УДК 004.422

Волков А.С.
студент магистратуры
3 курс, факультет заочного обучения
Поволжский государственный университет телекоммуникаций и информатики
Россия, г. Самара
Волкова К.А.
студент магистратуры
3 курс, факультет заочного обучения
Поволжский государственный университет телекоммуникаций и информатики
Россия, г. Самара

ОБЗОР АРХИТЕКТУРНЫХ КОМПОНЕНТОВ СОВРЕМЕННОГО ВЕБ-ПРИЛОЖЕНИЯ

Аннотация: *Статья содержит краткий обзор элементов классической трехуровневой архитектуры для современных Web-приложений. В качестве языка для реализации серверной логики взят язык программирования Java. Затрагиваются современные тенденции по разработке приложений данного типа.*

Ключевые слова: *архитектура ПО, web-приложение, java, база данных.*

Annotation: *The article contains a brief overview of the elements of the classic three-tier architecture for modern Web applications. The programming language Java is taken as the language for the implementation of server logic. It affects current trends in the development of applications of this type.*

Key words: *software architecture, web application, java, database.*

Классическая архитектура веб-приложения представлена в виде трехуровневой структуры: слой представления (пользовательский интерфейс), слой бизнес логики (сервер) и слой хранения данных (база данных). Очевидно,

что название того или иного слоя определяет суть решаемой на данном слое проблемы. При этом функциональная характеристика системы, как совокупность возможных операций системы, может затрагивать как все три слоя, так и только один из них. Кроме функциональности, информационным системам свойственны такие категории стабильность и надежность, масштабируемость и кластеризация, гибкость и сложность поддержки. В данной статье приведен обзор современных средств, применяемых в различных архитектурах приложений с позиции серверного разработчика на языке программирования Java.

Слой представления

Технически существует возможность разработки пользовательского интерфейса с помощью сервер-ориентированных технологий на базе Java: есть реализации спецификаций Java Enterprise Edition вроде JavaServer Faces, JavaServer Pages; шаблонизатор Spring Thymeleaf или библиотеки, транслирующие Java-код в оптимизированный Javascript - GWT (Google Web Toolkit), Vaadin. Однако в индустрии основной выбор для пользовательского интерфейса строится вокруг применения HTML в связке с JavaScript или любого производного от него фреймворка (программного каркаса) вроде AngularJS, React, Ember. С точки зрения архитектора системы выбор технологии для разработки пользовательского интерфейса часто диктуется навыками и умениями текущей команды разработчиков. В случае, если в команде программисты не обладают навыками работы с Javascript-решениями для реализации данной задачи, то вполне оправдан выбор сервер-ориентированных технологий вроде GWT. При плохом проектировании данный подход может неизбежно привести к повышению связанности серверного кода и его визуального представления. Разрешить данную проблему можно следуя принципам модульности и единственной ответственности каждого компонента системы. Неплохим решением было бы разделение слоя бизнес логики и пользовательского интерфейса сетевыми

вызовами.

Слой бизнес логики

Слой бизнес логики, серверный слой или так называемый Middleware может базироваться на реализации спецификаций Java Enterprise Edition [1] - Web Application Server вроде WildFly от RedHat, WebLogic от Oracle или WebSphere от IBM. Данные сервера приложений предлагают разработчику целую платформу, берущую на себя обязанность по соединению с базой данных, кластеризацию, балансировку нагрузки и мониторинг. Однако наиболее популярным решением для серверной логики сегодня является другой подход – использование Spring Framework [2]. Он обладает высокой степенью модульности, имеет множество интеграций с популярными библиотеками, позволяет реализовать различные подходы вроде реактивного программирования основанное на событийной модели, REST-сервисов. Среди модулей данного программного каркаса значатся: Spring Data JPA (работа с БД), Spring MVC (API), Spring Security (безопасность и аутентификация) и т.д. Многие начинающие разработчики сталкиваются с проблемой конфигурации базового окружения, которое бы позволило приступить к непосредственному написанию самого приложения. В данном вопросе производители стремятся облегчить настройку и выпускают версии фреймворка в виде стартеров с набором заранее предопределенных параметров и автоматическим подключением более низкоуровневых библиотек. Яркий пример – Spring Boot.

Брокеры сообщений

Для взаимодействия между компонентами или интеграции с внешними системами в архитектуре приложения, не обязательно написанного на Java, может применяться брокер сообщений. В свою очередь брокер сообщений осуществляет маршрутизацию, возможно гарантирует доставку, распределение потоков данных, подписку на нужные типы сообщений. Примерами брокеров сообщений могут служить – RabbitMQ, ActiveMQ, Kafka.

Слой хранения данных

Реляционные базы данных, например, Oracle Database, PostgreSQL, MySQL, предоставляют широкий функционал по хранению, индексации, репликации и масштабированию данных. Выбор конкретной базы зависит от потенциальной нагрузки, объема данных и финансовых возможностей. SQL базы данных все также остаются стандартом для индустрии, но можно выделить класс задач, которые специализированные средства решают эффективней. Приведем пример: если схема данных меняется так стремительно, что поддерживать разработку в реляционной базе данных становится крайне сложно, решением может быть использование MongoDB. Эта документоориентированная СУБД не требует описания схемы таблиц данных. Проведем сравнение NoSQL и SQL баз данных.

Таблица 1. Сравнение SQL и NoSQL баз данных

MySQL	MongoDB
Проверенная временем технология, реализующая стандарт SQL. Можно реализовать переход на другие SQL базы данных. Есть поддержка транзакций, возможность тонкой настройки. Относительно сложный язык запросов через SQL. Реляционная структура требует большего планирования и контроля.	Гибкий JSON формат документов, не требующий изучения сложного SQL. Как следствие простые запросы позволяют допускать меньше ошибок при разработке. Есть возможность динамически менять схему документа. Встроенная масштабируемость. Плохо подходит для транзакционных операций. NoSQL-структура дает большую скорость разработки.

Для хранения и быстрого доступа к некоторой очень востребованной выборке данных, например, сведений об учетной записи текущего пользователя и его правах в системе, могут применяться распределенные кэши (хранилища типа ключ-значения). С удешевлением стоимости оперативной памяти это привело к буму развития распределенных хранилищ данных в памяти, так называемых In-Memory Data Grid. Яркие представители: memcached, Hazelcast, Apache Ignite. Для некоторых доменных областей существуют специализированные средства хранения данных, например, хранилище типа «ключ-значение» Chronicle Map применяется в области

трейдинга.

Вывод

Подводя итог, надо отметить, что это далеко не все, что можно встретить в архитектуре современного веб-приложения. Необходимость выдерживать нагрузку под большим числом запросов поставило перед инженерами задачу масштабирования и эффективного развертывания дополнительных экземпляров, балансировки нагрузки (Docker, Kubernetes). Крупные монолитные приложения стали разбивать на небольшие «микросервисы» [3] (Spring Cloud, Netflix OSS). В трендах интеллектуальный анализ, машинное обучение, Big Data. Новые объемы данных и нагрузки порождают спрос на системы мониторинга и поиска по системам логирования (Elasticsearch, Logstash, Kibana). Помимо компонентов архитектуры активно совершенствуются и инструменты разработки, тестирования, сборки и развертывания. Выбор той или иной архитектуры, а также конкретных её компонентов должен проводиться, отталкиваясь от бизнес-требований, технических деталей конкретных решений, компетенций и размера команды разработчиков.

Использованные источники:

1. Гупта, А. Java EE 7: Основы / А. Гупта.— М.: Вильямс, 2014.— 336 с.
2. Уоллс, К. Spring в действии / К. Уоллс.— М.: ДМК Пресс, 2013.— 752 с.
3. Microservices. [Электронный ресурс]. URL: https://martinfowler.com/articles/microservices.html (дата обращения: 15.12.2018).