Я. Г. Карчевская

WEB-ПРИЛОЖЕНИЕ ТУРИСТИЧЕСКОГО АГЕНСТВА

Рассматривается подход к созданию web-приложения, содержащего клиентскую и серверную части. Приводятся наиболее популярные технологии решения проблем реализации приложения.

Введение

Для создания клиент-серверного приложения требуется понимание целого набора инструментов, языков и систем. Кроме знания HTML/CSS и JavaScript, необходимо освоить языки и инструменты для реализации серверной части, которая осуществляет взаимодействие с базой данных. Сложность создания приложения усиливается постоянным развитием и изменением технологий.

В статье представлен набор средств, которые активно используются разработчиками и являются одними из самых актуальных на данный момент.

I. Клиентская часть

Для реализации клиентской части можно использовать язык JavaScript, он гибок и прост в использовании. Но по мере усложнения JS-кода возрастает преимущество использования языка TypeScript. TypeScript – это язык-расширение JavaScript, которое добавляет возможность статической типизации. Когда TS-код компилируется в JS, все определения типов опускаются. TypeScript позволяет проще писать комплексные решения, которые легче развивать и тестировать в дальнейшем. Однако важно учитывать тот факт, что разработка на TypeScript займет больше времени.

Существуют два основных подхода к разработке клиента: можно создавать как одностраничные веб-приложения(SPA), так и многостраничные(MPA). SPA-приложения работают в рамках браузера и не требуют перезагрузки страницы или загрузки дополнительных страниц во время использования. Самыми популярными примерами SPA-приложений являются GitHub, Gmail, Google Maps и Facebook. В SPA есть всего одна страница с разметкой, стилями и скриптами. При переходе на другие страницы, приложение подгружает соответствующие модули и контент. Кроме вышеперечисленного, SPA-приложения имеют, как правило, уже готовые удобные решения для организации REST API, управления отображением HTML-кода и множество других полезных при разработке средств. Так, на страничке приложения турагенства отображается список туров, предлагаемых пользователю сайта в виде блоков с информацией о туре. Блок HTML-кода создается только один, а данные в нем зависят от переданных значений. С помощью специальных инструкций, реализующих циклы, на странице отображается нужное количество блоков с различными данными, соответствующими информации о каждом туре.

При таком архитектурном подходе чаще всего используются технологии React, Angular и Vue. Они являются «тремя китамис в разработке клиентской части приложения.

Рис. 1 – Популярные JavaScript-фреймворки

У каждого фреймворка есть свои плюсы и минусы, а это означает, что при разработке продукта нужно сделать правильный выбор для каждого отдельного случая.

II. Серверная часть

Сервер организует взаимодействие как с клиентской частью, так и базой данных. Здесь создаются API для фронтенда. Для реализации серверной части можно использовать языки Ruby и популярные фреймворки Ruby on Rails и Sinatra, Python и фреймворки Django и Flask или же языки Java, PHP. Однако, с опытом работы с JavaScript наилучшим решением станет Node.js. Node.js — это просто окружение JS, то есть, при знании JavaScript, нет необходимости в изучении нового языка программирования. Функциональность Node.js расширяется с помощью пакетов. Наиболее часто используется фреймворк Express, все альтернативы в десятки раз менее популярны. Он существенно упрощает разработку веб-приложений на базе Node.js.

III. Взамодействие клиента и сервера

Итак, определившись с технологиями для создания приложений клиента и сервера, возникает вопрос их взаимодействия. REST API – это «мостикс между двумя этими сторонами. REST – это набор принципов и ограничений взаимодействия клиента и сервера в сети интернет, использующий существующие стандарты (HTTP протокол, стандарт построения URL, форматы данных JSON и XML) в ходе взаимодействия. Так, клиентской части предоставляется некий URL, определенный на серверной части, по которому

клиент получает необходимую информацию из базы данных.

Например, при загрузке страницы туров, с клиента, используя URL, отправляется запрос на сервер для предоставления информации о каждом туре.

IV. База данных

При выборе базы данных предстоит принять важное решение: остановиться на реляционной (SQL) или нереляционной (NoSQL) структуре БД. MySQL – отличный выбор для любого приложения, которому будет удобно пользоваться ее заранее определенной структурой и готовыми схемами. MongoDB, напротив, подойдет для бизнесов с быстрым ростом или для баз данных, в которых не используются определенные схемы. Для работы с MongoDB используется специальная ODM-библиотека (Object Data Modelling) Mongoose, которая позволяет сопоставлять объекты классов и документы коллекций из базы данных.

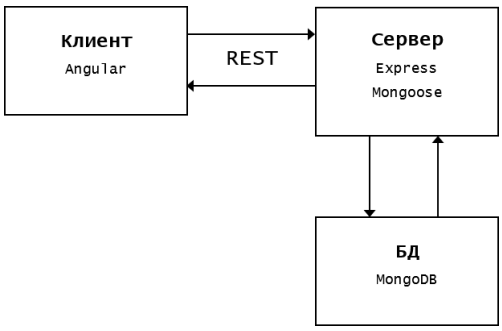

Рис. 2 – Архитектура клиент-серверного приложения

V. Аутентификация и авторизация

Когда в системе создаётся новый пользователь, его пароль необходимо хэшировать и сохранять в базе данных. Пароль в базе сохраняют вместе с адресом электронной почты и другими сведениями о пользователе.

Когда же клиент входит в свою учетную запись, он отправляет серверу аутентификационный ключ. Обычно это – пара логин/пароль. Если пользователь существует в базе данных – сервер хэширует отправленный ему пароль и сравнивает то, что получилось, с хэшем пароля, сохранённым в базе данных.

Использование JSON Web Token (JWT) – JSON объекта, определенного в открытом стандарте RFC 7519 – считается одним из безопасных способов передачи информации между двумя участниками. Сервер будет обеспечивать пользователя токеном, с помощью которого он позднее сможет взаимодействовать с приложением. Когда пользователь сделает запрос к API приложения, к нему добавится полученный ранее JWT для проверки того, что данные были действительно отправлены авторизованным источником.

Так, пользователь, имеющий учетную запись, сможет аутентифицироваться при каждом запросе, например, для получения данных о своих заказах.

VI. Выводы

Архитектура клиент-серверного приложения предполагает разделение системы на клиентов и на серверов. Разделение интерфейсов означает, что, например, клиенты не связаны с хранением данных, которое остается внутри каждого сервера, так что мобильность кода клиента улучшается. Серверы не связаны с интерфейсом пользователя или состоянием, так что серверы могут быть проще и масштабируемы. Серверы и клиенты могут быть заменяемы и разрабатываться независимо, пока интерфейс не изменяется.

Существует множество инструментов для реализации данной архитектуры приложения. Использование технологии SPA для создание клиентской части и Node.js для серверной является актуальным подходом для простой и удобной разработки клиент-серверных приложений.

1. Express: fast, unopinionated, minimalist web framework for Node.js [Electronic resource] / StrongLoop, IBM, and other expressjs.com contributors, 2017. – Mode of access: https://expressjs.com/. – Date of access: 14.04.2020.
2. Node.js Docs [Electronic resource] / Joyent, Inc., 2020. – Mode of access: https://nodejs.org/en/docs/. – Date of access: 14.04.2020.
3. single page application (SPA) и multi page application (MPA) [Электронный ресурс] / MIREHEAD: fintech и блокчейн разработка, 2019. – Режим доступа: https://merehead.com/ru/. – Дата доступа: 14.04.2020.
4. Как подступиться к fullstack-разработке [Электронный ресурс] / Хабр: сообщество IT-специалистов, 2019. – Режим доступа: https://habr.com/ru/. – Дата доступа: 14.04.2020.

Карчевская Яна Геннадьевна, студентка кафедры информационных технологий автоматизированных систем БГУИР, yana-03032017@mail.ru.

Научный руководитель: Трофимович Алексей Фёдорович, старший преподаватель БГУИР, зам. декана ФИТиУ.