддиректории основного веб-сайта. Таким образом, первым и единственным запросом на клиент загружается Vue-приложение редактора. Последующие запросы выполняет уже Vue-приложение, асинхронно.

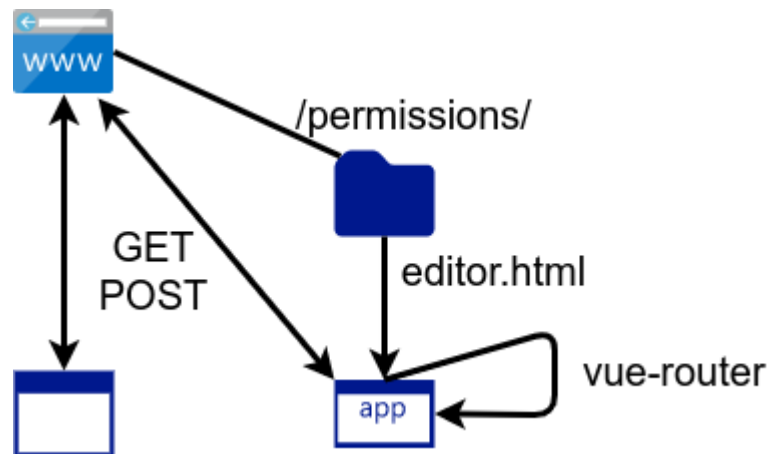


Рисунок 3.2 – Архитектура интеграции редактора LuckPermsWeb

3.8 Интеграция системы прав доступа в систему авторизации Symfony

Авторизация — это предоставление определённому лицу прав на выполнение определённых действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.

Для интеграции с системой авторизации Symfony необходимо обеспечить проверку наличия прав доступа при возникновении запросов авторизации в системе Symfony. Такая логика реализуется путем расширения класса Voter. Причем, вводится следующий формат прав доступа, которые могут быть проверены стандартным образом: “permission.<идентификатор права доступа>”. Реализация класса PermissionVoter приведена в листинге 3.7.

```

class PermissionVoter extends Voter
{
    const PERMISSION_PREFIX = 'permission.';

    /**
     * @inheritDoc
     */
    protected function supports(string $attribute, $subject)
    {
        return substr($attribute, 0, strlen(self::PERMISSION_PREFIX)) ===
self::PERMISSION_PREFIX;
    }

    /**
     * @inheritDoc
     */
    protected function voteOnAttribute(string $attribute, $subject, Token-
Interface $token)
    {
        $user = $token->getUser();
        if (!($user instanceof User)) {
            return false;
        }
        return $user->hasPermission(substr($attribute, strlen(self::PERMIS-
SION_PREFIX)));
    }
}

```

Листинг 3.7 – Реализация класса `PermissionVoter`, проверяющего наличие запрошенного права доступа

В целях оптимизации реализована ленивая загрузка списка прав пользователя из базы данных. То есть, список загружается только если в рамках текущего HTTP-запроса необходима авторизация пользователя. Ленивая загрузка достигается путем хранения списка прав во вложенном объекте класса `CachedPlainPermissionsHolder`. Для инстанцирования этого класса используются обработчики жизненного цикла сущности библиотеки `Doctrine`. Реализация обработчиков представлена в листинге 3.8.

```

class PermissionAttachment extends AutoTaggedEntityListener
{
    /**
     * @var PermissionRepository
     */
    private $repository;
}

```



```

/**
 * PermissionAttachment constructor.
 * @param PermissionRepository $repository
 */
public function __construct(PermissionRepository $repository)
{
    $this->repository = $repository;
}

public function postLoad(User $user)
{
    $this->attachPermissions($user);
}

public function postPersist(User $user)
{
    $this->attachPermissions($user);
}

private function attachPermissions(User $user)
{
    $user->setPermissions(new CachedPlainPermissionsHolder($this->repository, $user->getId()));
}
}

```

Листинг 3.8 – Реализация обработчиков жизненного цикла сущности User

4 ТЕСТИРОВАНИЕ ВЕБ-САЙТА

4.1 Главная страница

Изображение главной страницы (рисунок 4.2) соответствует требованиям технического задания.

На этой странице располагаются следующие элементы:

- главное меню в верхней части экрана, которое фиксируется при прокрутке;
- слайдер с фоновыми изображениями и описанием преимуществ проекта;
- список новостей и элементы навигации по страницам.

При переходе по страницам новостей область видимости перемещается на первую видимую новость (рисунок 4.1).

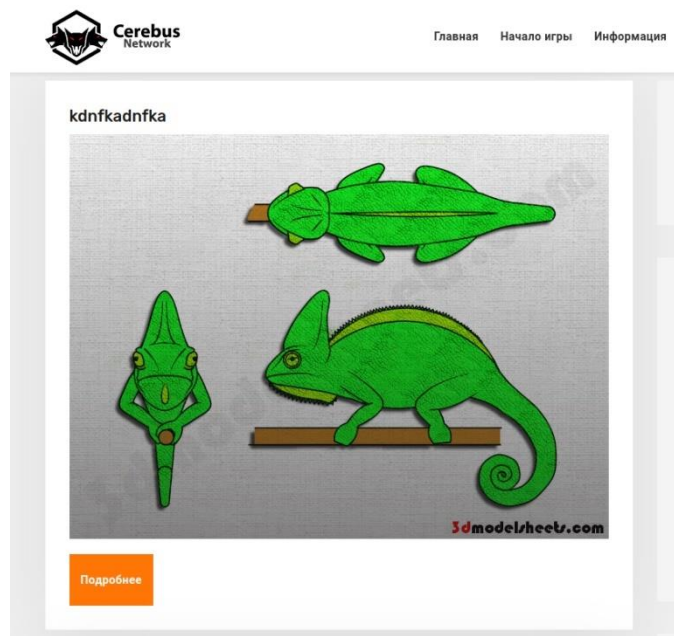


Рисунок 4.1 – Положение области видимости при переходе на другую страницу новостей

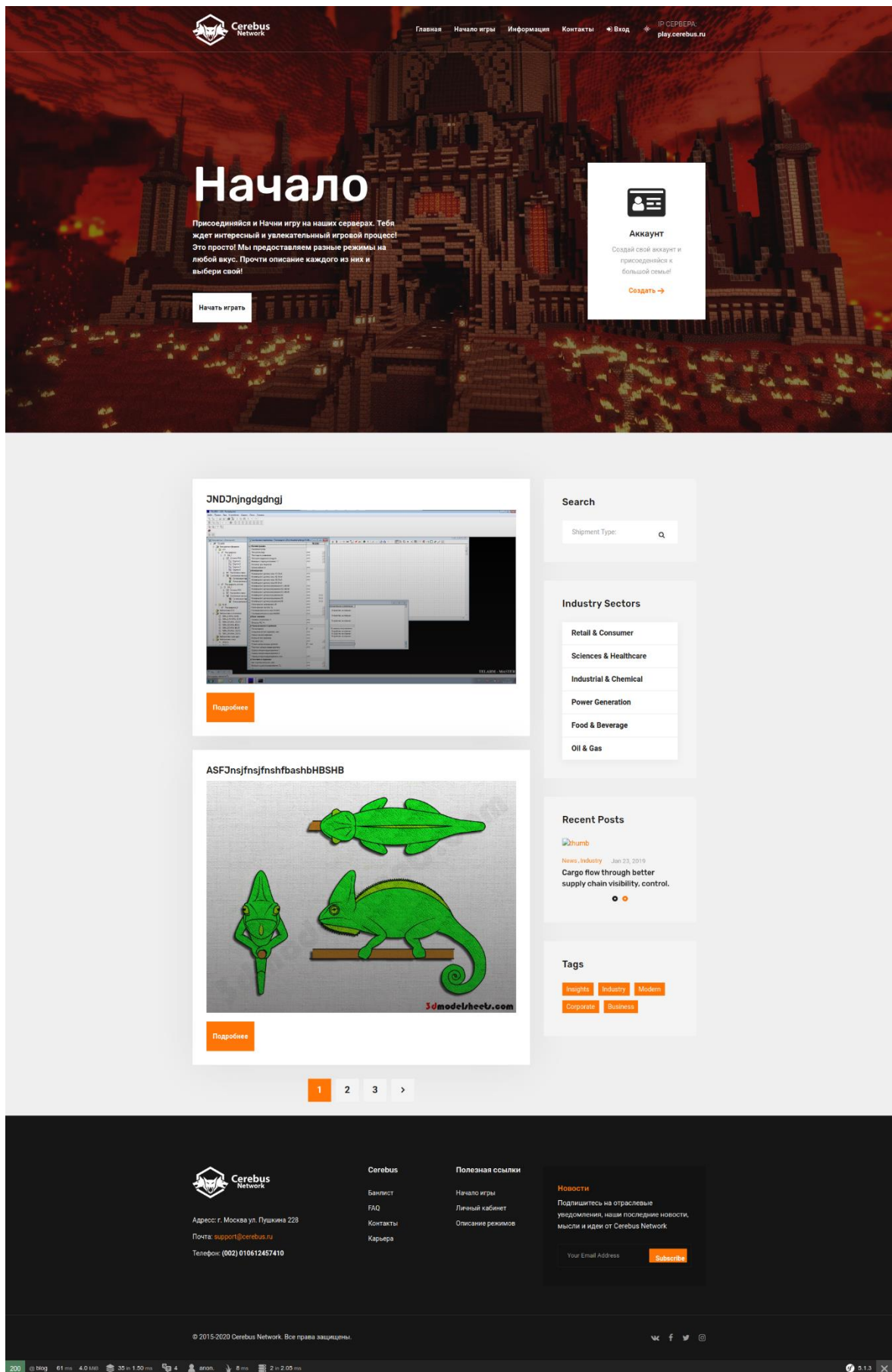
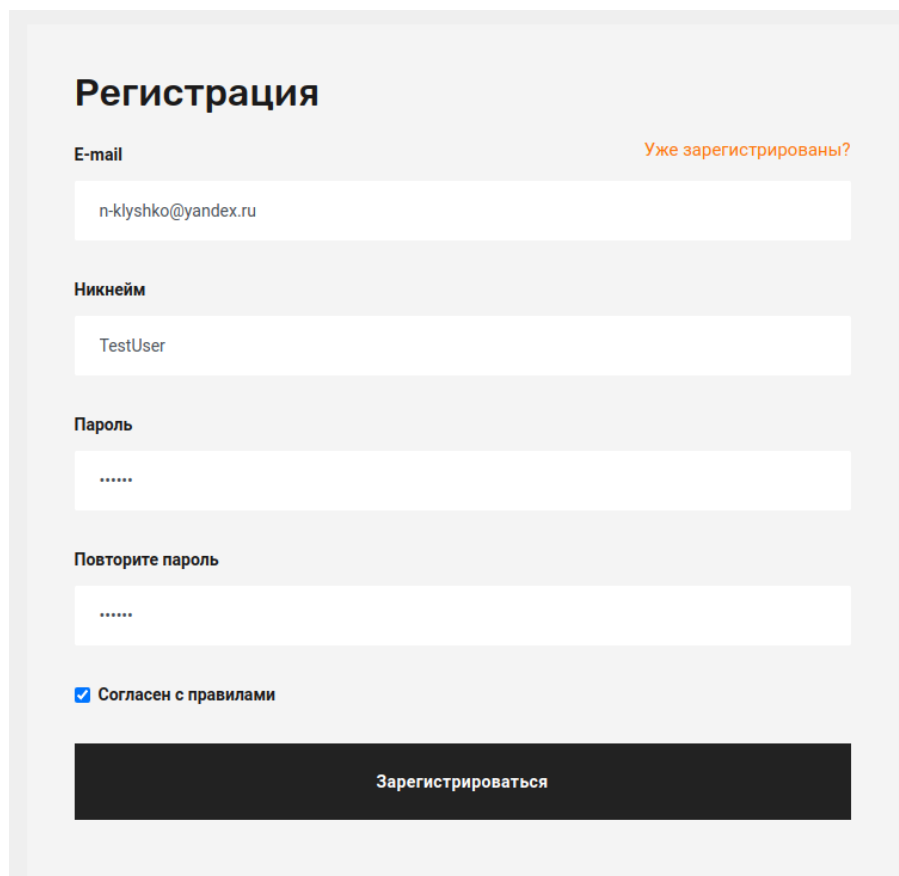


Рисунок 4.2 – Скриншот главной страницы

4.2 Регистрация пользователя

Введем корректные данные для регистрации нового пользователя (рисунок 4.3). После нажатия на кнопку “Зарегистрироваться” браузер перенаправляется на главную страницу и в главном меню отображается имя пользователя (рисунок 4.4). Это свидетельствует об успешной регистрации.



The screenshot shows a registration form with the following fields and elements:

- Регистрация** (Registration) - Title of the form.
- E-mail** - Label for the email field, with a link [Уже зарегистрированы?](#) (Already registered?).
- n-klyshko@yandex.ru** - Text entered in the email field.
- Никнейм** (Nickname) - Label for the nickname field.
- TestUser** - Text entered in the nickname field.
- Пароль** (Password) - Label for the password field.
-** - Masked text in the password field.
- Повторите пароль** (Repeat password) - Label for the repeat password field.
-** - Masked text in the repeat password field.
- ☒ **Согласен с правилами** (Agree with terms) - Checked checkbox.
- Зарегистрироваться** (Register) - Button to submit the form.

Рисунок 4.3 – Форма регистрации с корректным заполнением полей

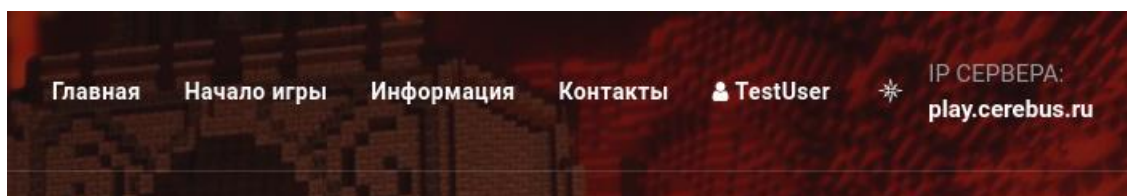


Рисунок 4.4 – Вид главного меню после успешной регистрации

Кроме этого, на главной странице отображается уведомление о том, что пользователь не подтвердил e-mail (рисунок 4.5).

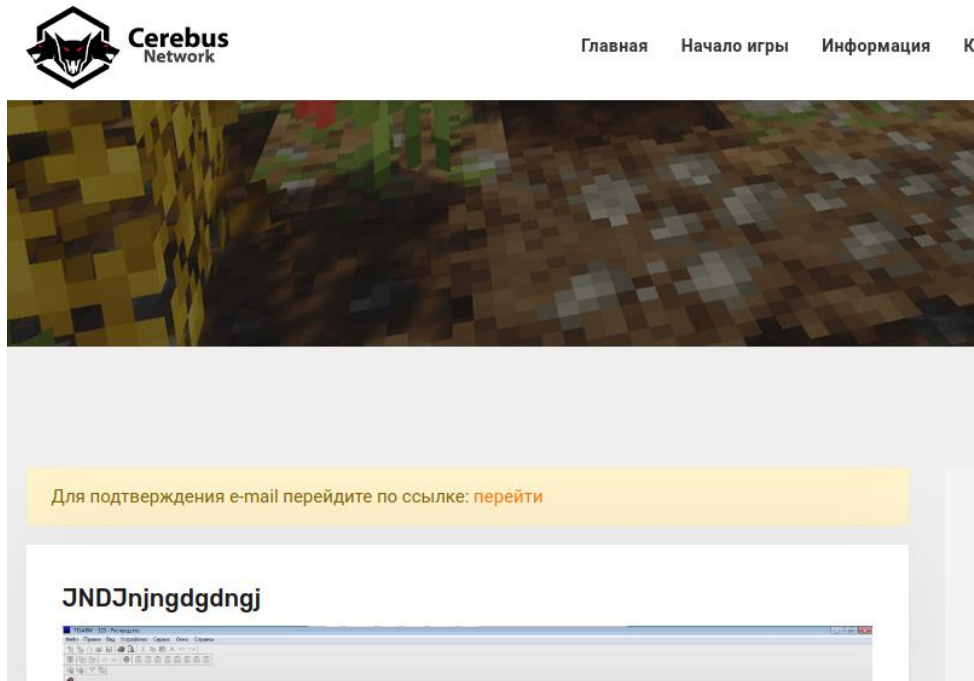


Рисунок 4.5 – Уведомление о необходимости подтверждения e-mail

При нажатии на кнопку “перейти” отображается веб-страница, подтверждающая успешную отправку электронного письма с подтверждением.

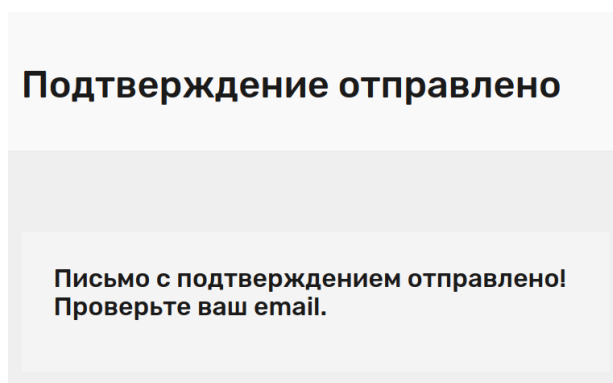


Рисунок 4.6 – Страница об отправке письма с подтверждением

Содержимое письма с подтверждением изображено на рисунке 4.7. При переходе по указанной ссылке происходит подтверждение заданного e-mail и отображается соответствующее уведомление (рисунок 4.8).

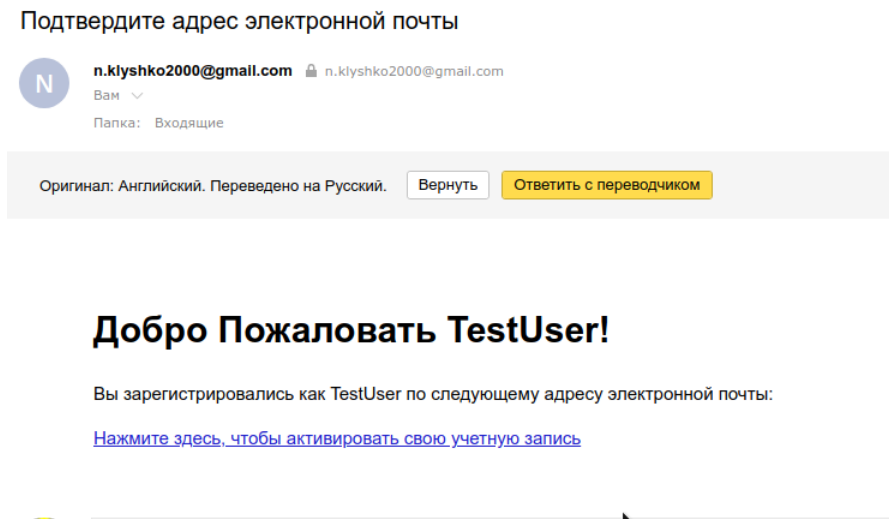


Рисунок 4.7 – Содержимое письма для подтверждения e-mail

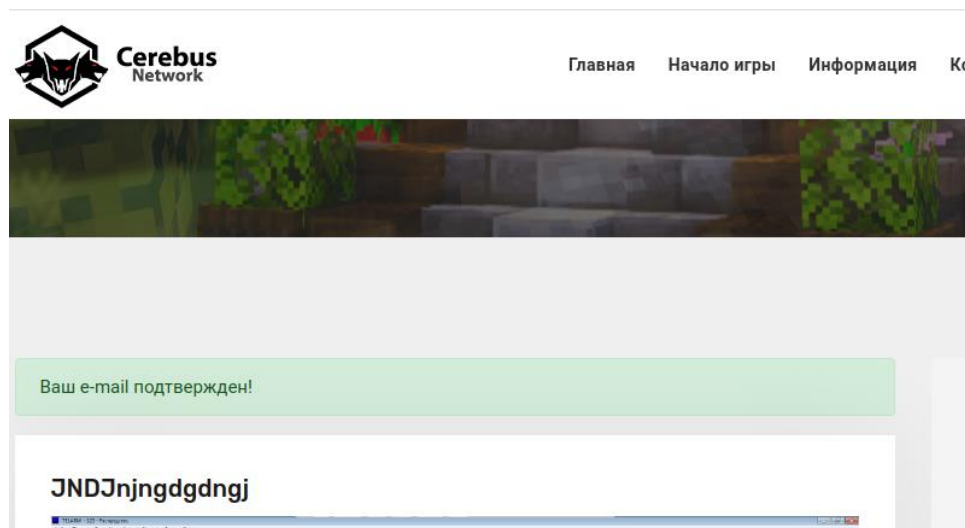
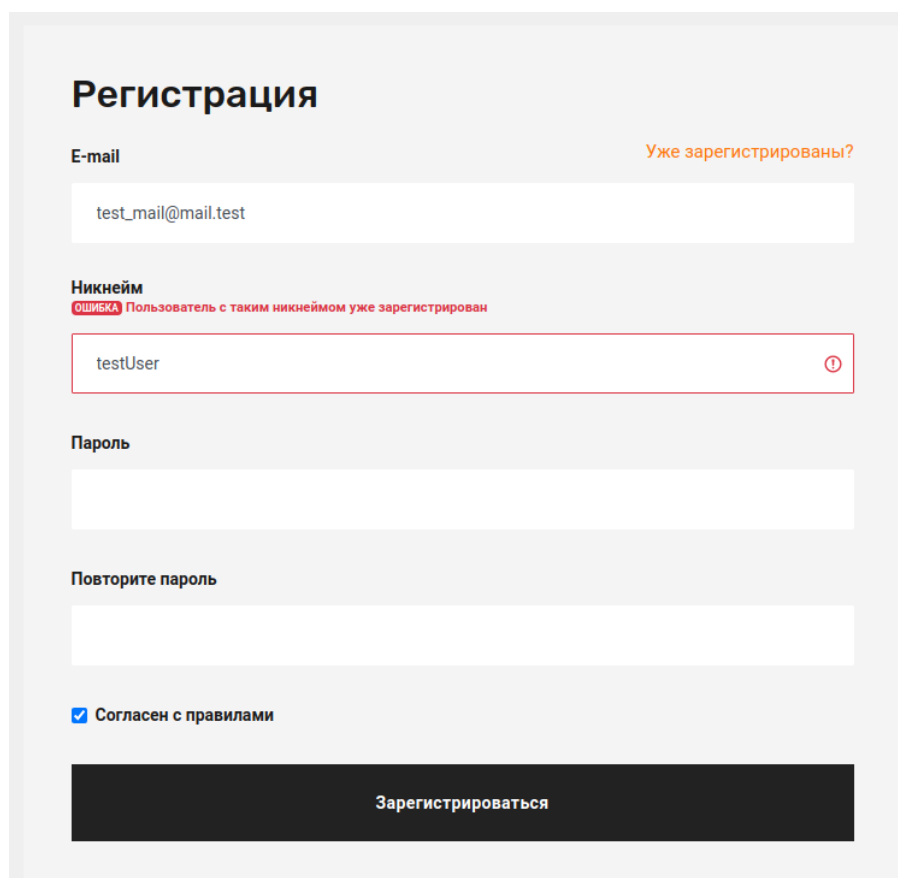


Рисунок 4.8 – Уведомление об успешном подтверждении e-mail

Для проверки валидации формы регистрации введем никнейм, который уже зарегистрирован в системе. При нажатии на кнопку “Зарегистрироваться” в форме регистрации будут отображены ошибки (рисунок 4.9).



Регистрация

E-mail [Уже зарегистрированы?](#)

test_mail@mail.test

Никнейм
ОШИБКА Пользователь с таким никнеймом уже зарегистрирован

testUser

Пароль

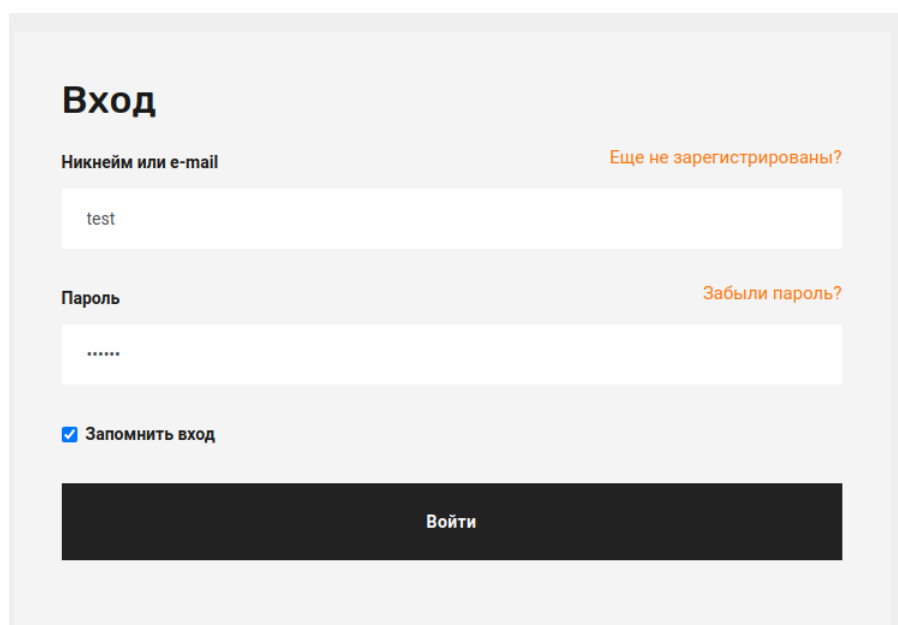
Повторите пароль

☒ **Согласен с правилами**

Зарегистрироваться

Рисунок 4.9 – Отображение ошибки в форме регистрации

4.3 Аутентификация пользователя



Вход

Никнейм или e-mail [Еще не зарегистрированы?](#)

test

Пароль [Забыли пароль?](#)

☒ **Запомнить вход**

Войти

Рисунок 4.10 – Корректно заполненная форма входа

При успешной аутентификации пользователя, у которого не добавлен e-mail, на главной странице отображается соответствующее уведомление (рисунок 4.11).

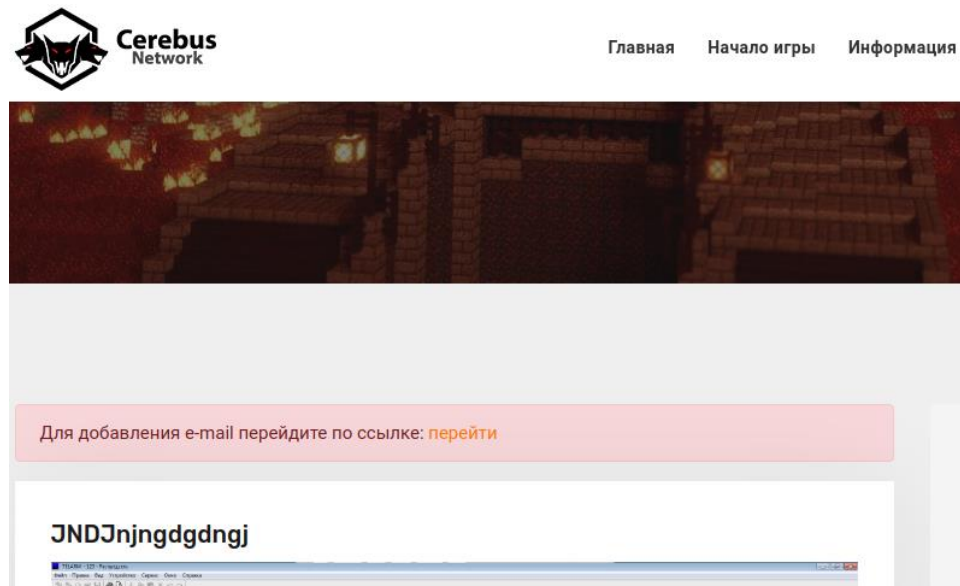


Рисунок 4.11 – Уведомление о необходимости добавить e-mail

При нажатии на кнопку “перейти” происходит переход к форме добавления e-mail (рисунок 4.12). После отправки формы выполняется сценарий подтверждения e-mail.

The image shows a web form titled "Добавить электронную почту" (Add email). Below the title, the label "E-mail" is present. There is a text input field with a light blue background and an orange border, containing the text "test3@mail.ru". Below the input field is a large black button with the white text "Добавить". The entire form is set against a light gray background.

Рисунок 4.12 – Форма добавления e-mail

4.4 Восстановление пароля

Для восстановления пароля необходимо нажать на кнопку “Забыли пароль?” на форме входа. В результате происходит переход на страницу запроса восстановления пароля (рисунок 4.13).

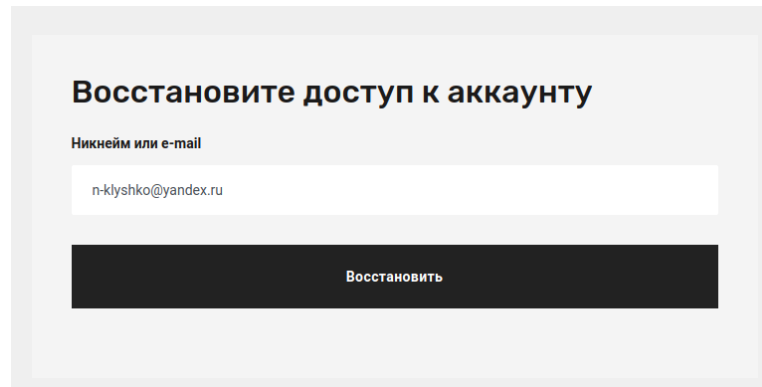
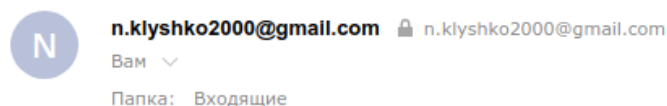


Рисунок 4.13 – Форма запроса восстановления пароля

После заполнения поля никнейм или e-mail и нажатия на кнопку “Восстановить”, происходит отправка электронного письма со ссылкой на страницу изменения пароля, действующую в течение часа. Содержимое электронного письма изображено на рисунке 4.14.

Подтвердите адрес электронной почты



Восстановление пароля Cerebus Network!

Вы запросили восстановление пароля к аккаунту Cerebus Network

[Click here to reset your password](#)

Рисунок 4.14 – Содержимое письма для восстановления пароля

При нажатии на указанную ссылку происходит переход к форме изменения пароля (рисунок 4.15). После нажатия на кнопку “Подтвердить”, в случае совпадения паролей, новый пароль будет установлен для этого аккаунта.

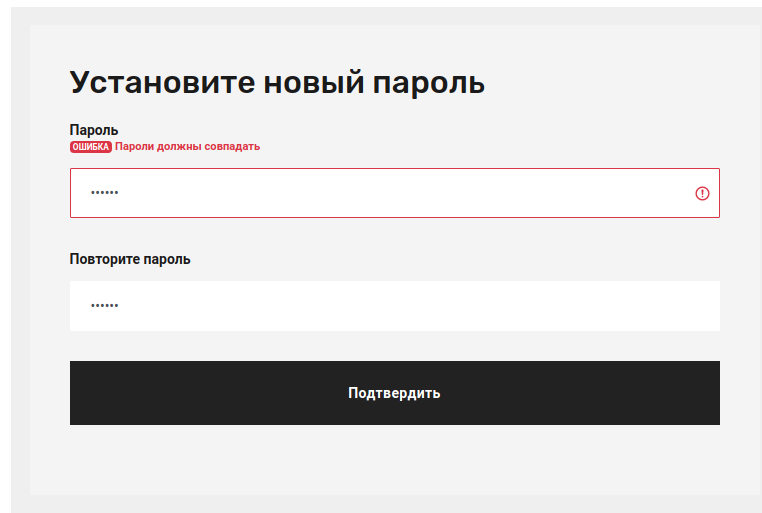


Рисунок 4.15 – Форма подтверждения изменения пароля

4.5 Редактор прав доступа

Интерфейс редактора прав доступа представлен на рисунке 4.16. Для доступа к этой странице используется url `/permissions/index.html`, чтобы веб-сервер nginx не перенаправил запрос на приложение Symfony. Vue-приложение, расположенное по этому адресу, определяет этот переход как начало работы и запрашивает с сервера данные о правах доступа, после чего происходит переход к графическому редактору без перезагрузки страницы.

В левой панели можно выбрать объект, для которого необходимо изменить или просмотреть права доступа. После выбора объекта, эта информация отображается в основной части редактора. В нижней части находится панель для добавления новых прав доступа. Для сохранения изменений необходимо нажать на кнопку “Save”, расположенную в верхнем правом углу.

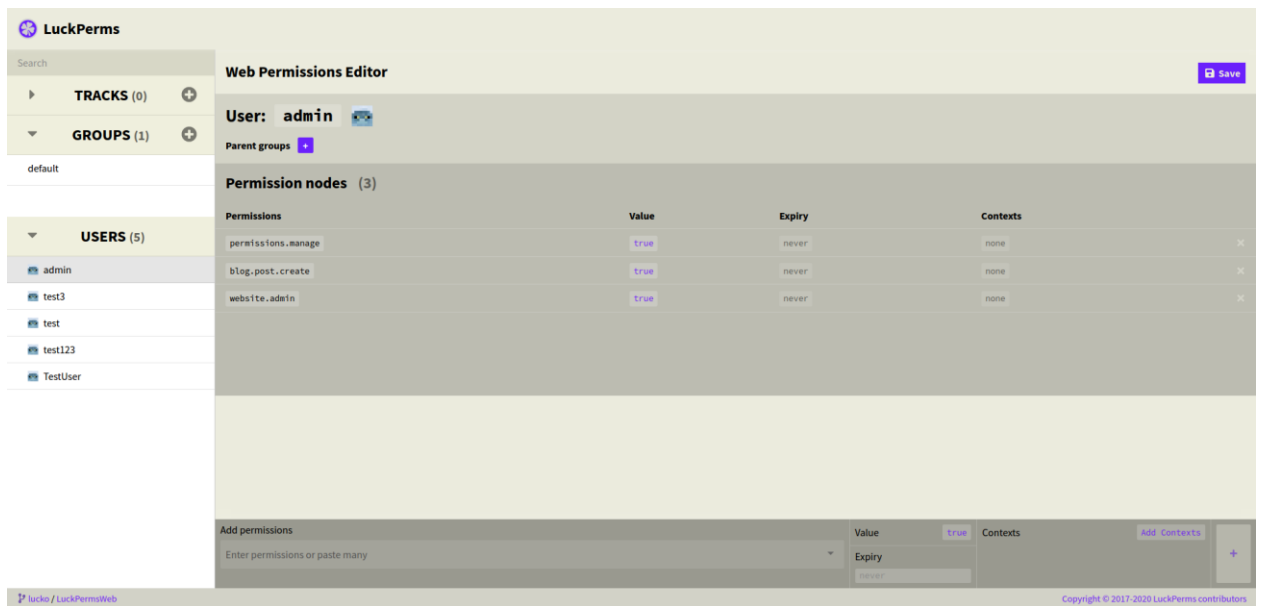


Рисунок 4.16 – Интерфейс редактора прав доступа

4.6 Редактор новостей

Редактор состоит из трёх основных элементов: заголовок, содержимое (content) и добавление изображений. Интерфейс редактора изображен на рисунке 4.17.

Для добавления изображений необходимо нажать на кнопку с изображением символа «+», после чего откроется окно менеджера загрузок (рисунок 4.18). В этом окне можно выбрать предзагруженное изображение, которые необходимо прикрепить к этой новости, или загрузить дополнительные (будут автоматически выбраны для прикрепления).

Прикрепленные изображения отобразятся в нижней панели редактора новости. Для вставки изображения в текст необходимо использовать кнопку в виде прямоугольника со стрелкой, а для выбора изображения в качестве заглавного для этой новости – кнопку с изображением браузерного окна.

После нажатия кнопки “Создать пост”, новость будет добавлена в базу данных. С записью в базе данных также будут ассоциированы использованные в тексте изображения.

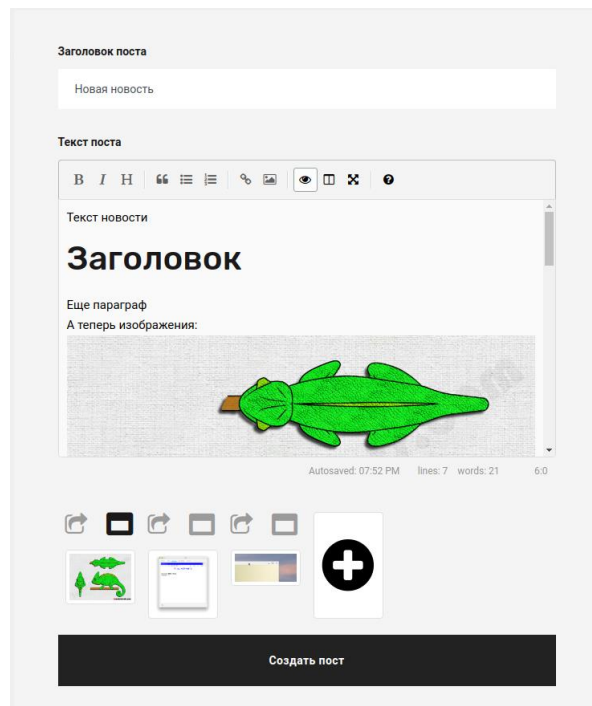


Рисунок 4.17 – Интерфейс редактора новости

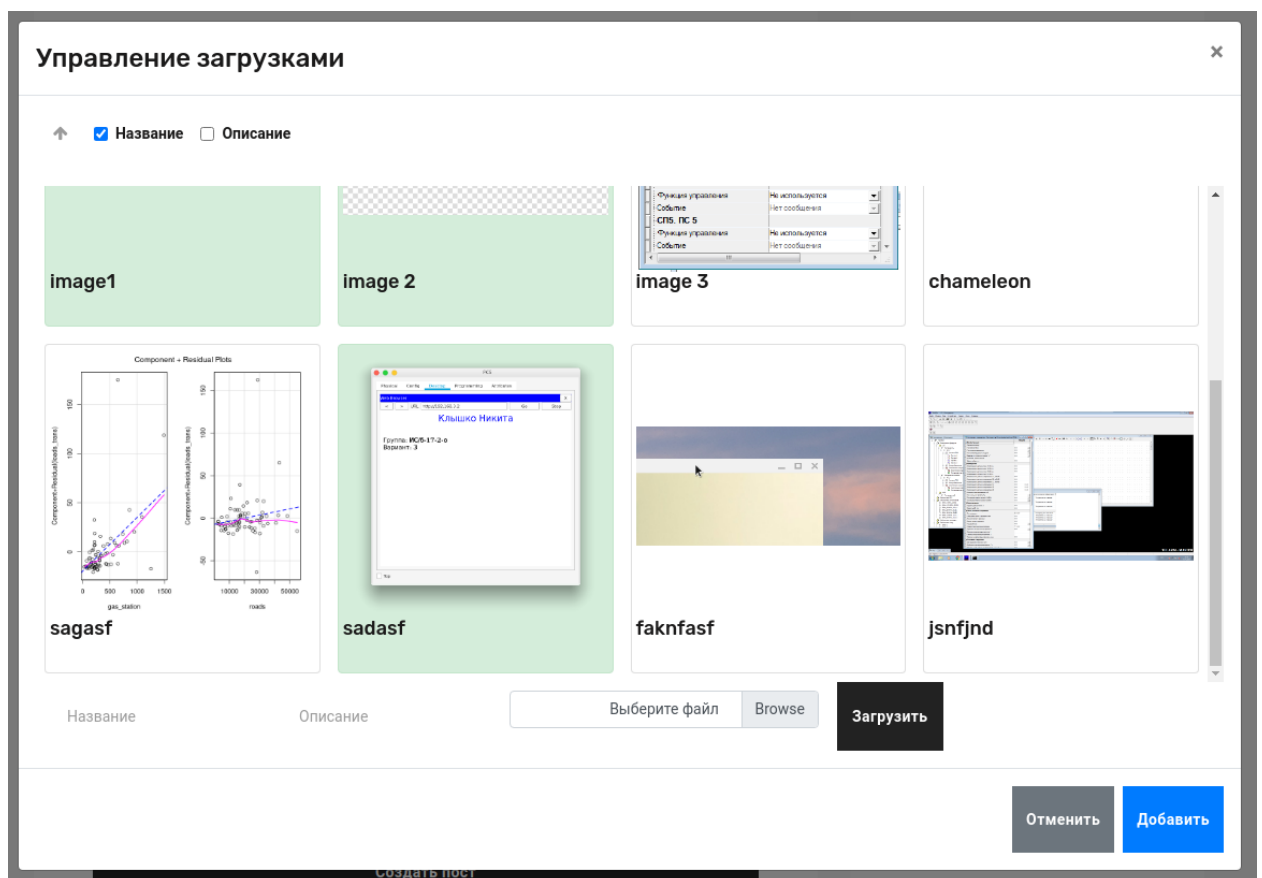


Рисунок 4.18 – Интерфейс менеджера загрузок

ЗАКЛЮЧЕНИЕ

В ходе данного курсового проекта разработана новостной веб-сайт с возможностью регистрации личного аккаунта для игрового проекта «Cerebus Network».

Аналитическая часть работы заключалась в исследовании предметной области и определении сущностей и связей между ними. В этой части была проанализирована предметная область новостного сайта, а также предметная область системы контроля прав доступа «LuckPerms». Предложено проектное решение по интеграции этих систем.

На основе проведенного анализа, была разработана схема базы данных и запросы для выборки, добавления и изменения записей. Для взаимодействия с базой данных были разработаны ORM-модели. Также были разработаны контроллеры и представления всех необходимых веб-страниц.

В ходе разработки функциональности вебсайта были применены технологии программирования как на стороне клиента, так и на стороне сервера для обеспечения удобного пользовательского опыта. Кроме этого, был применен нестандартный подход для реализации переводимости (локализации) Vue-компонентов.

Таким образом, в ходе реализации проекта были выполнены все поставленные задачи, цель курсового проекта была достигнута.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Дейт К. Дж. Введение в системы баз данных [Текст] / К. Дж. Дейт. - 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильямс", 2005. — 1328 с.: ил. — Парал. тит. англ.
2. Мейер Д. Теория реляционных баз данных [Текст] / Д. Мейер. - Мир: Москва, 1987 - 608с.
3. Macrae, C. Vue.js: Up and Running. Building accessible and performant web apps [Текст] / C. Macrae. - O'Reilly Media: Sebastopol, 2018 – 173с.
4. Potencier, P. Symfony 5. Быстрый старт [Текст] / P. Potencier. - Symfony SAS: France, 2020 – 360с.

ПРИЛОЖЕНИЕ А

Листинг запросов инициализации БД

```
--liquibase formatted sql

--changeset klyshko.n:201912151136
CREATE TABLE "user" (
    user_id          uuid NOT NULL,
    nickname         VARCHAR(16) NOT NULL,
    email            VARCHAR(255),
    password          VARCHAR(255) NOT NULL,
    password_salt     VARCHAR(255),
    registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    experience        int4 DEFAULT 0 NOT NULL,
    level             int2 DEFAULT 0 NOT NULL,
    last_online_date  TIMESTAMP,
    played_time       int8 DEFAULT 0 NOT NULL,
    balance           NUMERIC(2) DEFAULT 0 NOT NULL,
    email_verified_at TIMESTAMP,
    minecraft_license BOOLEAN DEFAULT FALSE,
    PRIMARY KEY (user_id),
    UNIQUE (nickname)
);

--changeset klyshko.n:201912151137
CREATE UNIQUE INDEX "user_nickname_unique" ON "user" (LOWER("nickname"));

--changeset klyshko.n:201912151138
CREATE UNIQUE INDEX "user_email_unique" ON "user" (LOWER("email"));

--changeset klyshko.n:201912151207
CREATE TABLE donation (
    id                SERIAL NOT NULL,
    user_id            uuid NOT NULL,
    date               TIMESTAMP NOT NULL,
    amount             NUMERIC(2) NOT NULL,
    payment_id         VARCHAR(255),
    payment_account    VARCHAR(255),
    payment_provider   VARCHAR(255),
    details            text,
    PRIMARY KEY (id),
    CONSTRAINT "user-donation_fk1" FOREIGN KEY (user_id) REFERENCES "user"(user_id)
    ON UPDATE RESTRICT ON DELETE RESTRICT
);

--changeset klyshko.n:201912151717
CREATE TABLE "user-password_reset_request" (
    user_id            uuid NOT NULL,
    created_at         TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    expires_at         TIMESTAMP NOT NULL,
    token_hash         VARCHAR(255) NOT NULL,
    PRIMARY KEY (user_id),
    CONSTRAINT "password_reset_request_user_id_fk" FOREIGN KEY (user_id) REFERENCES "user"(user_id)
    ON UPDATE RESTRICT ON DELETE RESTRICT
);

--liquibase formatted sql

--changeset klyshko.n:202008052202
CREATE TABLE "rememberme_token" (
    "series"          CHAR(88)          UNIQUE PRIMARY KEY NOT NULL,
    "value"           VARCHAR(88)        NOT NULL,
    "last_used"       TIMESTAMP          NOT NULL,
    "class"           VARCHAR(255)       NOT NULL,
    "nickname"        VARCHAR(16)        NOT NULL
);

--changeset klyshko.n:202008131243
CREATE TABLE "user_permissions" (
    "id"              SERIAL PRIMARY KEY NOT NULL,
    "uuid"            uuid               NOT NULL,
    "permission"      VARCHAR(255)       NOT NULL,
    "value"           BOOL               NOT NULL,
    "server"          VARCHAR(255)       NOT NULL,
    "world"           VARCHAR(255)       NOT NULL,
```

```

"expiry"      BIGINT                                NOT NULL,
"contexts"    VARCHAR(255)                          NOT NULL,
CONSTRAINT "user_permission_unique" UNIQUE ("uuid", "permission"),
CONSTRAINT "user_permissions_user_uuid_fk" FOREIGN KEY (uuid) REFERENCES "user"(user_id)
ON UPDATE CASCADE ON DELETE RESTRICT
);

--changeset klyshko.n:202008131244
CREATE INDEX "user_permissions_uuid" ON "user_permissions" ("uuid");

--changeset klyshko.n:202008131245
CREATE TABLE "groups" (
  "name" VARCHAR(36) PRIMARY KEY NOT NULL
);

--changeset klyshko.n:202008131246
CREATE TABLE "group_permissions" (
  "id"          SERIAL PRIMARY KEY          NOT NULL,
  "name"        VARCHAR(255)                NOT NULL,
  "permission"  VARCHAR(255)                NOT NULL,
  "value"       BOOL                       NOT NULL,
  "server"     VARCHAR(255)                 NOT NULL,
  "world"      VARCHAR(255)                 NOT NULL,
  "expiry"     BIGINT                      NOT NULL,
  "contexts"   VARCHAR(255)                 NOT NULL,
  CONSTRAINT "group_permission_unique" UNIQUE ("name", "permission"),
  CONSTRAINT "group_name_fk" FOREIGN KEY ("name") REFERENCES groups("name")
  ON UPDATE CASCADE ON DELETE CASCADE
);

--changeset klyshko.n:202008131247
CREATE INDEX "group_permissions_name" ON "group_permissions" ("name");

--changeset klyshko.n:202008131250
CREATE TABLE "actions" (
  "id"          SERIAL PRIMARY KEY          NOT NULL,
  "time"        BIGINT                     NOT NULL,
  "actor_uuid"  VARCHAR(36)                NOT NULL,
  "actor_name"  VARCHAR(100)               NOT NULL,
  "type"        CHAR(1)                    NOT NULL,
  "acted_uuid"  VARCHAR(36)                NOT NULL,
  "acted_name"  VARCHAR(36)                NOT NULL,
  "action"      VARCHAR(300)               NOT NULL
);

--changeset klyshko.n:202008131251
CREATE TABLE "tracks" (
  "name"        VARCHAR(255) PRIMARY KEY NOT NULL,
  "groups"      TEXT              NOT NULL
);

--changeset klyshko.n:202008132325 runOnChange:true
CREATE OR REPLACE VIEW "players"("uuid", "username", "primary_group") AS
  SELECT "user_id", "nickname", 'default'::varchar(255) FROM "user";

--changeset klyshko.n:202008132352 runOnChange:true
CREATE OR REPLACE RULE "players_insert_fake" AS ON INSERT TO "players" DO INSTEAD NOTHING;

--changeset klyshko.n:202008132353 runOnChange:true
CREATE OR REPLACE RULE "players_update_fake" AS ON UPDATE TO "players" DO INSTEAD NOTHING;

--changeset klyshko.n:202010140919
INSERT INTO "groups"("name") VALUES ('default');

--liquibase formatted sql

--changeset klyshko.n:202009160942
CREATE TABLE "upload" (
  "upload_id"   VARCHAR(255) PRIMARY KEY NOT NULL,
  "upload_date" TIMESTAMP      DEFAULT CURRENT_TIMESTAMP NOT NULL,
  "title"       VARCHAR(255) NOT NULL,
  "description" VARCHAR(255) NOT NULL,
  "type"        VARCHAR(255) NOT NULL
);

--changeset klyshko.n:202009301124

```



```

CREATE TABLE "blog_post" (
  "post_id"      SERIAL          PRIMARY KEY NOT NULL,
  "author_id"    uuid            NOT NULL,
  "title"        VARCHAR(255)    NOT NULL,
  "content"      TEXT            NOT NULL,
  "image"        VARCHAR(255)    NULL,
  "state"        VARCHAR(255)    NOT NULL,
  "created_at"   TIMESTAMP       DEFAULT CURRENT_TIMESTAMP NOT NULL,
  "updated_at"   TIMESTAMP       NULL,
  "published_at" TIMESTAMP       NULL,
  CONSTRAINT "post-author_fk1" FOREIGN KEY (author_id) REFERENCES "user"(user_id)
                        ON UPDATE RESTRICT ON DELETE RESTRICT
);

--changeset klyshko.n:202009301142
CREATE TABLE "referenced_upload" (
  "entity_type"  VARCHAR(255),
  "entity_id"    BIGINT,
  "upload_id"    VARCHAR(255),
  PRIMARY KEY ("entity_type", "entity_id", "upload_id"),
  CONSTRAINT "reference-upload_fk" FOREIGN KEY (upload_id) REFERENCES upload(upload_id)
                        ON UPDATE RESTRICT ON DELETE RESTRICT
);

```

ПРИЛОЖЕНИЕ Б

Исходный код приложения

RegistrationController.php:

```
class RegistrationController extends AbstractController
{
    /**
     * @Route("/register", name="app_register")
     *
     * @param Request $request
     * @param UserPasswordEncoderInterface $passwordEncoder
     * @param GuardAuthenticatorHandler $guardHandler
     * @param FormAuthenticator $authenticator
     * @param TranslatorInterface $translator
     * @param ConstraintParser $constraintParser
     * @return Response
     * @throws Exception
     */
    public function register(Request $request, UserPasswordEncoderInterface $passwordEncoder,
        GuardAuthenticatorHandler $guardHandler, FormAuthenticator $authenticator,
        TranslatorInterface $translator, ConstraintParser $constraintParser): Response
    {
        $user = new User();
        $form = $this->createForm(RegistrationForm::class, $user);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            // encode the plain password
            $user->setPassword(
                $passwordEncoder->encodePassword(
                    $user,
                    $form->get('plainPassword')->getData()
                )
            );

            try {
                $user->setId(Uuid::v4()->toRfc4122());
                $this->getDoctrine()->getManager()->persist($user);
                $this->getDoctrine()->getManager()->flush();

                return $guardHandler->authenticateUserAndHandleSuccess(
                    $user,
                    $request,
                    $authenticator,
                    'main' // firewall name in security.yaml
                );
            } catch (UniqueConstraintViolationException $e) {
                switch ($constraintParser->parseConstraint($e)) {
                    case 'nickname':
                        $form->get('username')->addError(
                            new FormError($translator->trans('error.register.same_nickname_already_exists')));
                        break;
                    case 'email':
                        $form->get('email')->addError(
                            new FormError($translator->trans('error.register.same_email_already_exists')));
                        break;
                    default:
                        $form->addError(new FormError($translator->trans('error.register.unknown')));
                        break;
                }
            } catch (Exception $e) {
                $form->addError(new FormError($translator->trans('error.register.unknown')));
            }
        }

        return $this->render('security/register.html.twig', [
            'registrationForm' => $form->createView(),
        ]);
    }
}
```

}

RecoveryController.php:

```

class RecoveryController extends AbstractController
{
    private $emailSenderAddress;

    public function __construct($emailSenderAddress)
    {
        $this->emailSenderAddress = $emailSenderAddress;
    }

    /**
     * @Route(path = "/recovery/request", name = "security_recovery_request")
     *
     * @param Request $request
     * @param TranslatorInterface $translator
     * @param UrlGeneratorInterface $urlGenerator
     * @param PasswordResetService $service
     * @param MailerInterface $mailer
     * @return Response
     * @throws TransportExceptionInterface
     */
    public function request(Request $request, TranslatorInterface $translator,
        UrlGeneratorInterface $urlGenerator, PasswordResetService $service, MailerIn-
terface $mailer)
    {
        $form = $this->createForm(RecoveryForm::class);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {

            $user = $this->getDoctrine()->getManager()->getRepository(User::class)
                ->findOneByIdentifier($form->get('identifier')->getData());
            if ($user == null) {
                $form->get('identifier')
                    ->addError(new FormError($translator->trans('error.user_not_found')));
            } else {
                try {
                    if ($user->getEmail() == null || $user->getEmailVerifiedAt() == null) {
                        $form->addError(new FormError($translator->trans('error.recov-
ery.email_not_bound')));
                    } else {
                        $token = $service->generateResetToken($user);
                        $url = $urlGenerator->generate('security_recovery_password', [
                            'token' => $token->getToken(),
                        ], UrlGeneratorInterface::ABSOLUTE_URL);
                        $email = (new TemplatedEmail())
                            ->from($this->emailSenderAddress)
                            ->to($user->getEmail())
                            ->subject($translator->trans('email.verify.subject'))
                            ->htmlTemplate('emails/password_reset.html.twig')
                            ->context([
                                'url' => $url
                            ]);
                        $mailer->send($email);
                        return $this->render('security/email_verification_sent.html.twig');
                    }
                } catch (RecentPasswordResetRequestAlreadyExistsException $e) {
                    $form->addError(
                        new FormError($translator->trans('error.recent_password_reset_request_already_ex-
ists')));
                } catch (Exception $e) {
                    $form->addError(
                        new FormError($translator->trans('error.recovery.unknown'))
                    );
                }
            }
        }
        return $this->render('security/recovery_request.html.twig', [
            'requestForm' => $form->createView()
        ]);
    }
}

```

```

* @Route(path = "/recovery/password", name = "security_recovery_password")
*
* @param Request $request
* @param GuardAuthenticatorHandler $guardHandler
* @param FormAuthenticator $authenticator
* @param PasswordResetService $service
* @param UserPasswordEncoderInterface $passwordEncoder
* @return Response|null
* @throws ExpiredPasswordResetTokenException
* @throws InvalidPasswordResetTokenException
*/
public function password(Request $request, GuardAuthenticatorHandler $guardHandler,
                        FormAuthenticator $authenticator, PasswordResetService $service,
                        UserPasswordEncoderInterface $passwordEncoder, EntityManagerInterface $em) {

    $form = $this->createForm(PasswordChangeForm::class);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $fullToken = $request->getSession()->get('recovery_token');
        if (!$fullToken) {
            throw new InvalidPasswordResetTokenException();
        }
        $user = new User();
        $password = $passwordEncoder->encodePassword(
            $user,
            $form->get('password')->getData()
        );
        $userId = $service->updateUserPassword($fullToken, $password);

        $user->setId($userId);

        $user = $em->getRepository(User::class)->find($userId);

        return $guardHandler->authenticateUserAndHandleSuccess(
            $user,
            $request,
            $authenticator,
            'main' // firewall name in security.yaml
        );
    }

    $fullToken = $request->get('token');
    $request->getSession()->set('recovery_token', $fullToken);

    return $this->render('security/recovery_password.html.twig', [
        'passwordForm' => $form->createView()
    ]);
}

}

```

PermissionController.php:

```

class PermissionController extends AbstractController
{
    /**
     * @Route("/permission", methods={"GET"})
     * @Security("is_granted('IS_AUTHENTICATED_REMEMBERED') and is_granted('permission.permissions.manage')")
     */
    public function getPermissions(PermissionRepository $repository) {
        // $this->denyAccessUnlessGranted('IS_AUTHENTICATED_REMEMBERED');
        $users = $repository->getUsers();
        $groups = $repository->getGroups();
        $holders = $users;
        foreach ($groups as $group) {
            array_push($holders, $group);
        }
        $data = new PermissionsDataSnapshot($holders, $repository->getTracks());
        $response = JsonResponse::fromJsonString(json_encode($data));
        //debug
        $response->headers->set('Access-Control-Allow-Origin', '*');
        return $response;
    }
}

```

```

/**
 * @Route("/permission", methods={"POST"})
 * @Security("is_granted('IS_AUTHENTICATED_REMEMBERED') and is_granted('permission.permissions.manage')")
 */
public function postPermissions(Request $request, PermissionRepository $repository) {
    $modification = json_decode($request->getContent(), true);
    $changes = [];
    foreach ($modification['changes'] as $change) {
        if (array_key_exists($change['type'], $changes)) {
            array_push($changes[$change['type']], $change);
        } else {
            $changes[$change['type']] = [$change];
        }
    }
    if (array_key_exists('track', $changes)) {
        $repository->setTracks($changes['track']);
    }
    if (array_key_exists('user', $changes)) {
        $repository->setUsers($changes['user']);
    }
    if (array_key_exists('group', $changes)) {
        $repository->setGroups($changes['group']);
    }

    $repository->deleteGroups($modification['groupDeletions']);
    $repository->deleteTracks($modification['trackDeletions']);

    $response = new Response('OK', Response::HTTP_OK,
        //debug
        ['Access-Control-Allow-Origin' => '*']);
    return $response;
}

//debug
/**
 * @Route("/permission", methods={"OPTIONS"})
 */
public function options() {
    return new Response('OK', Response::HTTP_OK, [
        'Access-Control-Allow-Origin' => '*',
        'Access-Control-Allow-Methods' => 'POST, GET, OPTIONS',
        'Access-Control-Allow-Headers' => '*'
    ]);
}
}

```

EmailController.php:

```

class EmailController extends AbstractController
{
    private $emailSenderAddress;

    public function __construct($emailSenderAddress)
    {
        $this->emailSenderAddress = $emailSenderAddress;
    }

    /**
     * @Route("/email/bind", name="security_email_bind")
     * @IsGranted("IS_AUTHENTICATED_REMEMBERED")
     * @param Request $request
     * @param TranslatorInterface $translator
     * @return Response
     */
    public function bind(Request $request, TranslatorInterface $translator) : Response
    {
        $user = $this->getUser();
        if (!$user instanceof User || $user->getEmail() != null) {
            return $this->redirectToRoute("main");
        }

        $form = $this->createForm(EmailBindForm::class, $user);
        $form->handleRequest($request);
    }
}

```

```

        if ($form->isSubmitted() && $form->isValid()) {
            try {
                $this->getDoctrine()->getManager()->flush();
                return $this->redirectToRoute('security_email_verify');
            } catch (UniqueConstraintViolationException $e) {
                $form->get('email')
                    ->addError(new FormError($translator->trans('error.register.same_email_already_ex-
ists')));
            }
        }
        return $this->render('security/email_bind.html.twig', [
            'emailBindForm' => $form->createView()
        ]);
    }

    /**
     * @Route("/email/verify", name="security_email_verify")
     *
     * @param Request $request
     * @param MailerInterface $mailer
     * @param UrlGeneratorInterface $urlGenerator
     * @param UrlSigner $urlSigner
     * @param TranslatorInterface $trans
     * @return Response
     * @throws TransportExceptionInterface
     */
    public function verify(Request $request, MailerInterface $mailer, UrlGeneratorInterface $urlGenerator,
        UrlSigner $urlSigner, TranslatorInterface $trans) : Response
    {
        if ($request->get('email') != null && $request->get('id') != null) {
            $user = $this->getDoctrine()->getManager()->getRepository(User::class)->find($request-
>get('id'));
            if ($user->getEmail() != $request->get('email') || $user->getEmailVerifiedAt() != null) {
                return $this->redirectToRoute('main');
            }
            if ($urlSigner->validate($request->getUri())) {
                $user->setEmailVerifiedAt(new DateTime());
                $this->getDoctrine()->getManager()->flush();
                $request->getSession()->getFlashBag()->add('success', $trans->trans('user.alert.email_ver-
ified'));
            } else {
                $request->getSession()->getFlashBag()->add('error',
                    $trans->trans('user.alert.email_verification_error',
                        ['%url%' => $urlGenerator->generate('security_email_verify')]));
            }
            return $this->redirectToRoute('main');
        }

        $this->denyAccessUnlessGranted('IS_AUTHENTICATED_REMEMBERED');

        $user = $this->getUser();
        if (!$user instanceof User) {
            return $this->redirectToRoute('main');
        }
        if ($user->getEmail() == null) {
            return $this->redirectToRoute('security_email_bind');
        }
        if ($user->getEmailVerifiedAt() != null) {
            return $this->redirectToRoute('main');
        }

        //параметры должны быть расположены в лексикографическом порядке
        $url = $urlGenerator->generate('security_email_verify', [
            'email' => $user->getEmail(),
            'id' => $user->getId()
        ], UrlGeneratorInterface::ABSOLUTE_URL);
        $url = $urlSigner->sign($url, 1);

        $email = (new TemplatedEmail())
            ->from($this->emailSenderAddress)
            ->to($user->getEmail())
            ->subject($trans->trans('email.verify.subject'))
            ->htmlTemplate('emails/email_verification.html.twig')
            ->context([
                'username' => $user->getUsername(),
                'url' => $url
            ])
    }

```

```

    });
    $mailer->send($email);
    return $this->render('security/email_verification_sent.html.twig');
}

```

```

}

```

UploadController.php:

```

class UploadController extends AbstractController
{
    /**
     * @Route("/admin/upload/image", name="upload_image")
     * @Security("is_granted('IS_AUTHENTICATED_REMEMBERED') and is_granted('permission.blog.post.create')")
     */
    public function uploadImage(Request $request, UrlGeneratorInterface $urlGenerator, UploadRepository $repository)
    {
        if($request->isXmlHttpRequest()) {
            return new JsonResponse([
                'status' => 'Error',
                'message' => 'Error'
            ], Response::HTTP_BAD_REQUEST);
        }

        $form = $this->get('form.factory')->createNamed('upload', ImageUploadForm::class);
        $form->submit($request->request->all());

        if ($form->isSubmitted() && $form->isValid()) {
            $file = $request->files->get('image');
            $data = $form->getData();

            $newFilename = uniqid().'.'.$file->guessExtension();

            try {
                $file->move(
                    $this->getParameter('uploads_directory'),
                    $newFilename
                );
            } catch (FileNotFoundException $e) {
                return new JsonResponse([
                    'status' => 'Error',
                    'error' => 'Can\'t save file'
                ], Response::HTTP_INTERNAL_SERVER_ERROR);
            }

            $repository->addUpload(new Upload($newFilename, $data['title'], $data['description'], Upload::IMAGE_TYPE));

            return new JsonResponse([
                'status' => 'Ok',
                'url' => '/uploads/' . $newFilename,
                'title' => $data['title'],
                'desc' => $data['description']
            ], Response::HTTP_OK);
        }

        return new JsonResponse([
            'status' => 'Error',
            'errors' => $form->getErrors()
        ], Response::HTTP_BAD_REQUEST);
    }

    /**
     * @Route("/admin/uploads", methods={"GET"}, name="get_uploads")
     */
    public function getUploads(Request $request, UploadRepository $repository)
    {
        $uploads = array_map(function($upload) {
            return [
                'url' => '/uploads/' . $upload['upload_id'],
                'title' => $upload['title'],
                'desc' => $upload['description']
            ];
        }, $repository->getUploads());
        return new JsonResponse($uploads);
    }
}

```

```
    }
}
```

BlogController.php:

```
class BlogController extends Controller
{
    const REFERENCED_UPLOAD_PATTERN = '!\\[^[!\\[\\]]*\\(\\/uploads\\/([^(]*)\\)\\/';
    const POSTS_PER_PAGE = 2;

    /**
     * @Route("/", name="main")
     * @param EntityManagerInterface $em
     * @return Response
     */
    public function main(EntityManagerInterface $em)
    {
        return $this->page(1, $em);
    }

    /**
     * @Route("/blog/page/{page}", name="blog")
     *
     * @param $page
     * @param EntityManagerInterface $em
     * @return Response
     */
    public function page(int $page, EntityManagerInterface $em) {
        $repository = $em->getRepository(Post::class);
        $count = $repository->getPublicCount();
        return $this->render('index.html.twig', [
            'posts' => $repository->getPublicPaged(self::POSTS_PER_PAGE, $page),
            'current_page' => $page,
            'total_pages' => ceil($count / self::POSTS_PER_PAGE)
        ]);
    }

    /**
     * @Route("/blog/post", methods={"GET", "POST"}, name="create_post")
     * @Security('is_granted('IS_AUTHENTICATED_REMEMBERED') and is_granted('permission.blog.post.create')')
     *
     * @param Request $request
     * @param UploadRepository $uploadRepository
     * @param EntityManagerInterface $em
     * @param TranslatorInterface $trans
     * @return Response
     * @throws ConnectionException
     */
    public function createPost(Request $request, UploadRepository $uploadRepository, EntityManagerInterface $em,
                               TranslatorInterface $trans)
    {
        $user = $this->getRememberedUser();

        $form = $this->createForm(CreatePostForm::class);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $content = $form->get('content')->getData();
            $usedUploads = [];
            if (preg_match_all(self::REFERENCED_UPLOAD_PATTERN, $content, $usedUploads)) {
                $usedUploads = $usedUploads[1];
            } else {
                $usedUploads = [];
            }
            $em->getConnection()->beginTransaction();
            try {
                $post = new Post();
                $post->setTitle($form->get('title')->getData());
                $post->setContent($content);
                $post->setImage($form->get('image')->getData());
                $post->setState('draft');
                $post->setAuthor($user);
                $em->persist($post);
                $em->flush();
                $uploadRepository->addReference('post', $post->getId(), $usedUploads);
            }
        }
    }
}
```



```

        $em->getConnection()->commit();
    } catch (ConnectionException $e) {
        $em->getConnection()->rollBack();
        throw $e;
    }
    $request->getSession()->getFlashBag()->add('success', $trans->trans('blog.post.alert.success-
fully_created'));
    // clear form
    $form = $this->createForm(CreatePostForm::class);
}
return $this->render('blog/create_post.html.twig', [
    'createForm' => $form->createView()
]);
}

/**
 * @Route("/blog/post/{id}", methods={"PUT"})
 * @param $id
 * @param Request $request
 */
public function updatePost($id, Request $request)
{
}

/**
 * @Route("/blog/post/{id}", methods={"GET"}, name="view_post")
 * @param $id
 * @param EntityManagerInterface $em
 * @return Response
 */
public function showPost($id, EntityManagerInterface $em)
{
    return $this->render('blog/post.html.twig', [
        'post' => $em->getRepository(Post::class)->find($id)
    ]);
}
}

```