Весці БДПУ. Серыя 3. 2017. № 3. С. 82–90.

УДК 004.4

ПРОГРАММНАЯ ПЛАТФОРМА LARAVEL ДЛЯ СОЗДАНИЯ ВЕБ-ОРИЕНТИРОВАННЫХ ПРИЛОЖЕНИЙ И СЕРВИСОВ

С. М. Гардейчик,
студент V курса физико-математического факультета БГПУ;

А. И. Шербаф,
кандидат физико-математических наук, доцент кафедры информатики и методики преподавания информатики БГПУ

Поступила в редакцию 30.05.17.

UDC 004.4

COMPUTING PLATFORM LARAVEL FOR BUILDING WEB-ORIENTED APPLICATIONS AND SERVICES

S. Gardeichik,
student of the 5th year, Faculty of Physics and Mathematics, BSPU

A. Sherbaf,
Candidate of Physics and Mathematics, Associate Professor of the Department of Informatics, BSPU

Received on 30.05.17.

В данной статье обсуждаются современные возможности разработки и проектирования веб-приложений, в частности, раскрывается понятие фреймворк (другими словами, каркас), которое включает в себя структуру программной системы, а также программное обеспечение, упрощающее разработку и объединение различных компонентов большого программного проекта. Перечислены особенности фреймворков и их преимущества. Достаточно подробно рассматривается PHP-фреймворк Laravel: описан менеджер зависимостей для PHP Composer, описываются способы маршрутов в Laravel, связующее программное обеспечение для установления взаимодействия между различными приложениями, а также системы контроля версий для баз данных проекта. Отдельное внимание уделяется архитектурной модели MVC.

Ключевые слова: веб-приложение, архитектура MVC, PHP-фреймворк Laravel.

In the given article, contemporary opportunities for the building and design of web-applications, in particular a notion of framework which includes a program system structure as well as software simplifying the building and integration of various components of a big project are discussed. Special features and their advantages of frameworks are defined. The given PHP-framework Laravel, in particular dependency manager for PHP, Composer, Middleware that provides services to software applications beyond those available from the operating system, as well as ways of systems of version control for a project database are described. A special attention is paid to MVC architecture.

Keywords: web-application, MVC architecture, PHP-framework Laravel.

Введение. Рассмотрим подробнее понятие «фреймворк». В Википедии фреймворк определяется следующим образом: программная платформа, определяющая структуру программной системы, иными словами, программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

Веб-приложения часто используют стандартизированную структуру организации компонентов. Одним из главных преимуществ использования фреймворков является то, что при разработке проекта на фреймворках создание структуры значительно упрощается. Фреймворк можно трактовать как набор готовых функций, процедур и многого другого для избавления программистов от рутинной работы.

Фреймворк отличается от библиотеки тем, что библиотека используется в программном продукте как набор подсистем близкой функциональности, она не влияет на архитектуру основного программного продукта, никакие ограничения на нее не накладываются. Фреймворк устанавливает правила построения архитектуры приложения, на начальном этапе разработки форми-

рует каркас, который можно расширять и изменять согласно заданным требованиям. Фреймворк может включать вспомогательные программы, библиотеки кода, язык сценариев и другое программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

При разработке веб-приложений часто возникают задачи, для многих из которых были найдены универсальные решения, называемые шаблонами проектирования. Использование шаблонов проектирования значительно экономит время, а также способствует выработке общей стратегии построения веб-приложений и упрощению процесса создания документации для создаваемых скриптов. Фреймворки предоставляют четкую структуру приложения и реализуются с использованием так называемых патернов (шаблонов) проектирования.

Фреймворк можно рассматривать как множество конкретных и абстрактных классов с определенными способами взаимоотношений между ними. Конкретные классы обычно реализуют взаимные отношения между классами, а абстрактные классы представляют собой точки расширения, в которых можно использовать или адаптировать постоянные части. Для обеспечения расширения возможностей обычно используются техники объектно-ориентированного программирования (например, части приложения могут наследоваться от базовых классов фреймворка). Для реализации проекта часто разработчики сталкиваются с проблемой выбора:

- использовать коробочные (коммерческие) CMS (Content Management System – системы управления содержимым) и адаптировать их под разрабатываемый проект;
- воспользоваться фреймворком и на его основе реализовать требуемый функционал;
- использовать язык программирования и писать код «с нуля».

Создание проекта на языке программирования требует значительных временных затрат, тем не менее этот подход часто используется для разработки масштабных проектов. Заметим, что уровень безопасности решений на фреймворках, как правило, значительно выше самостоятельно созданных систем. Разработка на CMS будет оправданна, если CMS содержит необходимые модули для реализуемого проекта, а встроенные в CMS процессы соответствуют заданным требованиям. Если для реализации проекта понадобится существенная доработка CMS, то есть ее возможности и процессы не позволяют достичь требуемого результата, то целесообразно использовать фреймворки.

Основная часть. Выбор фреймворка для создания веб-приложения является непростой задачей. В настоящее время все большую популярность среди специалистов приобретает фреймворк Laravel – бесплатный веб-фреймворк с открытым кодом, предназначенный для разработки приложений с использованием архитектурной модели MVC (англ. Model View Controller – модель-представление-контроллер). Laravel – php-фреймворк нового поколения, с его помощью можно создавать веб-приложения, используя простой и выразительный синтаксис. Он позволяет максимально упростить решение основных задач, таких, как аутентификация, маршрутизация, сессии и кэширование.

Ниже покажем, почему PHP-фреймворк Laravel приобрел популярность среди разработчиков.

Composer. До недавнего времени в PHP не существовало удобного способа для управления зависимостями проекта. В настоящее время существует две основные системы управления пакетами для PHP – Composer и PEAR. С нашей точки зрения, Composer делает процесс управления зависимостями максимально простым. Composer – это система упаковки для PHP, которая похожа на PEAR. Разработчикам не требуется фреймворк с массивной инфраструктурой, если для разрабатываемого проекта нужно несколько приложений. В этом случае достаточно установить Composer (менеджер зависимостей для PHP) и описать, от каких библиотек зависит проект, Composer автоматически установит требуемые библиотеки. Composer оперирует пакетами или библиотеками, они устанавливаются внутрь каждого проекта отдельно, а не глобально (это одно из основных отличий от PEAR).

Composer позволяет описать все библиотеки, от которых непосредственно зависит код разрабатываемого проекта. Запустив его на исполнение, Composer находит нужные версии требуемых пакетов (библиотек)

и их зависимостей для всего проекта, скачивает их и устанавливает в указанную директорию разрабатываемого проекта. По умолчанию пакеты скачиваются из официального репозитория packagist.org. Любой разработчик может добавить туда свой пакет, чтобы сделать его установку максимально доступной для всех. Пакеты можно скачивать не только с packagist.org, но и из любого git, mercurial или svn репозитория. При скачивании пакетов с github.com или bitbucket.org не требуется установленной системы контроля версий (git или hg), Composer работает через API этих сайтов. Все пакеты устанавливаются в текущую директорию (откуда была выполнена команда php composer install), это позволяет иметь несколько различных версий библиотек при работе над разными проектами параллельно. Команда php composer update обновляет все установленные (или установит заново случайно удаленные) пакеты до следующих версий. Специальный composer.lock файл позволяет зафиксировать комбинацию из стабильных версий всех используемых в проекте библиотек. После установки пакетов автоматически генерируется autoload.php, с помощью которого можно подключить установленные библиотеки в коде разрабатываемого проекта.

Composer распространяется в виде одного файла composer.phar (phar – это php-архив), в сущности это PHP-скрипт, принимающий несколько команд (install, update, ...), скачивающий и распаковывающий библиотеки.

Покажем, как используется Composer при разработке реального веб-приложения. Допустим, что разрабатываемый проект зависит от следующего перечня пакетов (библиотек):

- `laravel/framework` – фреймворк Laravel;
- `barryvdh/laravel-debugbar` – пакет, который интегрирует в фреймворк Laravel PHP Debug Bar (удобный инструмент, позволяющий контролировать и отлаживать код);
- `caouecs/laravel-lang` – локализация в Laravel, что позволяет поддерживать несколько языков интерфейса;
- `barryvdh/laravel-ide-helper` – пакет, который на основе кода разрабатываемого приложения генерирует файл-хелпер, содержащий сгенерированные статические классы фасадов, необходимые для автодополнения IDE;
- `fzaninotto/faker` – генерирует различные тестовые данные: строки, числа, тексты любых размеров (имена с учетом пола), номера телефонов, электронные адреса и т. д.

Файл composer.json представляет собой ряд инструкций для Composer'а, он должен быть в корне проекта, в нем указывается, от каких библиотек зависит проект. Первая и самая главная инструкция: «require».

Подключаем пакеты, которые были определены в примере выше:

```
«require»: {
   «laravel/framework»: «5.3.*»,
   «barryvdh/laravel-debugbar»: «v2.3.0»,
   «caouecs/laravel-lang»: «3.0.18»,
   «barryvdh/laravel-ide-helper»: «v2.2.1»,
   «fzaninotto/faker»: «1.6.x-dev»
}
```

После запуска команды php composer.phar install, Composer клонирует пакеты из репозитория и распаковывает их в директорию vendor, которую он сам создает в корне проекта. После распаковки в директории vendor будут расположены файл autoload.php и папки с клонированными пакетами. Если библиотеки не оформлены Composer-пакетами, то нужно дать дополнительную информацию об устанавливаемой библиотеке (например, описать правила автозагрузки классов и функций для autoload.php).

Маршрутизация. Большинство начинающих разработчиков PHP используют тривиальные системы маршрутов. Предположим, существует дерево каталогов: blog/admin/, соответствующее желаемому URI. Добавим файл в дерево каталогов следующим образом: blog/admin/index.php, теперь возможно получить к нему доступ, посетив localhost: 8080/blog/admin/index.php. Однако в дальнейшем понадобится больше гибкости и контроля над тем, какой маршрут запускается в приложении.

Laravel использует простой подход к маршрутизации и поддерживает несколько способов создания маршрутов. Все маршруты приложения описываются, как правило, в файле route/web.php. При открытии этого файла виден маршрут по умолчанию для пути (url) «/»:

```
Route::get('/', function(){ return …; });
```

Это простейшая форма определения маршрута с использованием замыкания. Как видно, для определения маршрута используется фасад[1] Route. Фасад Route содержит

[1] Фасады предоставляют «статический» интерфейс к классам, доступным в сервис-контейнере.

множество методов для различного определения маршрутов. Формально, любой HTTP-запрос состоит из:

- URL, запрашиваемый «путь» (например: «/», «/news» и т.д.);
- метод запроса (каждый запрос к веб-серверу делается с использованием одного из методов). В зависимости от него по одному и тому же URL могут выполняться разные действия. К примеру, запрос типа GET использующий URL «/news/1», служит для получения новости № 1, а запрос типа DELETE «/news/1» служит для удаления той же новости. URL один, а действия разные.

Laravel позволяет создавать маршруты, учитывая метод запроса. В примере выше создается маршрут для запроса типа GET, обрабатывающий маршрут

«/» (`Route::get(«/», …)`).

Кроме метода `get()` фасад Route имеет также методы для всех других методов запроса:

```
Route::post(«/news/add», ...); // метод
POST
Route::delete(«/news/1», ...); // метод
DELETE
// и т.д.
```

Тема маршрутизации в Laravel довольно обширна, выше были описаны основы, необходимые для создания простого приложения. Laravel также позволяет группировать маршруты, задавать им префиксы, обрабатывать поддомены и т. д.

Middleware. Middleware (HTTP-посредники) предоставляют удобный механизм для фильтрации HTTP-запросов разрабатываемого приложения. Предположим, в Laravel есть посредник для проверки аутентификации пользователя: если пользователь не аутентифицирован, посредник перенаправит его на экран входа в систему, в противном случае посредник позволит запросу пройти далее в приложение. Посредники нужны не только для авторизации, например, CORS-посредник можно использовать для добавления особых заголовков ко всем ответам приложения, посредник логов может зарегистрировать все входящие запросы.

В Laravel есть несколько стандартных посредников, включая посредников для обслуживания, аутентификации, CSRF-защиты и многие другие. Все они расположены в каталоге `app/Http/Middleware`.

Создается посредник с помощью команды:

```
php artisan make:middleware RolesMiddleware
```

Эта команда поместит новый класс `RolesMiddleware` в каталог `app/Http/Middleware`. В этом посреднике описана логика, предназначенная для пропуска только тех запросов, в которых роль (права) доступа пользователя будет (будут) admin, а во всех остальных случаях будем перенаправлять пользователей на страницу «home» URI:

```
<?php
namespace App\Http\Middleware;
use Closure;
class RolesMiddleware
{
  // Выполнение фильтра запроса.
  public function handle($request, Closure
  $next)
  {
    if ($request->user()->roles !=
    «admin») {
      return redirect(‘home’);
    }
    return $next($request);
  }
}
```

Если полученная роль пользователя не равна роли «admin», то посредник вернет клиенту переадресацию, в противном случае запрос будет передан далее в приложение. Чтобы передать запрос дальше в приложение, надо вызвать замыкание `$next` с параметром `$request`. Посредник можно представить как набор «уровней», которые должен пройти HTTP-запрос, прежде чем достичь созданного приложения. Каждый уровень может проверить запрос, возможно, и отклонить его.

Чтобы посредник запускался для каждого HTTP-запроса в приложении, необходимо добавить этот посредник в `$middleware` класса `app/Http/Kernel.php`.

Предусмотрена возможность назначить посредника для конкретных маршрутов, перед этим необходимо добавить сокращенный ключ посредника в класс `app/Http/Kernel.php`. По умолчанию `$routeMiddleware` этого класса содержит записи посредников Laravel. Чтобы создать собственный посредник, нужно добавить его к этому списку и присвоить ему ключ на свой выбор. Например:

```
protected $routeMiddleware = [
          // ... Системные middleware
      ‘roles’ => \App\Http\Middleware\
RolesMiddleware::class,
      ];
```

Когда посредник определен в HTTP-ядре, можно использовать ключ middleware в массиве параметров маршрута:

```
Route::get('/admin',
['middleware' => 'roles', function () {
                                    // ...
}]);
```

Для назначения маршруту нескольких middleware необходимо использовать массив:

```
Route::get('/admin', ['middleware' =>
['auth', 'roles'],
function () {
  // ...
}]);
```

Вместо массива можно использовать сцепку метода middleware() с определением маршрута:

```
Route::get('/admin', function () {
  // ...
})->middleware(['auth', 'roles']);
```

Миграции. Миграции можно рассматривать как системы контроля версий для базы данных (БД) проекта. Миграции можно применять и отменять. При командной разработке с помощью этого механизма база данных проекта легко приводится в актуальное состояние на любом компьютере. Не надо хранить множество SQL-файлов для отслеживания изменений БД, миграции выполняют эту задачу.

Приведем основные операции (методы), которые должна реализовывать каждая миграция:

- up() – выполняет изменения текущей миграции;
- down() – отменяет сделанные изменения миграции.

MVC. PHP-фреймворк Laravel реализует шаблон проектирования MVC, который предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

Этот шаблон разделяет работу веб-приложения на три отдельные функциональные роли: модель данных (model), пользовательский интерфейс (view) и управляющую логику (controller). Таким образом, изменения, вносимые в один из компонентов, оказывают минимально возможное воздействие на другие компоненты. В данном паттерне модель не зависит от представления или управляющей логики, что делает возможным проектирование модели как независимого компонента и, например, создавать несколько представлений для одной модели.

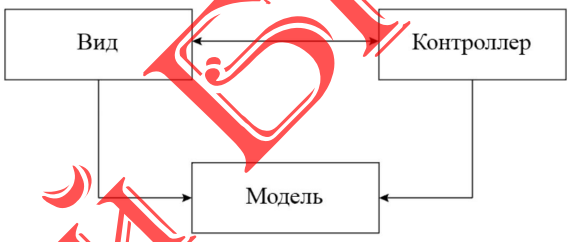

Рисунок 1 – Архитектура MVC

Основная цель применения этого паттерна заключается в разделении приложения на три основных компонента, каждый из которых отвечает за различные задачи.

За счет такого разделения повышается возможность повторного использования программного кода. Наиболее полезно применение данной концепции, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения. К одной модели можно присоединить несколько видов, не затрагивая реализацию модели. Например, некоторые данные могут быть одновременно представлены в виде электронной таблицы, гистограммы и круговой диаграммы. Можно изменить реакции на действия пользователя (нажатие мышью на кнопке, ввод данных), для этого достаточно использовать другой контроллер.

Рисунок 2 – Концепция MVC

Ряд разработчиков специализируются только в одной из областей: или разрабатывают графический интерфейс или разрабатывают бизнес-логику. Поэтому возможно добиться, что программисты, занимающиеся разработкой бизнес-логики (модели), вообще не будут осведомлены о том, какое представление будет использоваться.

Очевидное преимущество, получаемое от использования концепции MVC, это четкое разделение логики представления (интерфейса пользователя) и логики приложения, а также упрощение разработки больших приложений. Код получается более структурированным, тем самым облегчается поддержка, тестирование и повторное использование решений.

В настоящее время большинство современных фреймворков для веб-программирования реализуют концепцию MVC. К наиболее удачным примерам применения этого паттерна для языка PHP можно отнести Laravel.

Заключение. Используя лучшие практики программирования на PHP-фреймворке Laravel, была разработана компьютерная среда диагностики и контроля знаний. Приложение представляет собой платформу для создания, редактирования и прохождения тестов.

Приложение состоит из трех модулей: модуль преподавателя, модуль студента и модуль настроек, так же реализован межпользовательский чат.

Пользователю при первом посещении ресурса предлагается пройти регистрацию для создания личного кабинета, одним из пунктов регистрации является выбор «роли» пользователя (преподаватель/учащийся), от этого выбора зависят права доступа к отдельным возможностям приложения.

Пройдя регистрацию, пользователь перенаправляется в свой личный кабинет. Пользователь с правом доступа «студент» открыт доступ в «модуль студента» и «модуль настроек», а пользователю с правом доступа «преподаватель» кроме вышеперечисленных модулей открыт доступ и в «модуль преподавателя», где он может создавать, редактировать и удалять созданные им тесты.

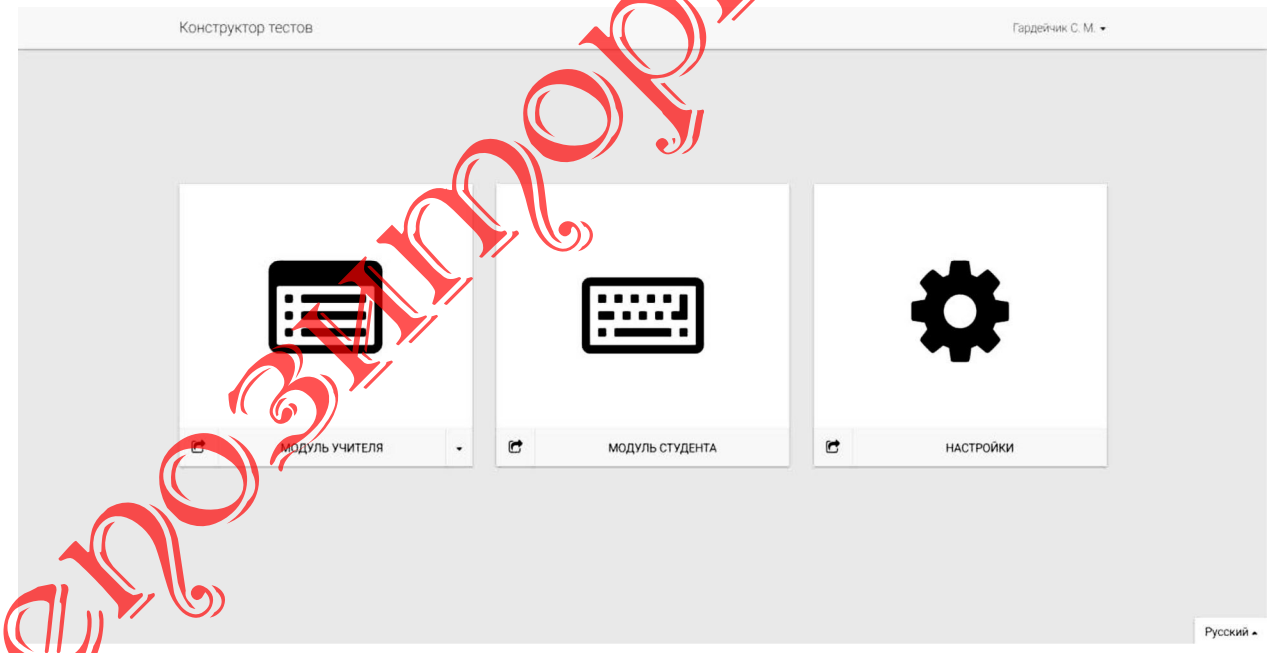

Рисунок 3 – «Главная» страница веб-приложения

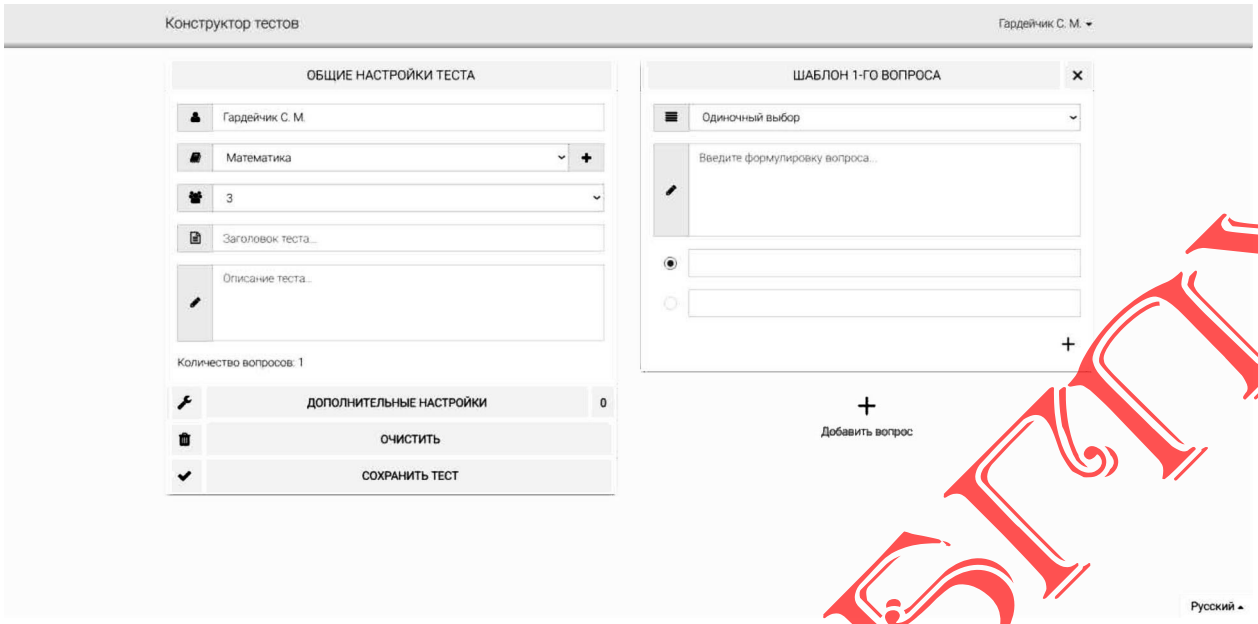

Рисунок 4 – Страница «Модуль преподавателя» веб-приложения

Рисунок 5 – Страница «Модуль студента» веб-приложения

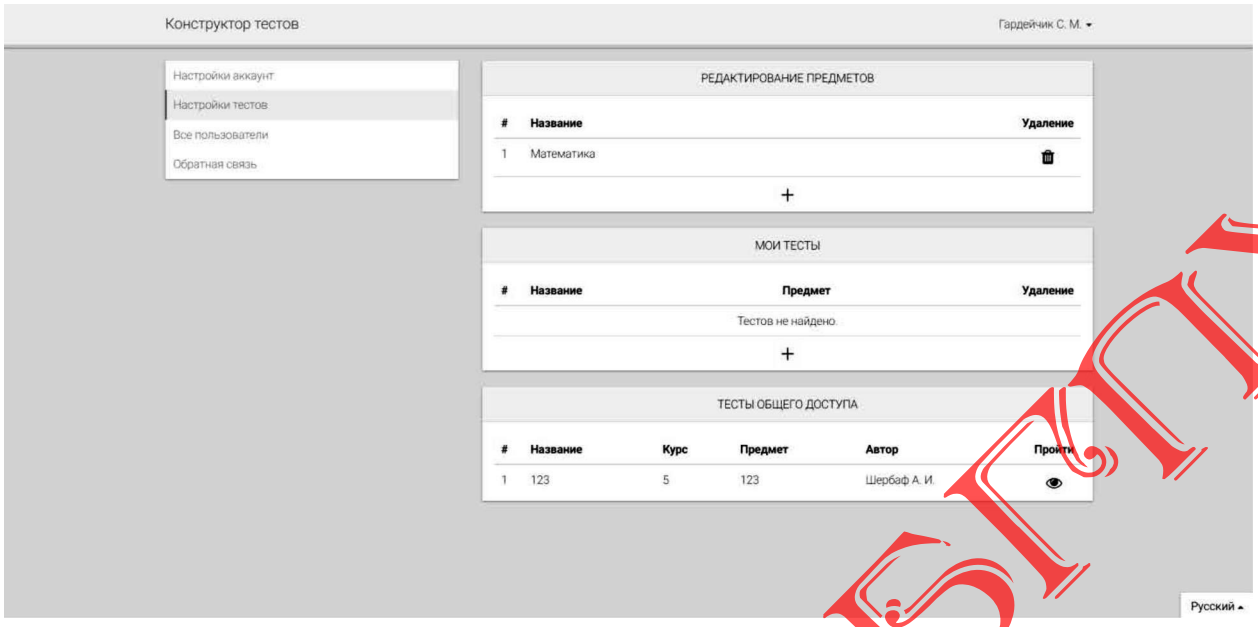

Рисунок 6 – Страница «Модуль настройки»: настройка тестов

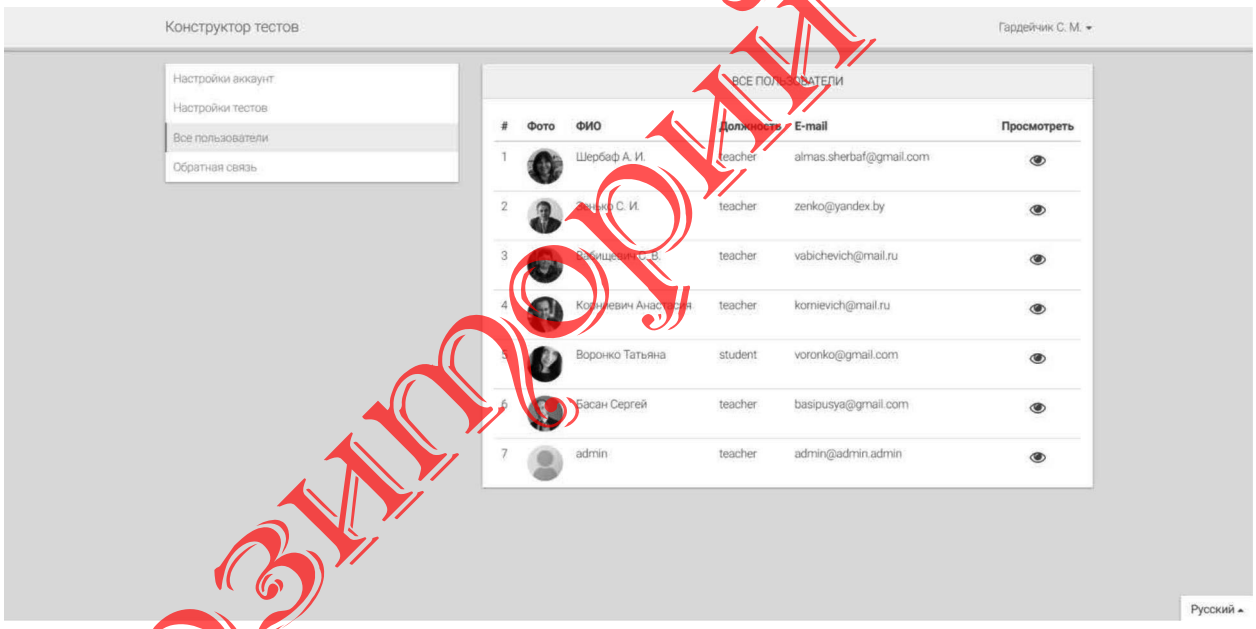

Рисунок 7 – Страница «Модуль настройки»: все зарегистрированные пользователи

Перечислим основные преимущества разработанного приложения.

Реализация MVC: программная разработка полностью реализует парадигму MVC, что гарантирует простоту масштабируемости и обновления.

Отсутствие инсталляции программного обеспечения: веб-приложение в отличие от десктопного (реализующего Windows Forms интерфейс) не требует инсталляции или загрузки программных модулей на ПК.

Обновления программного обеспечения: настольные приложения необходимо периодически обновлять, в большинстве случаев это нужно делать вручную, а веб-приложения обновляются поставщиком услуг.

Использование ресурсов: сложные приложения, требуют большого объема памяти и высокого быстродействия. В веб-приложениях большая часть тяжелой обработки выполняется поставщиком услуг, что мини-

мизирует требования по настройке системы и частично снижает нагрузку на систему.

Независимость от платформы: настольные приложения нужно разрабатывать для различных платформ, таких как Windows, Linux, Mac. Поскольку большинство облачных приложений предназначено для использования в веб-браузере, они работают на разных платформах, включая мобильные девайсы.

Хранение данных: в настольных приложениях данные хранятся в компьютере, тогда как в веб-приложениях они размещаются в выделенном пользователю пространстве сервера, что обеспечивает постоянный доступ к данным из любой точки мира.

Литература

1. *Stauffer, M.* Laravel: Up and running: A Framework for Building Modern PHP Apps / M. Stauffer. – USA : O'Reilly, 2016. – 454 p.
2. *McCool, S.* Laravel Starter / S. McCool. – Birmingham–Mumbai : PACKT Publishing, 2012. – 64 p.

References

1. *Stauffer, M.* Laravel: Up and running: A Framework for Building Modern PHP Apps / M. Stauffer. – USA: O'Reilly, 2016. – 454 p.
2. *McCool, S.* Laravel Starter / S. McCool. – Birmingham–Mumbai: PACKT Publishing, 2012. – 64 p.