УДК 004.415.23

Использование гибридных архитектур в создании веб-приложений

С. В. Вахрамов [1], Г. А. Зорин [2]

[1] s@vakhramoff.ru, [2] zorin.georgij-a@net.ugatu.su

ФГБОУ ВО «Уфимский государственный авиационный технический университет» (УГАТУ)

Аннотация. В настоящее время существует ряд традиционных паттернов разработки веб-приложений (например, MVC, MVP, MVVM, микросервисной архитектуры или «монолита»), применяемых при разработке и имеющих свои случаи применения, достоинства и недостатки. При этом, часто на стадии проектирования проекта могут быть заложены такие особенности, исходя из которых использование традиционных архитектур оказывается сложным, нецелесообразным или даже совсем невозможным. В статье приведена общая характеристика традиционных и гибридных архитектур, а также приведен пример применения гибридной архитектуры

Ключевые слова: гибридная архитектура; веб-приложение; паттерны разработки; MVC; MVP; MVVM; шаблон проектирования; микросервисная архитектура; монолитная архитектура.

О ТРАДИЦИОННЫХ АРХИТЕКТУРАХ ВЕБ-ПРИЛОЖЕНИЙ

К программным продуктам, как правило, существует ряд стандартных требований, таких как:

– масштабируемость, возможность для расширения;

– сопровождаемость, простота поддержки;

– тестируемость;

– надежность.

Кроме этого, требования предъявляются и к скорости разработки приложения.

Для достижения данных требований обычно необходимо использовать готовые унифицированные решения (или паттерны). Фундаментальным паттерном разработки является Model – View – Controller (MVC, рис. 1).

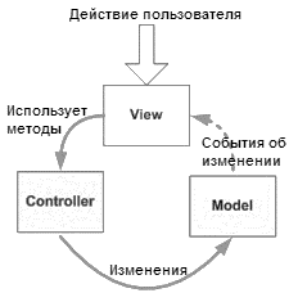

Рис. 1. Структура MVC

В нем под Моделью, обычно понимается часть, содержащая в себе функциональную бизнес-логику приложения, она полностью независима от остальных частей продукта. Модельный слой ничего не знает об элементах дизайна, и каким образом он будет отображаться. Модель обладает следующими признаками:

– это бизнес-логика приложения;

– модель обладает знаниями о себе самой и не знает о контроллерах и представлениях;

Иногда модель — это слой данных (база данных, XML-файл), иногда — это менеджер базы данных, набор объектов или просто логика приложения.

В обязанности Представления (View) входит отображение данных, полученных от Модели. Однако, представление не может напрямую влиять на модель (представление обладает доступом «только на чтение» к данным). Представление обладает следующими признаками:

– в представлении реализуется отображение данных, которые получаются от модели любым способом;

– в некоторых случаях, представление может иметь код, который реализует некоторую бизнес-логику.

Контроллер обеспечивает «связь» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия.

Применяются следующие распространенные виды паттерна MVC:

– Model – View – Controller (MVC)

– Model – View – Presenter (MVP, рис. 2)

– Model – View – View Model (MVVM, рис. 3)

MVP – шаблон проектирования пользовательского интерфейса, который был разработан для облегчения автоматического модульного тестирования и улучшения разделения ответственности в презентационной логике (отделения логики от отображения).

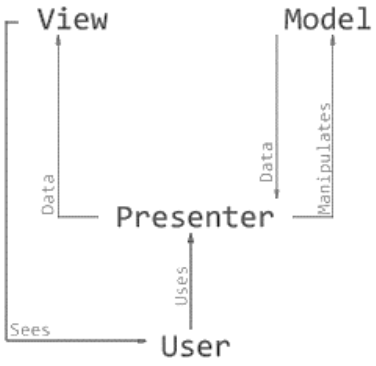

Рис. 2. Структура MVP

Шаблон MVP делится на три части:

– модель (Model);

– вид (View) – реализует отображение данных (из Модели), обращается к Presenter за обновлениями, перенаправляет события от пользователя в Presenter;

– представитель (Presenter) – реализует взаимодействие между Моделью и Видом и содержит в себе всю бизнес-логику; при необходимости получает данные из хранилища и преобразует для отображения во View.

Шаблон MVVM удобно использовать вместо классического MVC в тех случаях, когда в платформе, на которой ведется разработка, есть «связывание данных». В шаблонах проектирования MVC/MVP изменения в пользовательском интерфейсе не влияют непосредственно на Модель, а предварительно идут через Контроллер (англ. Controller) или Presenter. В таких технологиях как WPF и Silverlight есть концепция «связывания данных», позволяющая связывать данные с визуальными элементами в обе стороны. Следовательно, при использовании этого приема применение модели MVC становится неудобным из-за того, что привязка данных к представлению напрямую не укладывается в концепцию MVC/MVP.

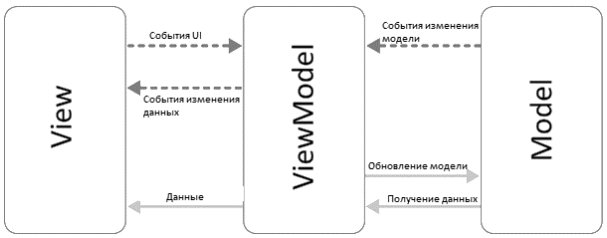

Рис. 3. Структура MVVM

Шаблон MVVM также делится на три части:

– *Модель*.

– *Представление* (*View*) — графический интерфейс. Выступает подписчиком на событие изменения значений свойств или команд, предоставляемых Моделью Представления. В случае, если в Модели Представления изменилось какое-либо свойство, то она оповещает всех подписчиков об этом, и Представление, в свою очередь, запрашивает обновленное значение свойства из Модели Представления. В случае, если пользователь воздействует на какой-либо элемент интерфейса, Представление вызывает соответствующую команду, предоставленную Моделью Представления.

– *Модель Представления* (*ViewModel*) – с одной стороны, абстракция Представления, а с другой – обертка данных из Модели, подлежащих связыванию. Она содержит Модель, преобразованную к Представлению, а также команды, которыми может пользоваться Представление, чтобы влиять на Модель.

О ГИБРИДНЫХ АРХИТЕКТУРАХ ВЕБ-ПРИЛОЖЕНИЙ

Часто на стадии проектирования проекта могут быть заложены такие особенности, исходя из которых использование традиционных архитектур (MVC, MVP, MVVM,

микросервисной архитектуры или «монолита») оказывается сложным, нецелесообразным или даже совсем невозможным. Например, усложняется процесс разработки, увеличивается время разработки проекта или оказывается нереальным внедрение какой-то функциональности просто из-за изначально неправильно выбранной архитектуры.

Далее будет рассмотрен пример гибридной архитектуры, созданной специально для проекта «Веб-приложение для обслуживания прокатного центра электро-бензоинструмента (созданное по модели SaaS — Software as a Service)».

ПРИМЕР ГИБРИДНОЙ АРХИТЕКТУРЫ

Выбор данной архитектуры был обусловлен тем, что проект в текущем виде состоит из достаточно большого количества небольших монолитных модулей. Выбор изначально монолитной архитектуры для разработки модулей по отдельности был связан с тем, что:

1. Так было удобнее вести разработку (вся нужна информация по модулю находится в одном файле и, по сути, используется им же);
2. Это незначительно, но ускорило скорость загрузки веб-страниц.

Для построения архитектуры было решено использовать гибридный подход: при создании приложения для увеличения скорости работы и удобства разработки была выбрана монолитная архитектура, которая дополнена элементами архитектуры MVC: так называемый контроллер (роутер) обеспечивает навигацию по модулям приложения и отвечает за проверку доступа к определенным разделам (например, чтобы обычный продавец не мог зайти в панель администратора, просто скопировав ссылку в браузере, проверка происходит на стороне сервера).

По сути, имеется один файл (контроллер), который выполняет функции:

1. Загрузки дополнительных вспомогательных подмодулей, используемых всеми модулями (например, функции работы с БД, временем, авторизацией и т.д.);
2. Роутинга (маршрутизации) по модулям;
3. Проверки уровня авторизации (при отсутствии доступа загружается модуль «Страница входа»).

Саму схему архитектуры можно увидеть на рис. 4. Контроллер (роутер) при необходимости может проверять права на доступ к модулю. Если же модуль доступен без прав, он может быть загружен напрямую без инициирования какой-либо проверки (например, страница входа доступна пользователю с любым уровнем доступа). Загрузчик модулей выполняет загрузку модулей, построенных по принципу монолитной архитектуры.

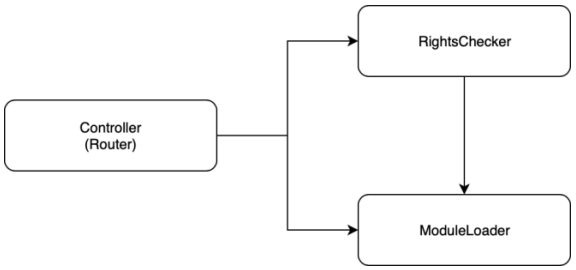

Рис. 4. Схема серверной части гибридной архитектуры веб-приложения

Благодаря использованию подобного подхода очень легко удалось организовать способы контроля пользователей с различным уровнем доступа к определенным модулям. В коде это выглядит достаточно легко и просто. При этом, при разработке конкретных модулей есть возможность полностью сфокусироваться на реализации конкретной функциональности без необходимости встраивания каких-либо проверок по уровню авторизации, перенаправления на страницы ошибок или авторизации, если пользователю по каким-то причинам доступ в модуль запрещен.

Данная гибридная архитектура соединила в себе элементы монолитной архитектуры и архитектуры MVC, при этом доказала свою эффективность в процессе разработки и дополнения системы новыми функциями.

СПИСОК ЛИТЕРАТУРЫ

1. Гагарина, Л. Г. Введение в архитектуру программного обеспечения. Учебное пособие / Л.Г. Гагарина, А.Р. Федоров, П.А. Федоров. - М.: Инфра-М, Форум, 2016. - 320 с. [L.G. Gagarina, A.R. Fedorov, P.A. Fedorov, “Introduction to software architecture. Tutorial”, (in Russian), Moscow: Infra-M, 2016]

2. Емельянова, Н. З. Проектирование информационных систем / Н.З. Емельянова, Т.Л. Партыка, И.И. Попов. - М.: Форум, 2009. - 432 с. [N.Z. Emelyanova, T.L. Partyka,

I.I. Popov, “Information Systems Design”, (in Russian), Moscow: Forum, 2009.]

3. Лукин, В. В. Технология разработки программного обеспечения. Учебное пособие / В.В. Лукин, В.Н. Лукин, Т.В. Лукин. - Москва: Гостехиздат, 2015. - 286 с. [V.V.Lukin, “Software development technology. Tutorial”, (in Russian), Moscow: Gostehizdat, 2015.]

4. Панюкова, Т. А. Проектирование программных средств / Т.А. Панюкова. - Москва: Гостехиздат, 2012. - 364 с. [T.A. Panyukova, “ Software Engineering”, (in Russian), Moscow: Gostehizdat, 2012.]

5. Проектирование информационных систем. Учебник и практикум / Д.В. Чистов и др. - М.: Юрайт, 2016. - 260 с. [D.V. Chistov et al., «Information Systems Design. Textbook and workshop», (in Russian), Moscow: Yurait, 2016.]

6. MVC, MVP and MVVM Design Pattern [Электронный ресурс] / Ankit Sinhal. — Электрон. текстовые дан. — Режим доступа: https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad, свободный. [MVC, MVP and MVVM Design Pattern [Online], (in English), Avaliable: https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad]

7. Монолитная vs Микросервисная архитектура [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: https://proglib.io/p/monolitnaya-vs-mikroservisnaya-arhitektura-2019-09-16, свободный. [Monolithic vs Microservice architecture [Online], (in English), Avaliable: https://proglib.io/p/monolitnaya-vs-mikroservisnaya-arhitektura-2019-09-16]

ОБ АВТОРАХ

ВАХРАМОВ Сергей Владимирович, магистрант 2-го курса факультета информатики и робототехники. Направление подготовки: «Программная инженерия».

ЗОРИН Георгий Андреевич, магистрант 2-го курса факультета информатики и робототехники. Направление подготовки: «Программная инженерия».

METADATA

Title: Using hybrid architectures to create WEB-applications

Authors: S. V. Vakhramov [1], G. A. Zorin [2]

Affiliation:

Ufa State Aviation Technical University (UGATU), Russia.

Email: [1] s@vakhramoff.ru, [2] zorin.georgij-a@net.ugatu.su

Language: Russian.

Source: Molodezhnyj Vestnik UGATU (scientific journal of Ufa State Aviation Technical University), no. 1 (22), pp. 34-37, 2020**.** ISSN 2225-9309 (Print).

Abstract: Currently, there is a number of traditional patterns of web-application development (for example, MVC, MVP, MVVM, microservice or monolithic architecture) that are currently in use (with their own advantages and disadvantages). At the same time, often at the design stage of the project, such features can be laid down, on the basis of which the use of traditional architectures may become difficult, inappropriate or even completely impossible. This article gives a general description of traditional and hybrid architectures, as well as an example of use of a hybrid architecture.

Key words: hybrid architecture; web application development patterns; MVC; MVP; MVVM; design pattern; microservice architecture; monolithic architecture.

About authors:

VAKHRAMOV, Sergey Vladimirovich, graduate student, Department of Computer Science and Robotics (USATU).

ZORIN, Georgiy Andreevich, graduate student, Department of Computer Science and Robotics (USATU).