Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«Оренбургский государственный педагогический университет»

Физико-математический факультет

Кафедра информатики и методики преподавания информатики

**Курсовая работа**

**по дисциплине «WEB-программирование»**

**разработка Информационной системы коммуникации**

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

Профиль подготовки: общий

Форма обучения: очная

Выполнил студент

Скробов Никита Вячеславович

4 курс, группа МОАИС

|  |  |
| --- | --- |
| Научный руководитель: | |
| Доц. Денисова Людмила Викторовна | |
|  |  |
| (оценка) | (подпись) |
| «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2019 г. | |

Оренбург, 2019 г.

**Оглавление**

[**Оглавление** 2](#_Toc536323426)

[Введение 3](#_Toc536323427)

[Глава 1. Используемый инструментарий 4](#_Toc536323428)

[1.1 Framework Laravel 4](#_Toc536323429)

[1.2 Выбор инструментария 8](#_Toc536323430)

[Глава 2. Программная разработка ИС на Laravel 9](#_Toc536323431)

[2.1 Создание таблиц БД 9](#_Toc536323432)

[2.2 Маршрутизация приложения 16](#_Toc536323433)

[2.3 Создание view представлений 17](#_Toc536323434)

[2.4 Администрирование сайта 20](#_Toc536323435)

[Заключение 22](#_Toc536323436)

[Список литературы 23](#_Toc536323437)

# 

# Введение

В современно мире у каждой, даже самой маленькой организации есть свои web-ресурсы, которые они используются для общения с пользователями. Особенно web-сервисы нужны начинающим инди разработчикам, которым крайне необходимо быстро получать информацию, о найденных багах и ошибках в разрабатываемых продуктах.

Так же своевременно уведомлять сообщество о внесенных изменениях в проекты и исправленных ошибках. Данный подход позволяет наладить прямое общение между разработчиков и пользователем, которые использует их продукт.

Таким образом, современным развивающимся компаниям крайне необходимо иметь свои web-ресурсы, чтобы облегчить коммуникацию со своими пользователями, в частности инди разработчикам программного обеспечения.

Таким образом, объектом работы являются информационные системы.

Предмет работы – проектирование информационной системы.

Цель работы – построить информационную систему коммуникации пользователей и разработчиков.

Таким образом, в работе ставятся следующие задачи:

1. Изучить предметную область.
2. Разработать модель базы данных.
3. Организовать выполнение PHP сценариев.
4. Реализовать базовый функционал ИС.

# Глава 1. Используемый инструментарий

# 1.1 Framework Laravel

Для решения поставленной задачи было решено использовать php-framework Laravel для работы с back-end частью проекта. Laravel — это web-framework с открытым кодом общего назначения, предназначенный для разработки с использованием архитектурной модели MVC (англ. Model View Controller — модель-представление-контроллер). [2].

Laravel определяют, как full-stack framework, поскольку он может работать со всеми возможными частями web-приложения. К таким частям можно отнести: web-сервисы, управление базами данных, генерация HMTL, маршрутизация, кеширование, аутентификация, локализация, управление группами пользователей, сборка css и js файлов, инкапсуляция ресурсов и многое другое. С Laravel возможно взаимодействовать, как через встроенную утилиту командной строки artisan, управляя средой проекта Laravel, так и просто с помощью настройки конфигурационных файлов вручную. Примером служит конфигурационный файл .env, в котором возможно настроить подключение к БД, почтовому клиенту и другим возможностям. Но как правило оба способа используются параллельно.

Еще одна интересная особенность Laravel заключается в том, что она накладывает некоторые довольно серьезные ограничения на структуру веб-приложений. Но эти ограничения помогают упростить процесс разработки. Laravel отличается от многих других вертикально интегрированных сред тем, что она конфигурациям предпочитает конвенции. В то время, как в некоторых web-framework, написанных на Java, Python или PHP, часто требуется редактирование очень большого объема файлов с конфигурациями, то для начала работы с Laravel никаких настроек практически не требуется, разве что кроме нескольких строк на PHP. Стоит отметить, что большое количество конфигурационных файлов в Laravel написано на PHP. Такой подход в избегании файлов конфигурации позволяет приложениям, написанным под разные нужды, поддерживать узнаваемую структуру кода.

Учитывая все выше сказанное, неудивительно, что проекты под Laravel имеют практически одинаковую файловую структуру – такую, в которой каждый функциональный блок имеет свое назначенное место. Такая структура совсем не обязательна для полноценной работы программы, но ненавязчивое предложение такого способа хранения файлов в проекте, гарантирует более-менее унифицированную архитектуру среди всех приложений Laravel.

Таблица 1.1

Файловая структура

|  |  |
| --- | --- |
| /app/ | Состоит из контроллеров MVC модели, моделей ORM. |
| /config/ | В этой папке находятся файлы для настройки любых модулей Laravel: начиная с кеширования и БД, заканчивая почтовым модулем и файловой системой. |
| /database/ | В данной папке хранятся файлы миграций, происходит генерация БД из моделей. |
| /public/ | Единственная доступная папка для всего WWW. На этот каталог должен ссылаться web-сервер. Он содержит загрузочный файл index.php, который запускает ядро Laravel. Так же в этом каталоге могут храниться любые статические файлы, такие как JavaScript, CSS, изображения и другие. |
| /resources/ | В данном месте хранятся ресурсы для работы с фронтендом: исходные JavaScript и css файлы для сборок, файлы локализации и файлы представлений. |
| /routes/ | Здесь хранятся файлы для конфигурации модуля маршрутизации, а именно web.php. |
| /storage/ | Эта папка предназначена для хранения генерируемых работой Laravel файлов: сессий, кеша и логов. |
| /tests/ | В этой папке хранятся файлы модульных тестов. |
| /vendor/ | Это место для всех сторонних частей программы. В типичном приложении Laravel в него включается исходный код Laravel, его зависимости, а также плагины, содержащие дополнительные предварительно упакованные функции. |
| .env | Из этого файла Laravel берет environment-переменные, например, здесь, с помощью изменения значения переменной DEBUG, можно задать состояние Laravel приложения – production или debug. |

Стандартная файловая структура состоит из большого количества каталогов.

Laravel следует архитектурному паттерну MVC (Model-View-Controller) который обеспечивает разделений бизнес-логики от логики представления, связанной с графическим пользовательским интерфейсом GUI. В случае с web-приложениями Laravel бизнес-логика обычно состоит из моделей данных для таких вещей, как пользователи, сообщения в блогах, а GUI, это просто web-страница в web-браузере. Шаблон проектирования MVC очень популярен в пространстве web-разработки.

В шаблоне MVC существует три компонента:

Модель – каскад, вокруг которого работает приложение. Модели основаны на реальных предметах, таких как человек, событие, билет или продукт. Самый простой пример модели: запись в блоге. Модели как правило являются постоянными и хранятся вне приложения, часто в базе данных. Модель - это больше, чем просто данные; Модель — это последний рубеж для защиты и представления данных.

Представление - визуальное представление модели с учетом какого-либо контекста. Обычно это результирующая разметка, которую framework передает браузеру, например, HTML страница, представляющая собой отображение информации в браузере. Уровень представления отвечает за создание пользовательского интерфейса, обычно на основе данных в модели.

Контроллер – связывающее звено, обеспечивающее связь между представлением и моделью. Контроллер отвечает за обработку входных данных, действие на модель и принятие решения о том, какие действия должны быть выполнены, например, рендеринг представления или перенаправление на другую страницу.

Типичное Laravel приложение состоит из вышеперечисленных компонентов MVC (рис. 1.1).

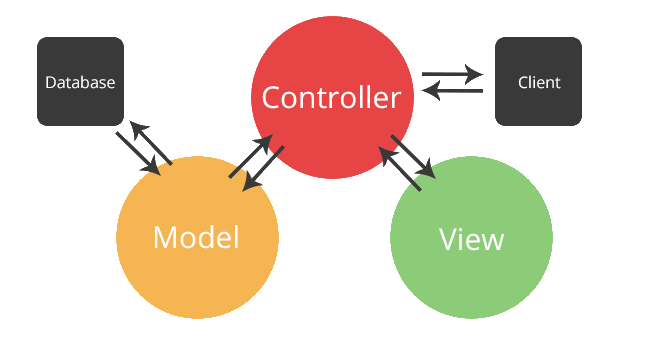


Рис. 1.1. MVC

При взаимодействии с приложением Laravel браузер отправляет запрос, который принимается web-сервером и передается в механизм маршрутизации Laravel. Маршрутизатор (Router) Laravel получает запрос и перенаправляет его на соответствующий метод класса контроллера, основанный на шаблоне URL маршрутизации.

Затем управляет передается в класс контроллера. В некоторых случаях контроллер немедленно выдает представление, которое является шаблоном, который преобразуется в HTML и отправляется обратно в браузер. Чаще всего для динамических сайтов контроллер взаимодействует с моделью, которая представляет собой объект PHP, который в свою очередь представляет элемент приложения (например, пользователя, мероприятие) и отвечает за связь с базой данных. После вызова модели контроллер затем отображает окончательный вид (HTML, CSS и изображения) и возвращает полную web-страницу в браузер пользователя.

Laravel продвигает идею о том, что модели, представления и контроллеры должны быть отделены друг от друга, сохраняя код для каждого из этих элементов в виде отдельных файлов в отдельных каталогах.

## 1.2 Выбор инструментария

Прежде чем приступить непосредственно к разработке необходимо определиться с технологическим стеком проекта. Прежде всего был проработан основной функционал сервиса для того, чтобы иметь основу для принятия решений о выборе технологий.

Перед разработкой приложения были сформулированы следующие требования к системе:

Масштабируемость. Возможность повышать пропускную способность сервиса и добавлять новый функционал без необходимости сильно менять существующий код.

Было решено вести разработку в IDE компании JetBrains. Их продукты отличаются большим функционалом и широкими возможностями для модификаций с помощью встроенной системы модулей. В PhpStorm, среде, где велась разработка, имелись встроенные инструменты для работы с модулями сборки, репозиториями и тестами, что значительно ускорило процесс разработки. В качестве репозитория была выбрана открытая платформа GitHub.

Перед тем, как выбрать back-end framework приложения, необходимо определиться с языком программирования. Из всех характеристик языка программирования нас интересует, является ли язык интерпретируемым или компилируемым. По определению, компилируемые языки программирования задачи выполняют быстрее, чем интерпретируемые, и есть много решений на базе компилируемо-интерпретируемых языков: Java и C#. Учитывая значительный опыт работы с языком C#, казалось, должно было быть принято решение использовать ASP.NET в качестве web-framework, но решение на ASP.NET может быть развернуто только на специальных серверах, с дорогостоящим программным обеспечением. Следовательно, выбор пал на Laravel, потому что кроссплатформенность стала решающим фактором. Проект разворачивается на любых операционных системах, и предоставляет возможность выбора платформы инди разработчикам.

С интерпретируемыми языками ситуация проще. Для того, чтобы протестировать изменения достаточно всего лишь обновить страницу в браузере, тем более, что этот процесс можно автоматизировать, подключив инструмент, который при изменении файлов разработки, сам обновляет страницу. Laravel framework включает в себя все эти характеристики, плюс удовлетворяет второму пункту из требований к системе - безопасности. Laravel имеет встроенные системы для работы с базами данных, аутентификацией, регистрацией и управлением ролями пользователей.

В качестве базы данных была выбрана реализация SQL от Oracle: MySQL. За последнее время в моду вошли не реляционные базы данных, такие как MongoDB и NoSQL. Следует отметить, что не реляционные базы данных очень хорошо применимы к узкому спектру задач, но значительно уступают реляционным, если речь заходит о «классических» базах данных. Такая база разрабатывается без знаний обо всех возможных применениях в программе. Правила нормализации как раз призваны подготовить базу данных к любым запросам от пользователя с наименьшими затратами по времени и памяти [4].

# Глава 2. Программная разработка ИС на Laravel

# 2.1 Создание таблиц БД

Во время разработки проекта было решено использовать миграции, чтобы ускорить время разработки БД проекта. С помощью простого и выразительного PHP кода можно с легкостью определить структуру таблиц БД использовав особенность php-framework Laravel миграции. С помощью миграции быстро и просто создаются, и связывают таблицы БД.

Для создания различных классов проекта используется интерфейс Artisan. Он избавляет от ручной генерации кода при создании проектов на Laravel. Для запуска миграции используется консольная команда make:migration, она создаст нужную миграцию, в которой можно определить столбцы таблицы create \_table:

php artisan make:migration create\_table

Миграция автоматически попадает в каталог database/migrations нашего проекта. Команда make:migration по умолчанию добавляет автоинкрементный ID и метки времени к файлу миграции.

Создадим миграции для нашего проекта с помощью команды make:migration и заполним их.

php artisan make:migration specifications

Перейдем в каталог database/migrations и отредактируем содержимое нашей миграции, чтобы создать таблицу спецификации, в которой будут храниться требования к компьютеру, которые потребуется для запуска игр компании.

public function up()  
{  
 Schema::create('specifications', function (Blueprint $table) {  
 $table->increments('id');  
 $table->string('name');  
 $table->timestamps();  
  
 });  
}  
public function down()  
{  
 Schema::dropIfExists('specifications');  
}

Создаем таблицу разработчики, она нужна для отображения в проекте разработчиков, которые приняли участие в разработке данного продукта.  
php artisan make:migration developers

public function up()  
{  
 Schema::create('developers', function (Blueprint $table) {  
 $table->increments('id');  
 $table->string('name');  
 $table->timestamps();  
  
 });  
}  
public function down()  
{  
 Schema::dropIfExists('develoeprs');  
}

Создаем таблицу жанры, для указания жанра в созданных играх.  
php artisan make:migration genre

public function up()  
{  
 Schema::create('genre', function (Blueprint $table) {  
 $table->increments('id');  
 $table->string('name');  
 $table->timestamps();  
  
 });  
}  
public function down()  
{  
  
 Schema::disableForeignKeyConstraints();  
 Schema::dropIfExists('genre');  
 Schema::enableForeignKeyConstraints();  
}

Создаем таблицу платформы, данная таблица будет содержать какая платформа потребуется для запуска продуктов компании.  
php artisan make:migration platforms

public function up()  
{  
 Schema::create('platforms', function (Blueprint $table) {  
 $table->increments('id');  
 $table->string('name');  
 $table->timestamps();  
 });  
}  
public function down()  
{  
 Schema::dropIfExists('platforms');  
}

Создаем таблицу игры, в которой хранятся все игровые проекты компании.  
php artisan make:migration game

public function up()  
{  
 Schema::create('game', function (Blueprint $table) {  
 $table->increments('id');  
 $table->string('name');  
 $table->date('dateRelease');  
 $table->longText('img');  
 $table->longText('about');  
 $table->integer('idSpecifications')->unsigned();  
 $table->foreign('idSpecifications')->references('id')->on('specifications');  
 $table->integer('idDevelopers')->unsigned();  
 $table->foreign('idDevelopers')->references('id')->on('developers');  
 $table->timestamps();  
 });  
}  
public function down()  
{  
 Schema::dropIfExists('game');  
}

Создаем промежуточную таблицу жанры\_игр, чтобы в дальнейшем организовать связь многие ко многим между таблицами.  
php artisan make:migration genre\_game

public function up()  
{  
 Schema::create('genreGame', function (Blueprint $table) {  
 $table->increments('id');  
 $table->integer('idGenre')->unsigned();  
 $table->foreign('idGenre')->references('id')->on('genre');  
  
 $table->integer('idGame')->unsigned();  
 $table->foreign('idGame')->references('id')->on('game');  
  
 $table->timestamps();  
  
 });  
}  
public function down()  
{  
 Schema::dropIfExists('genreGame');  
}

Создаем промежуточную таблицу игры\_платформы, для создания связи многие ко многим между таблицами.  
php artisan make:migration game\_platforms

public function up()  
{  
 Schema::create('gamePlatforms', function (Blueprint $table) {  
 $table->increments('id');  
 $table->integer('idPlatforms')->unsigned();  
 $table->foreign('idPlatforms')->references('id')->on('platforms');  
 $table->integer('idGame')->unsigned();  
 $table->foreign('idGame')->references('id')->on('game');  
 $table->timestamps();  
  
 });  
}  
public function down()  
{  
 Schema::dropIfExists('gamePlatforms');  
}

Создадим таблицу для обратной связи, чтобы пользователи могли напрямую общаться с разработчиками.  
php artisan make:migration feedback

public function up()  
{  
 Schema::create('feedback', function (Blueprint $table) {  
 $table->increments('id');  
 $table->string('name');  
 $table->string('email');  
 $table->longText('message');  
 $table->timestamps();  
 });  
}  
public function down()  
{  
 Schema::dropIfExists('feedback');  
}

Создадим таблицу сотрудников компании, для удобства администратор сайта может добавлять и удалять, редактировать сотрудников, которые в любой момент могу уволиться или устроиться на работу.  
php artisan make:migration employees

public function up()  
 {  
 Schema::create('employees', function (Blueprint $table) {  
 $table->increments('id');  
 $table->string('name');  
 $table->string('about');  
 $table->string('email');  
 $table->longText('img');  
 $table->timestamps();  
 });  
 }  
 public function down()  
 {  
 Schema::dropIfExists('employees');  
 }  
}

Чтобы запустить наши миграции используется команда php artisan migrate. Эта команда создаст все наши таблицы БД.

# 2.2 Маршрутизация приложения

Стоит отдельно выделить маршрутизацию, с помощью нее будут строиться все маршруты нашего web-сайта.

В данной работе будет один маршрут, использующий замыкание: маршрут, представляющий из себя лендинг для гостей приложения. Заполнив маршрут. По этому маршруту нужно отрисовывать HTML-шаблон, который содержит мастер страницу «master.blade.php».

И еще один замкнутый маршрут будет для администраторов содержащий мастер страницу «masterHome.blade.php».

В Laravel все view хранятся в каталоге resources/views, и можно использовать вспомогательную функцию view(), чтобы возвратить один из этих шаблонов по нашему маршруту:

Route::get('/', function () {

return view('blog');

});

Данный маршрут будет основным для всех пользователей. Для пользователей, которые обладают правами администратора созданы маршруты, и основной:

Route::get('/home', 'HomeController@index')->name('home');

Создадим маршруты для пользователей и администраторов, их маршруты будут отличаться проверкой на право доступа к определенным страницам. Ниже представлены несколько примеров маршрутов для администраторов:

Auth::routes();

Route::get('/home', 'HomeController@index')->name('home');  
route::post('/home/employeesDetail/{id}','employeesController@edit')->name('employees-edit');

После команды Auth::routes указываются наши маршруты администраторов. Все маршруты делятся на get и post, для отображения администраторам страницы используется get, а для отправки данных на back-end используется post.

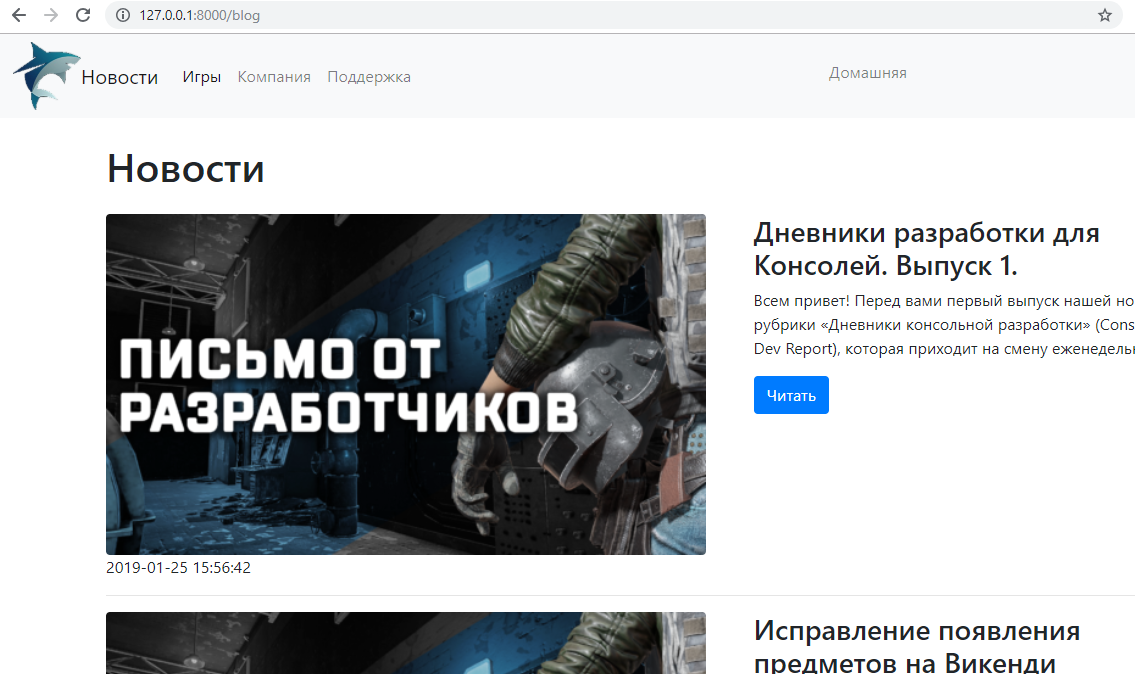
Для всех пользователей будут доступны следующие маршруты:

Route::get('/game','gameController@index')->name('game');  
Route::get('/blog','blogController@index')->name('blog');

## 2.3 Создание view представлений

View представления — это страницы web-сайта. Перейдем к созданию view представлений. Сначала создадим мастер страницу, на которой будет навигационная панель, единая для всех страниц, доступных пользователям.

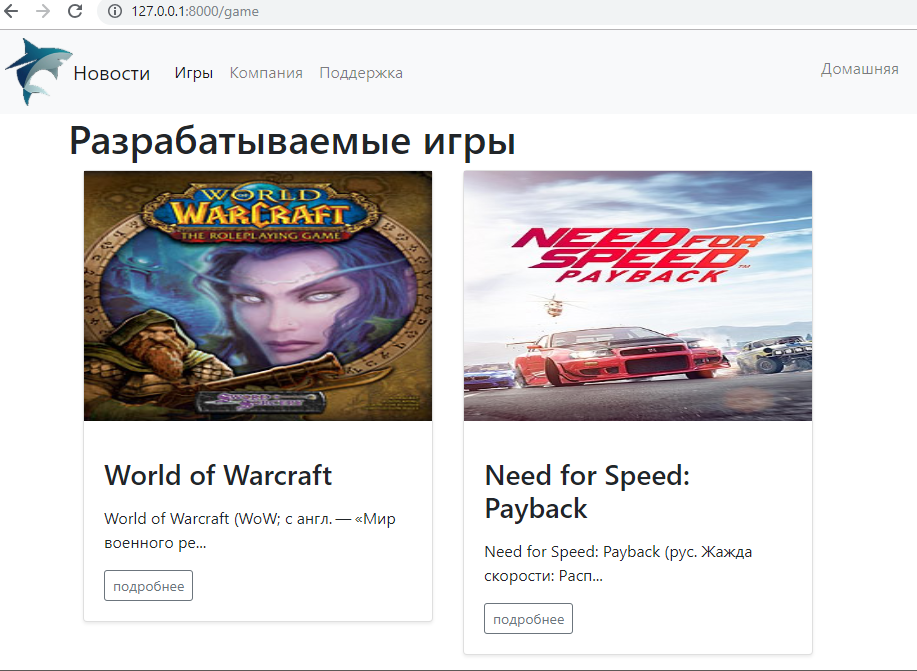
Входе написания проекта используется шаблонизатор blade с помощью, которого с легкостью и удается создать мастер страницу для нашего сайта. Создадим нашу основную страницу где будут размещаться новости от разработчиков (рис. 1.2).



1. Рис. 1.2. Страница новостей

Чтобы получить представление нашей страницы, мы обращаемся по маршруту /blog, который отправляет get запрос на контроллер с методом index, контроллер обрабатывает запрос и отправляет на наше представление данные, и мы получаем нашу страничку с данными, которые вытащили с БД.

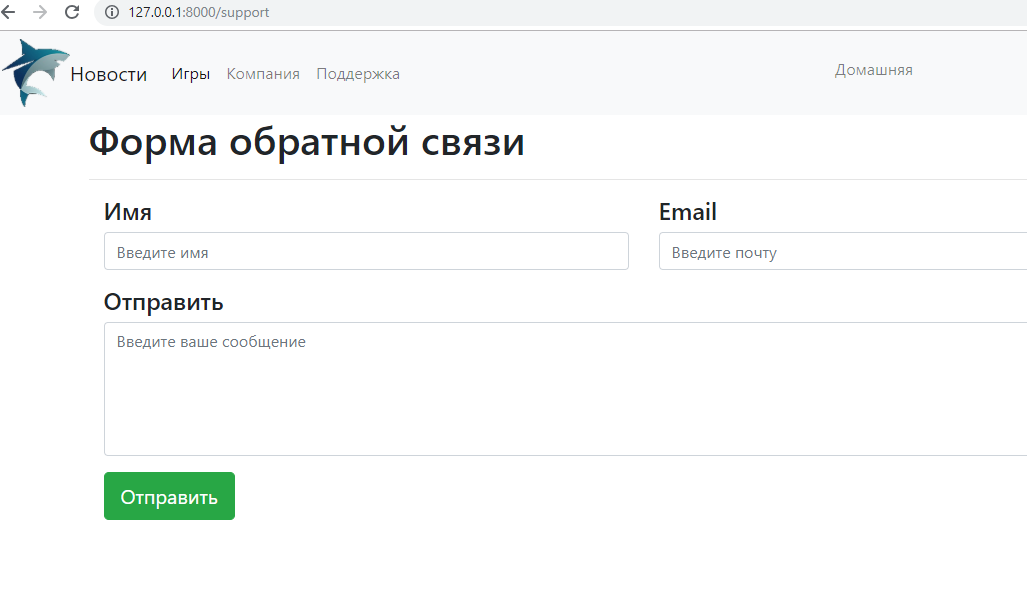
Для отображения игр, которые находятся в разработке создаем отдельную страницу (рис. 1.3).



1. Рис. 1.3. Страница игр

На данной странице отображаются все игры, которые находят в разработке у разработчиков. Страница доступна по адресу /game, но многие пользователи хотят подробно знать о том или ином продукте разработки, поэтому они переходят по кнопке подробнее, которая отправляет пользователей по адресу /game/{?}, где {?} является номером данной игры в базе.

Для полноценного общения с пользователями нам нужна страница для обратной связи, с помощью которой пользователи будут уведомлять разработчиков о найденных багах и неисправностях. Страница для обратной связи (рис. 1.4).

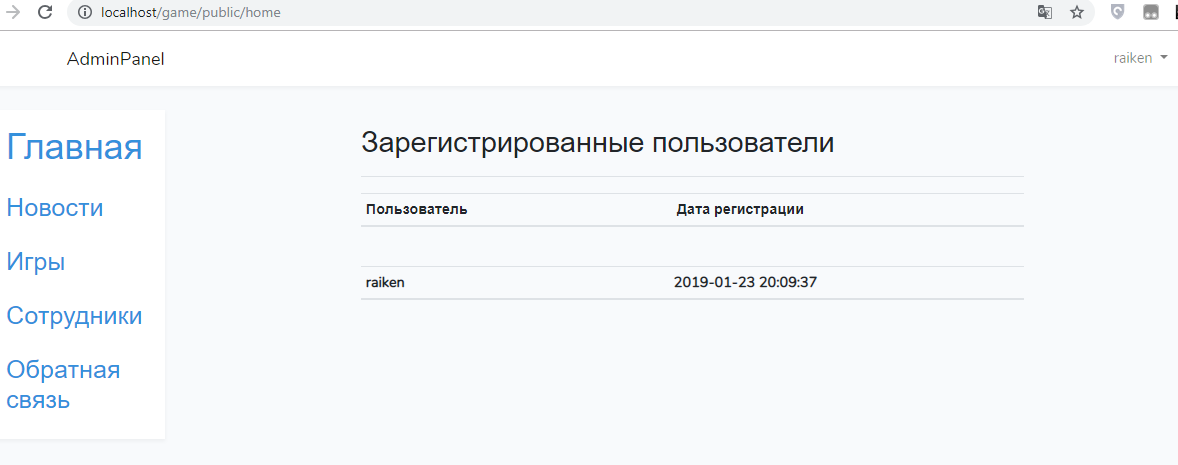


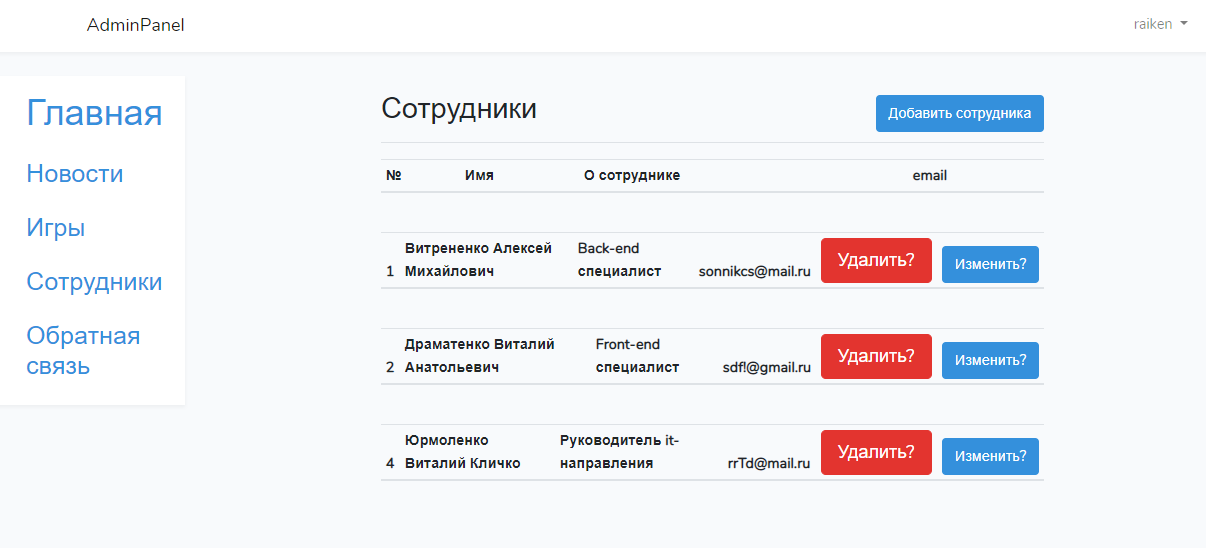
1. Рис. 1.4. Страница обратной связи

# 2.4 Администрирование сайта

Для создания панели администратора, воспользуемся шаблонизатором .blade.php. Для создания мастер страницы, на которой будет вся навигация панели администратора.

При авторизации нас перекидывает на главную страницу панели администратора. Где отображаются все пользователи, зарегистрированные на данный момент (рис. 1.5).



1. Рис. 1.5. Панель администрирования
2. На главной странице находится таблица зарегистрированных пользователей. Для редактирования сотрудников компании перейдем в панели навигации на вкладку «Сотрудники». На данной странице можно добавлять, удалять сотрудников или изменять сведения о сотруднике (рис.1.6).
3. 
4. Рис. 1.6. Главная страница панели администрирования

Перейдя на вкладку новости можно добавлять, и редактировать статьи на сайте. Для разработчиков главное быстро, и своевременно обновлять информацию о своих продуктах. (рис. 1.7).

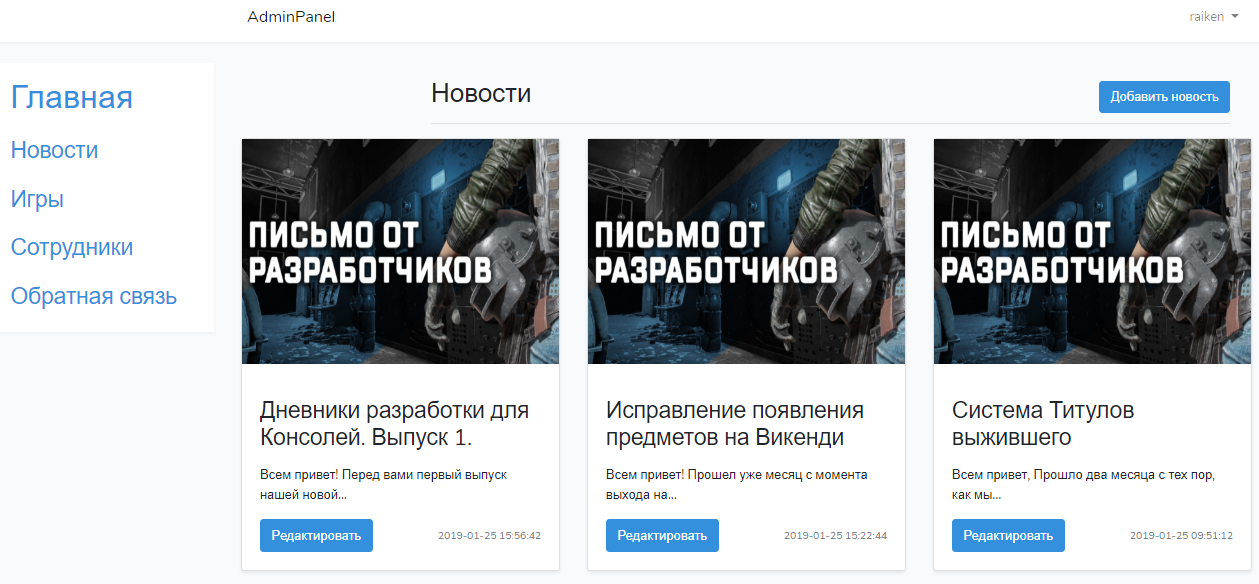


Рис. 1.7. Страница редактирования всех новостей

# Заключение

В ходе разработки проекта были исследованы возможные подходы по написанию web-сервиса, а также подобраны подходящие инструменты для решения поставленной задачи и разработано web-приложение для инди разработчиков. Которое могут быстро и легко развернуть любые инди разработчики, на любой платформе, будь то сервера linux или windows, без дополнительных затрат, так как используемое программное обеспечение бесплатно и находится в открытом доступе. Каждая организация сможет внедрить, данное решение для получения обратной связи от сообщества и сообщений об ошибках в разрабатываемых проектах.

Так же весь процесс разработки и его результаты были в полной мере описаны и обоснованы. У проекта есть множество перспектив для развития и улучшения.

# Список литературы

1. Бенедетти Р. Изучаем работу с jQuery. - М.:Питер, 2012. - 512c.
2. Дронов, В. HTML 5, CSS 3 и Веб 2.0. Разработка современных Веб-сайтов. - М.:БХВ, 2014. - 416c.
3. Кузин, А.В., Демин, В.М. Компьютерные сети: учебное пособие. – 2-е изд. – М: ФОРУМ, 2008 – 267 стр. Карпова И.П. Базы данных: Учебное пособие И.П. Карпова. – СПб.; Питер, 2013. – 240 c.
4. Макфалин Б. PHP и MySQL. Исчерпывающее руководство. 2-е изд.- Спб.: Питер, 2014. - 544 с.
5. Прохоренок, Н. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н. Прохоренок. - СПб.: БХВ-Петербург, 2008. - 640 с.