1. Документация - stanislav.yuzhakov - Confluence

При изучении имеющихся описаний реализуемой системы использовалась информация из следующих источников:

- Confluence:  Briolink
- Gitlab: https://gitlab.com/briolink/network - Connect to preview
- Figma:  Briolink

В результате анализа имеющейся документации установлено:

1. Наличие структурированных по разделам сайта и сценариям дизайнерских макетов
2. Наличие некоторых артефактов верхнеуровневой архитектуры системы
3. Наличие описания для подготовки к работе нового участника команды (onboarding). Описание частичное, в нём отсутствуют некоторые необходимые для начала работы этапы.
4. Отсутствие описания проектных и бизнес-требований реализуемой системы
5. Отсутствие аналитических описаний системы (сценарии использования системы, функциональные и нефункциональные требования)
6. Отсутствие описаний системной архитектуры системы
7. Отсутствие описаний процессов сборки и развёртывания системы (CI/CD)
8. Отсутствие описаний уровня инфраструктуры (deployment and connectivity)
9. Отсутствие описаний конфигурационных параметров системы

10. Отсутствие описаний для эксплуатации системы (алертинг, мониторинг, логгирование)

11. Отсутствие описаний критериев и сценариев тестирования системы

Отсутствующие артефакты и описания системы

Артефакт	Критичность
Бизнес-требования, тех. задание и аналитическое описание	
Требования бизнеса к реализуемой системе: описание бизнес-процесса взаимодействия с пользователями, общие требования к функциональности и SLA системы.	HIGH
Аналитическое описание системы: описание структуры системы (описание разделов сайта, отправляемых пользователю уведомлений) и UC.	HIGH
Аналитическое описание системы: детальные функциональные и нефункциональные требования для отдельных UC и системы в целом.	MEDIUM
Аналитическое описание сущностей используемых в системе (Словарь сущностей, аналитическая модель данных системы)	MEDIUM
Архитектура системы	
Описание IT-доменов. Функциональность, границы, связанность с другими доменами. Правила, особенности и налагаемые ограничения.	HIGH
Описание входящих в состав домена приложений (компонентов). Роль и функциональность приложения. Компонентная схема домена.	HIGH
Описание архитектуры микро-фронтенда	MEDIUM
Описание Gateway (схема роутинга через CDN, reverse-proxy и gateway)	MEDIUM
Диаграмма информационных потоков внутри домена и между доменами	LOW
Описание инфраструктуры (приобретаемые ресурсы AWS, их коннективности и уровни)	LOW
Технологическая документация	
Backend	
Описание API: список операций и их краткое описание	HIGH
Описание структуры БД: таблицы, ключи, индексы	HIGH
Описание сообщений SQS: формат, инициатор сообщения	HIGH

Документация пользователя API. Самодостаточная документация на API с описанием методов и используемых в них моделей данных, обязательности и правил заполнения запросов, полноты ответов, возможных ошибок.	MEDIUM
Документация разработчика API. Описание логики работы методов, используемых сущностей БД, генерируемых событий, поддерживаемых в логике аналитических функциональных требований.	LOW
Frontend	
Описание реализуемых в рамках домена страниц и компонентов	HIGH
Описание структуры UI Kit, guidelines по дизайну и стилям	LOW
Общая документация	
Описание технологического стека домена (можно как ссылка на общий стек с выделением особенностей каждого домена, при их наличии)	LOW
Описание конфигурационных параметров и настроек приложений	LOW

Критичность

HIGH - Необходимая для освоения основ проекта документация. Описание целей, структуры, принципиальных основ и базовых сценариев системы. Позволяет новым участникам проекта самостоятельно сориентироваться и сопоставить требования с разработанным кодом.

MEDIUM - Детальное описание реализуемой системы. Позволяет зафиксировать основные бизнес и технические решения для их передачи другим участникам проекта.

LOW - Технологическая документация, которая описывает подробности реализации в простом легко-доступном виде (в противопоставление программному коду). Необходима для ускорения получения информации техническими специалистами.

QA и OPS

Документация по QA и OPS не рассматривается в данном обзоре, так как данные процессы не являются системно организованными на момент исследования проекта.

2. Архитектура - stanislav.yuzhakov - Confluence

Обзор

Общее

Инфраструктурное и организационное устройство системы базируется на принципах *Domain Driven Design* - организации системы как совокупности слабо-связанных компонентов, каждый из которых относится только к одному функциональному домену. Выбранный подход позволяет гибко масштабировать как производительность и физические ресурсы системы, так и скорость разработки за счёт изолированной параллельной разработки каждого домена отдельной командой.

Frontend

Пользовательский интерфейс является *веб-приложением*: т.е. клиент-серверным приложением, в котором клиент взаимодействует с веб-сервером при помощи браузера. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит через Интернет.

Фронтенд реализуется по технологии *SPA (Single Page Application)*: это web-приложение, размещенное на одной странице, которая для обеспечения работы загружает все javascript-файлы (модули, виджиты, контролы и т.д.) , а также файлы CSS вместе с загрузкой самой страницы.

Фронтенд основан на технологии *микро-фронтендов* - архитектурном подходе, в котором независимые веб-приложения собраны в одно большое приложение (веб-сайт). Подход дает возможность объединить в одном приложении разные виджеты или страницы, написанные разными командами с использованием разных фреймворков.

Код фронтенда обрабатывается с помощью технологии *server-side rendering* - технологии, которая позволяет поисковому роботу индексировать страницы SPA фронтенда. Запланирована на вторую половину декабря.

Backend

Серверная часть организована в соответствии с принципами *микросервисной архитектуры* - сервис-ориентированной архитектуры, направленной на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей.

Бэкенд поставляет *GraphQL API* для обмена данными. GraphQL - язык API позволяющий управлять составов получаемых от сервера данных и имеющий инструменты описания доступных запросов и их форматов.

Для хранения данных используется *объектно-реляционная СУБД PostgreSQL*. СУБД основана на реляционной модели данных, но так же поддерживающая некоторые технологии, присущие объектно-ориентированным СУБД и

реализующая объектно-ориентированный подход.

Взаимодействие между микросервисным приложением и СУБД использует подход *CQRS* при котором схемы БД для записи данных (мастер) и схемы для чтения физически разделены. Внедрение CQRS в приложение может улучшить его производительность, масштабируемость и безопасность.

Синхронизация данных между write и read схемами организована по принципу *шины сообщений (message bus)*. Подход подразумевает формирование версионируемых сообщений отправляемых в message broker систему, откуда заданные потребители этих сообщений могут их читать и выполнять соответствующие действия (в данном случае - производить обновление read схемы БД).

Серверные приложения основаны на технологиях *Spring Boot Framework*. Фреймворк содержит инструменты для решения популярных задач серверной разработки, например: ORM, GraphQL-контроллеры, DI фреймворк, управление конфигурационными параметрами, контроль доступа.

Infrastructure

Инфраструктура развёртывания базируется на платформе *Kubernetes*. Kubernetes - система для оркестровки контейнеризированных приложений: автоматизации их развёртывания, масштабирования и координации в условиях кластера.

Идентификация и авторизация пользователей реализована с помощью системы *Keycloak*. Keycloak - надежная и гибкая IAM система с поддержкой актуальных протоколов OAuth 2.0 и OpenID Connect.

Авторизация пользователя реализована по сценарию *Authorization Code Grant* - классический сценарий обмена credentials на OAuth токены. В настоящий момент рекомендуется использовать его усиленную версию: *Authorization Code Grant with PKCE*.

Технологический стек

Frontend

- **Vue.JS 3.X**
- TypeScript
- CSS (bootstrap)
- Web components (LitElements)
- Microfrontend / Qiankon
- GraphQL / Apollo
- Vite

Backend

- **Kotlin, Java**

- Spring Framework (Spring Boot, Spring Data, Spring Security)
- GraphQL / DGS
- Open API / Swagger
- AWS Aurora, Postgres
- Liquibase
- Hibernate
- AWS SQS
- Gradle

Infrastructure

- Kubernetes, Docker
- AWS (deployment, setup)
- Helm, Werf
- Git, Gitlab, CI pipeline

Оценка технологий

Используемые технологии в полной мере соответствуют выбранным архитектурным подходам. Ключевые технологии являются либо уже устойчивыми (мэинстримными), либо перспективно-устойчивыми в горизонте следующих нескольких лет.

Оценка архитектурных решений

Система базируется на прогрессивных и гибких подходах, которые позволяют проводить дальнейшее развитие без необходимости радикального изменения архитектуры (рефакторинга) для обслуживания большого количества пользователей или увеличения участвующих в разработке команд. Можно сказать, что архитектурные решения соответствуют поставленной задаче в разработке MMP - минимального рыночного продукта, который не потребуется переписывать после первого раунда финансирования и который готов к масштабированию даже в случае взрывного интереса к этому продукту. Однако для фактического проведения масштабирования необходимо организовать процессы проектирования, документирования, тестирования и эксплуатации разрабатываемого решения. В настоящий момент эти процессы либо отсутствуют, либо эпизодически выполняются разработчиками.

Оценка производительности

Выбранные архитектурные решения имеют высокие перспективы как в горизонтальном масштабировании пропускной способности системы, так и в возможностях оптимизации быстродействия (времени ответа системы). Решение позволяет эластично масштабировать отдельные бизнес-домены без влияния на другие домены. Однако для получения высоких показателей производительности

необходимо выполнять регулярное практическое тестирование производительности и выполнять тюнинг java-сервисов и правил их масштабирования в кластере для оптимизации быстродействия и потребления ресурсов.

Оценка отказоустойчивости

Выбранные архитектурные решения имеют высокие перспективы с точки зрения обеспечения высокой отказоустойчивости системы по следующим показателям: аварийный downtime, сервисный downtime, время восстановления системы. Для получения высоких практических показателей необходимо выполнять регулярное тестирование (Stress Testing, Failover Testing, Recovery Testing) и соответствие необходимому SLA на сервисный downtime.

Оценка эксплуатации

Микросервисная архитектура имеет повышенные эксплуатационные расходы на физические ресурсы, т.к. каждый микросервис расходует некоторый минимальный объём ресурсов на поддержку своей работы. Так же большее многообразие компонентов, связей между ними и их конфигураций увеличивает трудоёмкость поддержки и обновления системы и предъявляет повышенные требования к квалификации ответственных за поддержку инженеров.

Использование оркестратора Kubernetes позволяет повысить стабильность эксплуатации и снизить количество инцидентов вносимых человеческим фактором. Но при этом повышает эксплуатационные расходы (необходимостью наличия дополнительных узлов) и резко повышает требования к квалификации ответственных за поддержку инженеров (эксплуатация системы может выполняться только devops-специалистами, обладающих экспертизой в Kubernetes экосистеме).

Так же в ландшафте системы присутствуют другие технологии с повышенной сложностью поддержки (GraphQL, SQS), предъявляющие повышенные требования к квалификации инженеров, которые занимаются эксплуатацией системы (тестированием и локализацией проблем).

Общеизвестно, что высокая технологическая сложность системы, предъявляет повышенные требования к сопроводительной документации.

Важно отметить, что в данный момент, архитектура платформы не описывает необходимые для промышленной эксплуатации (выполнения SLA) аспекты: мониторинг, алертинг, резервное копирование и восстановление.

Оценка разработки

Выбранные архитектурные подходы и технологический стек имеют высокий уровень масштабируемости и изоляции процессов разработки. Использование подхода *Domain Driven Design* для изолированной работы одной команды над одним бизнес-кейсом позволяет параллельно разрабатывать каждый домен изолированной командой и соответственно масштабировать общую скорость разработки проекта. Однако для организации разработки несколькими командами необходимо организовать процессы проектирования и согласования доработок. Недостаточная организованность в развитии ландшафта чревата конфликтами (несовместимостью) в результатах работы нескольких команд.

Высокий уровень конфигурационного управления и автоматизации текущего процесса разработки позволяют запустить CI/CD процессы для выпуска обновлений.

Технологический стек, который можно оценить достаточно обширным и включающим применение относительно новых технологий, накладывает значительно высокие ограничения на квалификацию разработчиков, технических менеджеров и QA специалистов.

Отдельно стоит отметить, что высокая технологическая сложность системы, так же предъявляет повышенные требования к качеству и полноте технической документации. Недостаточность описаний будет негативно влиять на время адаптации разработчиков на проекте и скорости ведения разработки.

3. Оценка трудозатрат

Автор: stanislav.yuzhakov (Unlicensed)
Дек. 23, 2021

Ниже представлена экспертная экспресс-оценка трудозатрат на реализацию аналогичного решения с идентичной архитектурой. Оценка даётся в человеко-днях исходя из 8ми-часового рабочего дня.

Раздел	**Архитектура (ч*д)**	**Frontend (ч*д)**	**Backend (ч*д)**
Log in	1	3	4
Sign up	1	5	5
User Account	3	5	10
Company Account	3	7	15
Connections create	3	4	7
Connections page	1	3	3
Connections edit	1	3	10
Connections confirm	1	4	3
Company connections	1	4	3

User connections	1	2	2
Statistics	1	5	5
Common (tech + devops)	10	20	30
Bugs (30%)	8,1	19,5	29,1
Total	35,1	84,5	126,1

4. Оценка утилизации физических ресурсов

Автор: stanislav.yuzhakov (Unlicensed)
Последнее обновление: дек. 23, 2021

Обзор

На момент исследования продуктивная конфигурация развёрнута в k8s кластере, который использует 2 сервера класса t4g.xlarge (AWS) для запуска подов. Серверы имеют следующую конфигурацию:

Ресурс	Количество
vCPU	4 (arm64)
Memory (GiB)	16
Baseline Performance / vCPU	40 %
Network Burst Bandwidth (Gbps)	Up to 5
EBS Burst Bandwidth (Mbps)	Up to 2,780
OS	Linux
Pods capacity	58

При этом средняя утилизация ресурсов составляет:

Ресурс	**Количество**
vCPU	< 30%
Memory (GiB)	< 10 GiB
Pods	< 40

Конфигурация развёртывания предусматривает по 1 реплике каждого микросервиса.

Оценка

Выделенное количество физических ресурсов предусматривает 2-3 кратное масштабирование системы, необходимое как для быстрой установки обновлений системы без перерыва в обслуживании пользователей, так и для потенциального запаса для роста количества микросервисов.

Для проведения оптимизации утилизации физических ресурсов необходимо провести нагрузочное тестирование системы с последующим тюнингом JVM. В ходе нагрузочных испытаний рекомендуется провести сравнительное исследование влияния мощности CPU на показатели системы (сравнение с процессорами класса t3 и t2), а также подбор следующих параметров: размера JVM heap, количества worker threads сервера приложений (tomcat), лимитов для подов.

5. Эксплуатация - stanislav.yuzhakov - Confluence

Логгирование и мониторинг

Обзор

На момент исследования в системе присутствуют только базовые инструменты мониторинга и диагностики:

- *Панель мониторинга Kubernetes*. Стандартный инструмент для наблюдения и отслеживания физических ресурсов кластера, а так же конфигурации и активности развёрнутых приложений. Инструмент позволяет отслеживать активность отдельных развёрнутых приложений, но не содержит инструментов для мониторинга качества предоставляемых сервисов.

- *Prometheus* - инструмент для сбора и хранения метрик.

- *Grafana* - инструмент для визуализации метрик и построения панелей мониторинга, содержащих аггрегированные данные показателей систем (как физические ресурсы, так и технические показатели или бизнес-метрики).

- *Аггрегатор логов AWS CloudWatch*. Стандартный инструмент сбора и просмотра лог-файлов приложений.

На этой конфигурации выполняется мониторинг базовых показателей активности приложений (как подов k8s) и утилизации физических ресурсов. Логгируются события ошибок (стектрейсы) и стандартный вывод приложений.

Веб-сайт не подключён к инструментам аналитики и логгирования, соответственно отсутствует какая-либо аналитическая или диагностическая информация о работе веб-сайта.

Оценка

Имеющиеся инструменты и данные позволяют выполнять мониторинг (наблюдение) за активностью отдельных приложений в системе и общей утилизацией физических ресурсов. Имеющаяся организация логов позволяет наблюдать наличие или отсутствие ошибок выполнения.

Этих инструментов недостаточно для того чтобы:

- Выполнять мониторинг качества предоставляемого сервиса. Т.е. отслеживать корректность и быстродействие выполнения конкретных бизнес-сценариев системы.

- Выполнять локализацию возникающих на продакшене ошибок (или непредвиденного поведения системы) на уровне достаточном для передачи проблемы разработчику для её оперативного устранения. Имеются ввиду проблемы обнаруженные сторонними пользователями, которые не могут выполнять самостоятельный сбор диагностических данных, а так же проблемы выявленные постфактум (например, по наличию ошибок в логах).

- Выполнять диагностику проблем производительности, вызванных увеличенным потоком запросов, неоптимальной реализацией сценариев или проблемной конфигурацией middleware (jvm, драйверов БД и SQL, конфигурацией пулов логических ресурсов)

Для получения перечисленных выше возможностей рекомендуется расширить инструментарий следующими инструментами:

Backend

- Метрики входящих запросов: количество, время выполнения, результат (http код, успех/неуспех). Для Spring Boot можно реализовать с помощью библиотеки actuator.
- Возможность логгирования запросов вместе с их содержимым. Для Spring Boot можно реализовать с помощью библиотеки LogBook.
- Трассировка запросов (сквозной идентификатор запроса, объединяющих операции выполненные в рамках одной операции). Можно реализовать с помощью приложений: AWS X-Ray, Zipkin, Jaeger, Spring Sleuth.
- Метрики подключений к СУБД и SQS: загруженность пулов соединений, время выполнения запросов, количество операций. Предоставляется библиотекой actuator.
- Метрики веб-сервера: загруженность пула worker-threads. Предоставляется библиотекой actuator.
- Метрики JVM: утилизация памяти, интенсивность работы GC. Предоставляется библиотекой actuator.
- Метрики Kubernetes: Pod restart rate, OOMEvents, Available capacity. Реализуется самописным решением.

Frontend

- Базовая аналитика на базе Google Analytics или аналогичного решения. Позволяет мониторить как общий траффик на сайт, так и траффик по отдельным страницам.
- Разметка тегами Google Tag Manager или аналогичное решение. Позволяет отслеживать как положительные, так и негативные события: изменение состояний страницы, ошибки выполнения JS, переходы на несуществующие страницы, ошибки заполнения форм и чекаутов.

6. Безопасность и тестирование - stanislav.yuzhakov

Безопасность

Защита сетевого уровня

В планах развития системы заявлено использование *Cloudflare* или другого аналогичного решения. Решения этого класса позволяют защититься от таких атак как DDoS, атака Ботов или несанкционированный доступ.

Защита веб-приложений

Так же существуют угрозы направленные на использование уязвимостей в самих веб-приложениях. Самые распространённые из них собраны под названием *OWASP Top 10*. В них входят такие уязвимости как инъекции, неверная конфигурация безопасности, небезопасная криптография, неработающий контроль доступа и другие. Для защиты от них рекомендуется проводить регулярное сканирование приложений такими сканерами как SonarQube или OWASP ZAP, а так же выполнять аудит безопасности.

Тестирование

В настоящий момент разработка системы не имеет выстроенного процесса тестирования, т.е. тестирование выполняется эпизодически, без сформулированных критериев и сценариев, силами разработчиков.

Для контроля качества разрабатываемой системы и стабильности вносимых изменений рекомендуется внедрить такие базовые практики **функционального тестирования** как:

- Smoke тестирование - проверка того, что критически-важные сценарии системы работают в соответствии с ожиданиями
- Регрессионное тестирование - проверка того, что свежие изменения в коде или приложении в целом не оказали негативного влияния на уже существующую функциональность или набор функций.
- Рекомендуется автоматизировать эти 2 вида тестирования для ускорения выполнения проверок перед установкой сборки на продакшен.

Так же на момент исследования не проводилось нагрузочного тестирования системы. Выполнение **нагрузочного тестирования** позволяет на практике подтвердить высокий потенциал масштабируемости и производительности системы, определить необходимые конфигурации для желаемого уровня производительности и выполнить оптимизацию физических ресурсов. Рекомендуется провести следующие виды нагрузочного тестирования:

- Тестирование масштабируемости (Scalability Testing) - тестирование системы для проверки работоспособности механизмов маштабируемости.

- Тестирование потенциальных возможностей (Capacity Testing) - тестирование системы для проверки того, что созданная конфигурация способна выдерживать ожидаемую нагрузку..
- Тестирование объемов (Volume Testing) - тестирование системы в конфигурации с большим количеством данных. Такое тестирование отвечает на вопрос - как будет работать система, если в ней появится большое количество данных.
- Стрессовое тестирование (Stress Testing) - тестирование системы для определения её устойчивости и поведения в условиях аномально высоких нагрузок.