Весці БДПУ. Серыя 3. 2018. № 3. С. 85–91.

УДК 004.738.5

SPA-АРХИТЕКТУРА МУЛЬТИФУНКЦИОНАЛЬНОГО ВЕБ-ПРИЛОЖЕНИЯ

С. М. Гардейчик,
магистрант БГПУ;
А. И. Шербаф,
кандидат физико-математических наук, доцент кафедры информатики и методики преподавания информатики БГПУ

Поступила в редакцию 28.06.18.

UDC 004.738.5

SPA-ARCHITECTURE OF A MULTIFUNCTIONAL WEB-APPLICATION

S. Gardeychik,
Master Steudent, BSPU;
A. Sherbaf,
PhD in Physics and Mathematics, Associate Professor of the Department of Informatics and Methods of Teaching Informatics, BSPU

Received on 28.06.18.

В данной статье обсуждаются клиент-серверные технологии для проектирования масштабируемых и отказоустойчивых веб-систем, а также технологии, упрощающие разработку и объединение различных компонентов большого программного проекта. Одним из трендов веб-разработки в настоящее время является высокопроизводительная SPA-технология, поэтому отдельное внимание уделено SPA-архитектуре. Описан выбор технологии по реализации SPA-архитектуры и подходящих для решения поставленной задачи библиотек и фреймворков, а также преимущества и недостатки SPA-технологии. Приведены результаты использования данной технологии для разработки мультифункционального приложения для управления и статистического анализа большого объема данных для определения показателей здоровья человека.

Ключевые слова: SPA-архитектура, PHP-фреймворк Laravel, JSON, RESTful, API, веб-приложение, JS-фреймворк Vue.JS, веб-сервис.

This article discusses the client-server technologies for designing scalable and fault-tolerant web-systems, as well as technologies that simplify the development and integration of various components of a large software project. Nowadays one of the trends of web development is a high-performance SPA-technology, so special attention is paid to SPA-architecture. The choice of technology for the implementation of SPA architecture, the libraries and frameworks suitable for the solution of the given task, and the advantages and disadvantages of SPA-technology are described. The results of using this technology in the development of a multifunctional application for the management and statistical analysis of a large amount of data for the determination of human health indicators are given.

Keywords: SPA-architecture, PHP-framework Laravel, JSON, RESTful, API, web-application, JS-framework Vue.JS, web-service.

Введение. Веб-технологии развиваются с большой скоростью, одни технологии приходят на смену другим. Одним из трендов веб-разработки в настоящее время является высокопроизводительная SPA-технология. Цель настоящей статьи – познакомить читателей с этим современным подходом к разработке веб-приложений, описать особенности и преимущества применения данной технологии. Ниже будут приведены результаты использования данной технологии для разработки мультифункционального приложения для управления и статистического анализа большого объема данных для определения показателей здоровья (далее – E-HealthDiary).

По определению Википедии, «одностраничное приложение (SPA, или single page application) представляет собой веб-сайт или приложение, размещенное на одной веб-странице». Таким образом, SPA – это веб-приложение, размещенное на одной веб-странице, которая для работы приложения загружает весь необходимый код вместе с загрузкой самой страницы.

Были попытки использовать такие технологии, как, например, IFrames, Java-апплеты,

Adobe Flash и Microsoft Silverlight, для сведения десктопного приложения в межплатформенную среду веб-браузера. Одностраничное веб-приложение (SPA) решает эту задачу без расширения (плагинов) для браузера или изучения нового языка программирования, SPA может быть реализована с использованием (как минимум) только JavaScript, HTML и CSS.

Для SPA характерен более гибкий и удобный пользовательский интерфейс, сходный с интерфейсом десктопного приложения. SPA использует современные браузеры и язык HTML 5 для переноса интерфейса и логики приложений с веб-серверов на браузер. SPA обеспечивает более естественный и контролируемый опыт взаимодействия, скрывая сложные переходы (запросы, отклики и пр.). SPA отображает только запрашиваемый пользователем контент, загружая HTML-данные частями. Средой реализации веб-приложений становится не операционная система, а браузер, управляющий памятью и предоставляющий функционал для работы с системными функциями, аппаратным окружением.

Существует множество подходов к конструированию одностраничных приложений. Чтобы найти правильное решение для будущего SPA-проекта, необходимо выбрать технологию по реализации SPA-архитектуры, подходящие для решения поставленной задачи библиотеки и фреймворки, а также четко понимать SPA-технологию, ее преимущества и недостатки.

Использование SPA-архитектуры имеет следующие преимущества: в случае, когда визуальный интерфейс не содержит бизнес-логики, каждый ресурс может работать параллельно, не внося конфликтов в проект; разделение представления и логики помогает разработчикам осуществлять более «чистое» модульное тестирование, так как им приходится работать только с невизуальной стороной функции; отдельные уровни (представление, данные, бизнес-логика и т. п.) помогают в обслуживании и развертывании веб-приложения; изолированный код можно легко изменить, не затрагивая другие части приложения.

Для грамотной реализации конкретного SPA-проекта необходимо владеть широким спектром технологий, для этого проанализируем существующие подходы к разработке SPA.

Основная часть. Остановимся подробнее на описании и сравнительном анализе MV*-фреймворков, которые представляют собой семейство JS-фреймворков и обеспечивают разделение кодовой базы. Здесь M обозначает Model (модель), а V – View (представление / вид). Кратко охарактеризуем эти понятия. Модель обычно содержит данные, бизнес-логику и логику валидации (проверку данных на достоверность), реагирует на команды контроллера, изменяя свое состояние. Представление является визуальным отображением данных, предоставленных моделью. Это может быть простая структура, взаимодействующая с другими частями фреймворка для обновлений и ответов на запросы пользователей; может содержать небольшое количество логики, в зависимости от реализации MV*-фреймворка.

Традиционные шаблоны пользовательского интерфейса включают в себя третий компонент, который помогает управлять отношениями между моделью и представлением. Большинство современных шаблонов проектирования пользовательских интерфейсов, основанных на MVC / MVVM, содержат два главных компонента – модель и представление. Третий компонент, который изменяется как по имени, так и по функционалу, обозначен звездочкой (*).

Приведем описание шаблонов UI проектирования, которые оказали наибольшее влияние на разработку веб-приложений (на стороне клиента/браузера): Model-View-Controller (MVC), Model-View-Presenter (MVP) и Model-View-ViewModel (MVVM).

Model-View-Controller (MVC). Шаблон MVC включает модель, представление и контроллер (рисунок 1).

Контроллер является точкой входа в веб-приложение, принимает сигналы от элементов управления в пользовательском интерфейсе (UI). Содержит логику, которая обрабатывает запросы пользователя, генерирует и отправляет команды модели для обновления своего состояния на основе полученных инструкций.

Взаимодействие с контроллером инициирует событие, приводящее к обновлению представления. Пользовательский интерфейс (UI) обновляется на основании данных, предоставленных моделью.

Model-View-Presenter (MVP). Идея этой концепции заключается в том, чтобы модель сделать более независимой от двух других компонентов MVC. В MVP объект, подобный

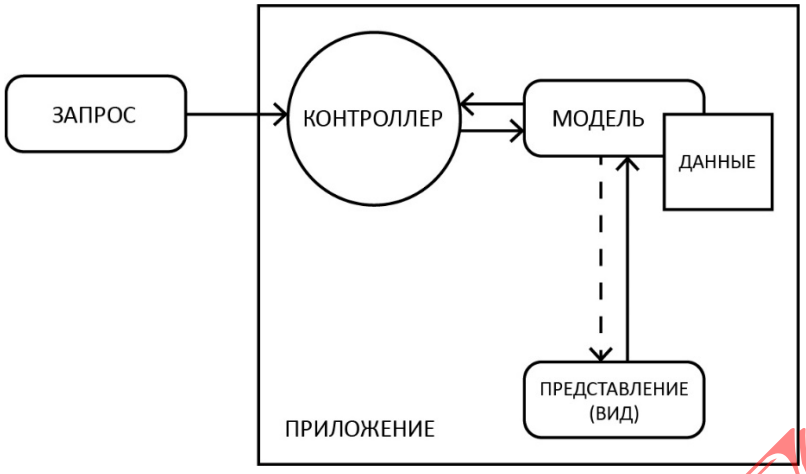

Рисунок 1 – Шаблон проектирования MVC использовался в течение многих лет в разработке графических пользовательских интерфейсов

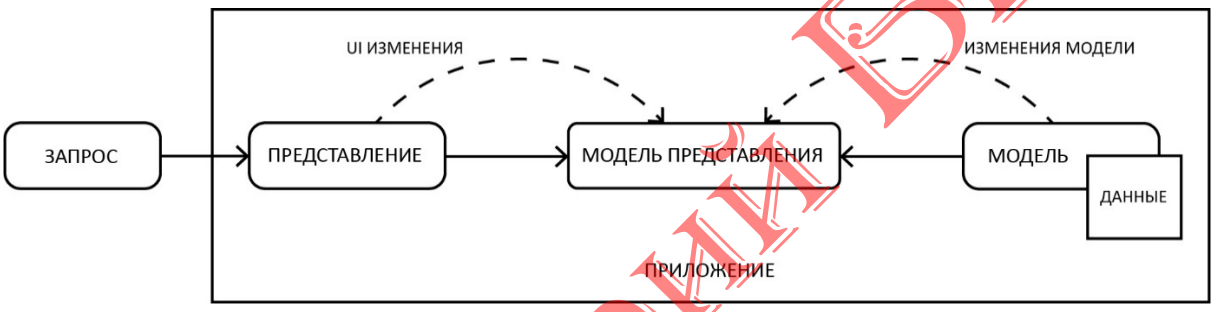

Рисунок 2 – MVP – это вариация MVC. С помощью этого шаблона представление (View) является точкой входа, но его логика находится в представителе (Presenter)

контроллеру, и представление будут совместно определять пользовательский интерфейс (UI). За моделью остается логика управления данными. В MVP нет контроллера, действующего в качестве посредника. Каждый вид поддерживается компонентом, называемым Presenter (рисунок 2).

Presenter содержит логику представления. Вид реагирует на действия пользователя, делегируя ответственность Presenter. Presenter имеет прямой доступ к модели для любых необходимых изменений и передает данные обратно в представление. Таким образом, он действует как «посредник» между моделью и видом

Model-Views-Viewmodel (MVVM). Данная парадигма была создана для стандартизации процесса разработки пользовательских интерфейсов и на ранних этапах использовалась в Microsoft® Windows Presentation Foundation (WPF). Это еще один шаблон проектирования, который возник для организации управляемого кода, связанного с пользовательским интерфейсом и сохраняющего различные компоненты отдельно. Как и в MVP, представление является точкой входа в приложение, компонент, который находится между моделью и видом (рисунок 3), называется ViewModel.

ViewModel является посредником между моделью и представлением, содержит свойства данных и определяет логику по обновлению данных в модели. Если в модели изменяются значения этих свойств, автоматически происходит обновление отображаемых данных в представлении, хотя напрямую модель и представление не связаны.

Vue. JS. На начало текущего года поиск по GitHub* выдает более чем 5.2 млн проектов на JavaScript, подавляющее большинство которых разработано на базе JS-фреймворков, таких, как Vue.js, React.js, Angular.js, Ember.js и др. По версии GitHub, за последнее время JS-фреймворк Vue.js приобретает стремительную популярность среди мирового IT-сообщества.

Прогрессивный JS-фреймворк Vue.js определен как ViewModel уровень шаблона MVVM, соединяющий модель и представление в двустороннее связывание данных.

* Крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

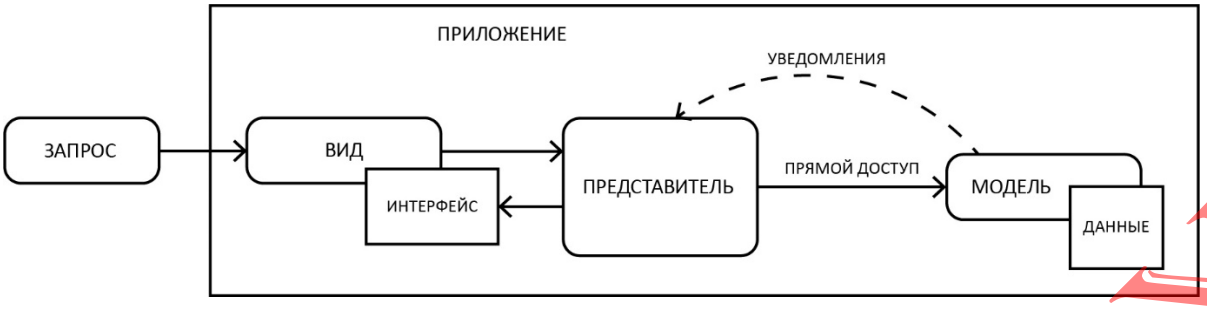

Рисунок 3 – В MVVM модель представления (ViewModel) осведомлена об изменениях как в модели, так и в представлении и поддерживает их синхронизацию

В отличие от большинства фреймворков Vue пригоден для постепенного внедрения. Его ядро решает задачи уровня представления (view), что упрощает интеграцию с другими библиотеками и существующими проектами. С другой стороны, Vue полностью подходит и для создания сложных одностраничных приложений (SPA), если использовать его совместно с современными инструментами и дополнительными библиотеками.

Декларативный рендеринг. Vue.js реализует систему, декларативного отображения данных в DOM, используя простой шаблонный синтаксис:

```
HTML-код: <div id="app">{{ message }}</div>
JS-код:
var app = new Vue({
  el: '#app',
  data: {
    message: 'Привет, Vue!'
  }
})
```

Разбиение приложения на компоненты. Важной концепцией Vue являются компоненты, позволяющие собирать большие приложения из меньших частей. Компоненты позволяют инкапсулировать код, затем использовать его многократно в различных частях приложения. Практически каждый пользовательский интерфейс может быть представлен как дерево компонентов (рисунок 4).

Во Vue компонент – это, по сути, экземпляр Vue с предустановленными опциями. Создать новый компонент во Vue просто:

```
// Определение нового компонента под названием todo-item
Vue.component('todo-item', {
  template: '<li>Это одна задача в списке</li>'
})
Теперь его можно использовать в шаблоне другого компонента:
<ul>
<!-- Создаём экземпляр компонента todo-item -->
  <todo-item></todo-item>
</ul>
```

Сейчас Vue используется уже по всему миру, например этот фреймворк вошел в Laravel и PageKit, а это значит, что в скором времени произойдет значительный рост числа пользователей Vue.

Передача состояния представления (REST). REST – архитектурный стиль для проектирования распределенных систем, представляет собой согласованный набор ограничений, таких как отсутствие состояния, наличие отношений клиент/сервер и единый интерфейс. REST не имеет строгого отношения к HTTP, но он чаще всего связан с ним. Для веб-систем, построенных с учетом REST (то есть не нарушающих накладываемых им ограничений), применяют

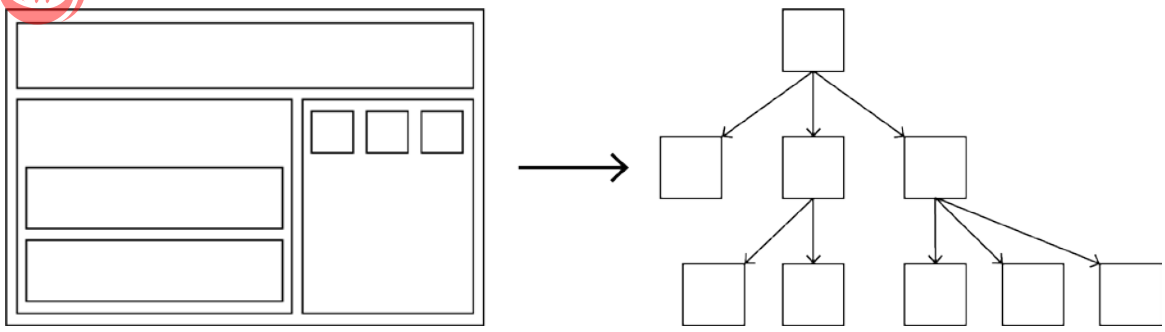

Рисунок 4 – Дерево компонентов

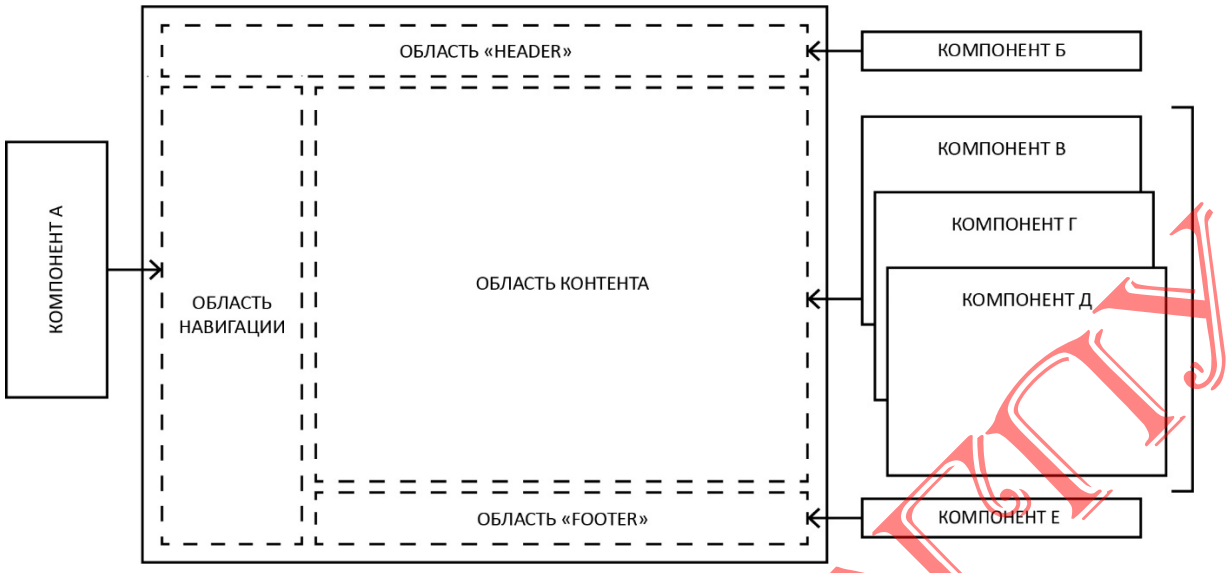

Рисунок 5 – Пример компонентной модели веб-системы E-HealthDiary

термин «RESTful». Несмотря на то что REST не является стандартом (определен как архитектурный стиль), большинство RESTful-реализаций используют стандарты, такие, как HTTP, URL, JSON и XML.

Существует шесть ограничений для построения распределенных REST-приложений по Филдингу, которые преобразуют любой веб-сервис в RESTful API. Действуя в рамках этих ограничений, система приобретает такие свойства, как производительность, масштабируемость, простота, переносимость и надежность. Архитектурными REST-ограничениями являются: реализация единообразного интерфейса; модель клиент-сервер; отсутствие состояния; кэширование; реализация многослойной системы; предоставление кода по запросу (необязательно).

Заключение. *Практическая реализация SPA-архитектуры.* Была спроектирована мультифункциональная, веб-система расчета и статистического анализа показателей физического развития студента «E-HealthDiary». Для оптимального удовлетворения требований предоставленного технического задания (ТЗ), кафедры физического воспитания БГПУ разработана веб-система с использованием SPA-архитектуры и сопутствующих технологий (JS-фреймворк Vue.js, PHP-фреймворк Laravel и др.). В ней фиксируются показатели физического развития, функционального состояния и физической подготовленности студента. Анализ динамики индивидуальных показателей физического состояния позволяет выбрать наиболее эффективные средства физического воспитания, дозировать и корректировать физическую нагрузку в процессе учебных и самостоятельных занятий физическими упражнениями без ущерба для здоровья, правильно организовывать двигательную активность студента. В дневнике здоровья отмечаются задания преподавателя для самостоятельных занятий, а также участие студентов в физкультурно-оздоровительных мероприятиях университета. Дневник здоровья служит документом, характеризующим уровень физического состояния будущего специалиста.

Архитектура клиентской части E-HealthDiary. SPA-архитектура является оптимальным решением для удовлетворения требований по разработке масштабируемой и многофункциональной асинхронной веб-системы. Для достижения гибкой многофункциональности клиентского приложения был использован прогрессивный JS-фреймворк Vue.js, изначально согласующийся с принципами SPA и корректно работающий в браузерах, перечисленных в требованиях ТЗ.

Важной концепцией Vue.js являются компоненты, позволяющие собирать большие приложения из меньших частей. Компоненты позволяют инкапсулировать код, затем использовать его многократно в различных частях приложения. Практически каждый пользовательский интерфейс может быть представлен в виде набора слабосвязанных компонентов (рисунок 5).

Проектирование маршрутизации между компонентами (рисунок 6), которые приводят к сложным конфигурациям между базовыми областями и компонентами и на определенном этапе разработки ими становится сложно управлять. В сложных макетах следует

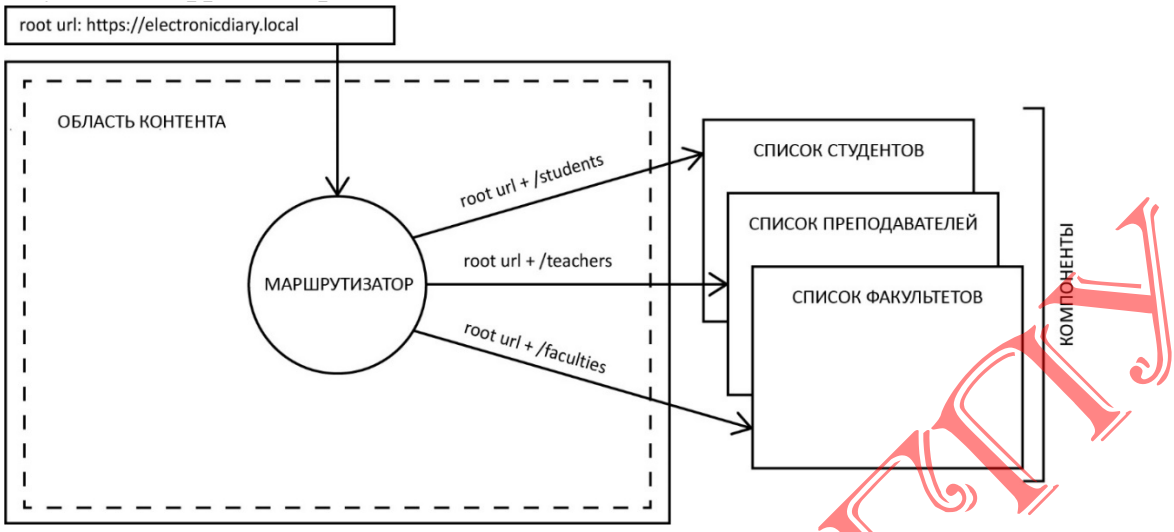

Рисунок 6 – Пример маршрутов с несколькими компонентами

управлять состоянием приложения, используя MV*-фреймворки.

Архитектура серверной части E-HealthDiary. Архитектура серверной части построена на основе PHP-фреймворка Laravel. Для структурирования серверной части ВП были взяты принципы архитектурного типового решения MVC. Рассмотрим основные компонентные представления серверной части ВП, которые были разработаны.

Маршрутизатор (Route) – в соответствии с маршрутом делегирует дальнейшую обработку запроса управляющим объектам нижележащего уровня: контроллеру или API.

Контроллер (Controller) – отвечает за функциональную обработку запроса. Содержит методы для реализации функциональных требований к ВП. Использует модель для взаимодействия с источником данных предметной области.

Модель (Model) – отвечает за бизнес-логику, непосредственно связанную с предметной областью. Предоставляет интерфейс для работы с сущностями БД. Инкапсулирует обработку данных соответствующей ей сущности.

RESTful API – стандартизированный интерфейс для работы с данными предметной области. Инкапсуляция бизнес-логики за определенным маршрутом. Возвращает все данные в формате JSON. Запрос к API определяется параметром в строке URL. В соответствии с RESTful API, на примере работы со студентами, семантика запросов была построена следующим образом (таблица). Аналогично построены интерфейсы запросов и для других сущностей.

Таблица – Семантика запросов в соответствии с REST

Ресурс/ Метод	GET	POST	PUT	DELETE
/student	Вернуть список всех студентов	Добавить студента	–	–
/student/:id	Получить информацию о студенте с указанным идентификатором	–	Обновить информацию о студенте с указанным идентификатором	Удалить информацию о студенте с указанным идентификатором

Модель взаимосвязи технологий веб-приложения. Покажем взаимосвязь технологий и решений клиентской и серверной части веб-приложения E-HealthDiary (рисунок 7).

Преимущества E-HealthDiary. Перечислим технические преимущества разработанного проекта.

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

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

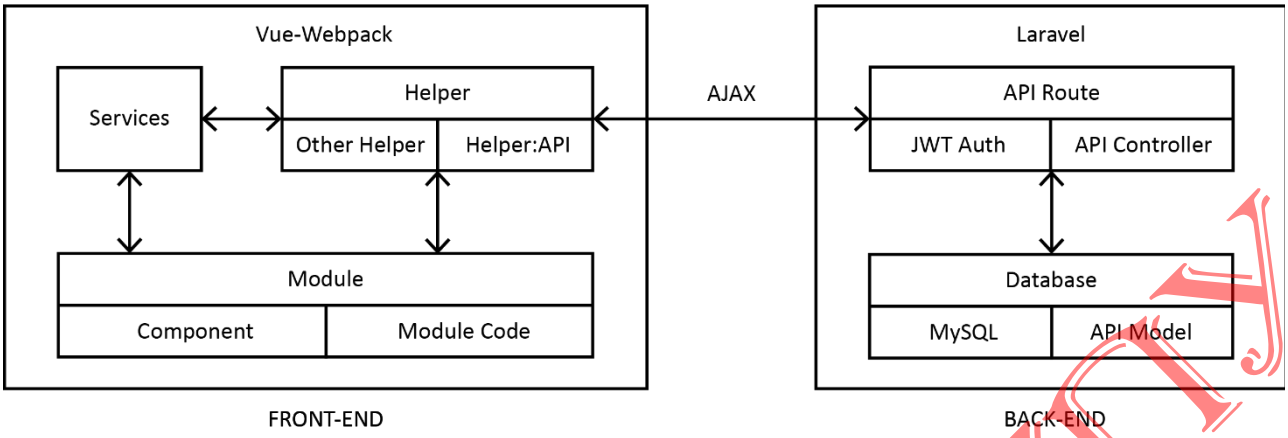

Рисунок 7 – Связь клиентской и серверной части веб-приложения

Использование ресурсов: сложные приложения, такие, как «Электронный дневник здоровья», требуют большого объема памяти и высокого быстродействия. В веб-приложениях большая часть тяжелой обработки выполняется поставщиком услуг, что минимизирует требования по настройке системы и частично снижает нагрузку на систему.

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

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.

Разработанная веб-система находится в стадии активного тестирования, что включает в себя:

1. Функциональное тестирование (ввод/вывод большого объема данных, отказоустойчивость веб-системы и т. д.).
2. Юзабилити-тестирование.
3. Тестирование пользовательского интерфейса.
4. Тестирование совместимости, выработка системных требований (совместимость с веб-браузерами и мобильными устройствами).
5. Тестирование производительности.
6. Тестирование безопасности.

В качестве тестирующего персонала выступили преподаватели кафедры физического воспитания БГПУ. Ознакомиться с данной разработкой можно по адресу h97966n.beget.tech.

Литература

1. *Dockins, K.* Design Patterns in PHP and Laravel / K. Dockins. – New York : Apress, 2017. – 1st ed. – 238 p.
2. PHP-фреймворк Laravel: документация [Электронный ресурс]. – Режим доступа: https://laravel.ru. – Дата доступа: 24.03.2018.
3. *Scott, E.* SPA Design and Architecture: Understanding Single Page Web Applications / E. Scott. – New York : Manning Publications Co., 2015 – 314 p.
4. *Гардейчик, С. М.* Программная платформа Laravel для создания веб-ориентированных приложений и сервисов / С. М. Гардейчик, А. И. Шербаф // Вести БГПУ. Серия 3. Информатика. – 2017. – № 3. – С. 82–90.

References

1. *Dockins, K.* Design Patterns in PHP and Laravel / K. Dockins. – New York : Apress, 2017. – 1st ed. – 238 p.
2. PHP-freymvork Laravel: dokumentatsiya [Elektronnyy resurs]. – Rezhim dostupa: https://laravel.ru. – Data dostupa: 24.03.2018.
3. *Scott, E.* SPA Design and Architecture: Understanding Single Page Web Applications / E. Scott. – New York : Manning Publications Co., 2015 – 314 p.
4. *Gardeychik, S. M.* Programmnaya platform Laravel dlya sozdaniya veb-oriyentirovannykh prilozheniy i servisov / S. M. Gardeychik, A. I. Sherbaf // Vesti BGPU. Seriya 3. Informatika. – 2017. – № 3. – S. 82–90.