СОВРЕМЕННЫЕ ПРИНЦИПЫ И ПОДХОДЫ К FRONTEND АРХИТЕКТУРЕ ВЕБ-ПРИЛОЖЕНИЙ

Сукиасян В.М.[1], Придиус Е.С.[2] Email: Sukiasyan1163@scientifictext.ru

[1]Сукиасян Владимир Мартунович - студент,
кафедра систем обработки информации и управления;
[2]Придиус Екатерина Сергеевна - студент,
кафедра защиты информации,
Московский государственный технический университет им. Н.Э. Баумана,
г. Москва

Аннотация: *в данной статье будет рассмотрено понятие архитектуры приложения, критерии хорошо построенной архитектуры приложения, понятие Frontend, его роль в приложении, структура, а также архитектурные особенности, ставшие актуальными в последнее время. Также будут рассмотрены и проанализированы принципы, на которых строятся современные Frontend-архитектуры веб-приложений, например, таких, как потоки данных и компонентный подход в архитектуре, их корреляция и сравнение между собой, а также примеры фреймворков, реализующих эти принципы.*

Ключевые слова: *архитектура, компонентный подход, поток данных, DOM, принципы, фронтенд.*

CONTEMPORARY PRINCIPLES AND APPROACHES TO FRONTEND WEB ARCHITECTURE

Sukiasyan V.M.[1], Pridius E.S.[2]

[1]Sukiasyan Vladimir Martunovich - Student,
DEPARTMENT OF AUTOMATIC INFORMATION PROCESSING AND CONTROL SYSTEMS;
[2]Pridius Ekaterina Sergeevna - Student,
DEPARTMENT OF INFORMATION SECURITY,
BAUMAN MOSCOW STATE UNIVERSITY,
MOSCOW

Abstract: *this article will discuss the concept of application architecture, the criteria for a well-built application architecture, the concept of Frontend, its role in the application, the structure, as well as architectural features that have become relevant in recent years. The principles on which modern Frontend architectures of web applications are built, such as data flows and component approach in architecture, their correlation and comparison with each other, as well as examples of frameworks that implement these principles will also be considered and analyzed.*

Keywords: *architecture, component architecture, data-flow, DOM, principles, frontend.*

УДК 004.415.25

Если рассматривать приложение как **систему** — то есть набор компонентов, объединенных для выполнения определенной функции, то можно сказать следующее:

Архитектура идентифицирует главные компоненты системы и способы их организации и взаимодействия. Также немаловажной характеристикой любой архитектуры является выбор таких методов, которые будут неизменны в будущем и составлять базис системы.

Архитектура — это организация системы, воплощенная в её компонентах, их отношениях между собой и с окружением.

Чтобы быстро понять какое архитектурное решение является верным необходимо задать себе вопрос: «Насколько сложно будет в будущем поменять данную архитектуру на другую?». Если ответ на этот вопрос «Да», то, скорее всего, вам нужно будет выбрать другую архитектуру.

Также можно выделить несколько критериев хорошей архитектуры:

- Гибкость и масштабируемость системы;
- Тестируемость и сопровождаемость;
- Независимость от бизнес-логики и предметной области;
- Эффективность работы.

Далее следует рассмотреть понятие архитектуры программного обеспечения (далее - ПО) в рамках Frontend части. Frontend называют клиентскую часть любого клиент-серверного ПО.

Структура Frontend, как и в принципе структуру почти любого ПО, условно можно поделить на какое-то определенное количество крупных блоков. Каждый из этих блоков имеет какую-то свою реализацию, которая позволяет связываться с другими блоками. Внутри этих блоков можно выделить

модули, внутри которых есть методы. И каждый узел этой цепочки вложенности имеет какой-то свой контракт общения, так называемые входные и выходные параметры. Можно сказать, что у каждого звена есть своя архитектура. И поэтому архитектура, в принципе, обладает свойством иерархичности.

Архитектурные особенности Frontend

Приведем некоторые особенности, ставшие актуальными для Frontend в последнее время:

- Веб приложения больше не привязаны к ПК (персональным компьютерам). В мире появляется все больше и больше гаджетов, мобильных телефонов, планшетов, IoT-Internet of Things (интернет-вещи). Стоит отметить, что больше всего интернет трафика проходит через мобильные приложения, а не через ПК.
- JavaScript, мощнейший язык для Веб разработки превратился в полноценный язык программирования и продолжает развиваться стремительными темпами;
- Для веб-приложений стали доступны такие API(Application Program Interface) как File System, Camera, аппаратные датчики и другие;
- UX (User Experience) становится решающим фактором при выборе продукта пользователями;
- Стек JavaScript технологий стал использоваться для кроссплатформенной разработки (одно и то же веб-приложение может быть запущено что на Android, что на iOS, что в бразуере).

И современная архитектура Frontend приложений вынуждена реагировать на все эти изменения новыми технологиями, фреймворками и подходами. Стоит отметить, что, если раньше любые технологии очень часто упирались в ограниченные ресурсные возможности устройств, то сейчас, ввиду прорывных технологий в производстве микросхем, эта проблема исчезает, что также влияет на тенденции архитектуры веб-приложений.

Современная архитектура Frontend базируется на трех принципах:

1. **Поток данных занимает центральное место в архитектуре**

Учитывая масштабы и сложность среды, в которой может выполняться frontend приложения, бизнес логика кода играет очень важную роль. По статистике большая часть времени разработки обычно тратится на отладку и чтение кода, поэтому необходимым условием любой архитектуры является быстрота нахождения нужного нам кода, ответственного за бизнес логику. Всякий раз, когда требуется внести какую-то новую функциональность в систему, чрезвычайно важно понимать, как она повлияет на будущую работоспособность и эффективность ПО.

Единственный способ осуществления такого рода задачи – это понимание на каждом этапе механизма перемещения данных по архитектуры ПО. Поэтому сегодня любой Frontend фреймворк построен на этом принципе, на четком разделении на модуль State (хранение и манипуляция с данными) и на модуль View (отображение данных).

К примеру, такой фреймворк как Angular I использовал принцип двунаправленного потока данных, что создавало неконтролируемые процессы передачи данных. После появления паттерна проектирования Flux, который использовал более эффективный и надежный принцип однонаправленного потока данных вышел Angular II, который использовал как раз принципа однонаправленности, как Flux.

2. **Компонентный подход**

Компонентный подход является неизбежным следствием первого пункта о потоке данных. Можно выделить три главных аспекта компонентного подхода:

1) Особое внимание потокам данных. Традиционные архитектуры frontend приложений ориентированы на горизонтальное распределение функционал. Такой способ еще можно сравнить с «размазыванием функционала тонким слоем» по всей архитектуре (довольно привычный способ в MV* паттернах). Но первый принцип о потоке данных требует, чтобы функциональность была ориентирована вертикально. Поэтому компонентой может являться какой-либо блок кода, инкапсулирующий какую-то одну «pure responsibility» логику (функцию с единственностью ответственности), вследствие чего может быть переиспользуемый

2) Постоянно усложняющие View (Представления). В настоящее время, в связи с появлением SPA приложений View становятся все сложнее и сложнее, что усложняет процесс внедрения новых feature. MV* подобные архитектуры не обладают возможностью решить проблему, в то время как использование компонентного подхода позволяет упрощать View классы.

3) Структура компоненты. Любая компонента должна состоять из 4 элементов: Графическая структура(HTML-**View**), Стилизация(CSS-**View**), Поведение(JavaScript-**Component**), бизнес логика(JavaScript-**Component/Model**).

Можно сказать, что компонентный подход позволяет переиспользовать и тестировать какие-то блоки нашего веб-приложения, не ломая ее архитектуру. К примеру, блок «Авторизации», состоящий из формы ввода полей и различных действий авторизации («войти», «забыли пароль» и т.д.), может быть вставлен как внутри отдельной страницы, так в внутри всплывающего диалогового окна, при этом не внося никакой лишней бизнес-логики в родительские элементы. В этом и заключается вся мощь компонентного подхода.

Резюмируя, приведем некоторые критерии компонент. Компоненты должны быть:

1. Независимые;
2. Слабосвязанные;
3. Переиспользуемые.

Компоненты могут быть достаточно сложные внутри, но обязаны быть простыми снаружи. Тут подразумевается, что интерфейс любой компоненты должен быть настолько простым, чтобы процесс подключения компоненты к родителю проходил без побочных эффектов.

На Рис. 1. приведен пример разбиения типичного View на компоненты.

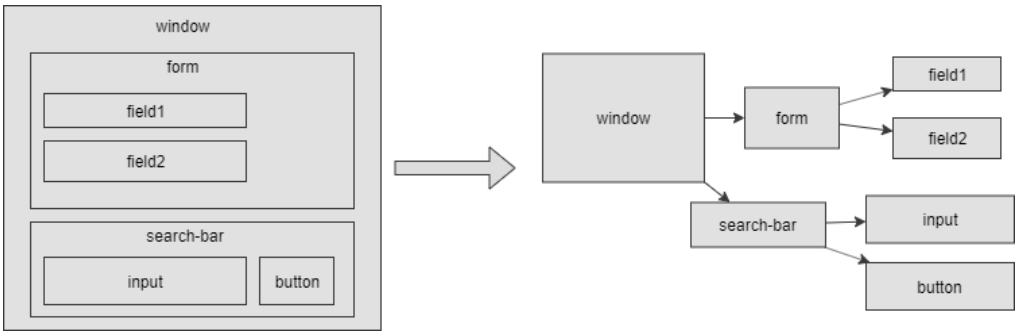

Рис. 1. Декомпозиции View

Компоненты делят на два типа:

1. Dummy-компоненты – компоненты, которые либо вообще не содержат никакой логики (чисто визуальные компоненты), либо содержат логику, которая глубоко инкапсулирована внутри компонента. К примеру, компоненты «Форма», «Чекбокс» или «Аватарка»
2. Smart-компоненты – компоненты, которые управляют множеством других компонентов, содержат в себе бизнес-логику и хранят какое-то состояние. К примеру «Блок рекомендаций» или «Фильтр».

3. Автоматическое выборочное обновление DOM

Не секрет, что все операции с DOM являются довольно дорогостоящими по ресурсам и времени, поэтому в какой-то момент Frontend сообщество пришло к идее создать технологию, которая смогла бы выполнять с DOM минимальное возможное количество действий по перерисовке каких-то конкретных DOM элементов при изменении данных веб-приложения, так называемый **State**. Собственно, к каком решению в итоге пришло сообщество Frontend в итоге:

1) Data-binding между Model и View.

В совокупности с компонентным подходом использование дата-биндинга является идеальным решением проблемы взаимосвязи данных(Model) и представления(View). View можно представить как функцию модели, которая знает какие поля данных нужны ей для отрисовки тех или иных компонент. К примеру, во фреймворке Angular существуют шаблоны, в React JSX –язык, являющийся смесью HTML и JavaScript языков.

2) Оповещение об изменении данных (Change detection).

Одним из требований также является возможность архитектурного решения каким-либо образом следить за тем, какие поля State были обновлены, удалены и добавлены. Данная возможность в Angular I реализовалась через метод грязной проверки (dirty checking), который проверял в бесконечном цикле через определенный период времени, какие поля были изменены. Во фреймворке React существуют неизменяемые структуры данных и посредством событийной шины происходит обновление необходимых полей State. В будущем Vue3 планируется использование объекта Proxy для реализации data checking (проверка данных на изменение).

3) Обновление DOM.

После того как были обнаружены изменения State, необходимо выполнить перерисовку DOM на основе них. В React для этого используется технология VirtualDOM.

Заключение

Таким образом, стоит сказать, что современные принципы построения архитектуры Frontend, как и в принципе любой другой области разработки ПО, основываются главным образом на компонентном и модульном подходе, который позволяет легко и быстро масштабировать любой продукт, занимаясь по большей части не проблемами реализации и внедрения в текущую архитектуру, а бизнес-требованиями к ПО. Также стоит отметить, что немаловажным критерием является скорость эффективность того или иного решения, вследствие чего и появились data-binding и VirtualDOM.

Список литературы / References

1. Современная архитектура frontend. Перспективы фронтенд инженера. [Электронный ресурс]. Режим доступа: https://blog.webf.zone/contemporary-front-end-architectures-fb5b500b0231/ (дата обращения: 31.10.2019).
2. Wikipedia. Архитектура программного обеспечения [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F/ (дата обращения: 09.11.2019).