Проект Siforeca

Решённые на сегодня три задачи:
1. Разработана детальная архитектура проекта.
2. Реализованы отдельные модули.
3. Готовая система с реализованным модулями доступна для изучения и тестирования.
Подробнее о каждой из выполненных задач.

Разработка архитектуры

Выбор подходящей архитектуры происходил, исходя из ожидаемых задач и требований перед системой. Такими требования были:

1. Модульность.
2. Расширяемость и заменяемость.
3. Асинхронность и событийность.
4. Масштабируемость под нагрузкой.
5. Облачное размещение.
6. Безопасность.
7. Современные инструменты программирования и администрирования.

Рассмотрим каждое из требований отдельно:

1. **Разделение на модули с чётким очертанием границ:**
   - **Модуль поставщик данных**, состоящий из трёх подмодулей:
     - ввода данных администратором
     - автоматического загрузчика данных с использованием внешних источников с открытым или закрытым (коммерческим) API
     - web скрепера для получения данных, не доступных через API
   - *Языки реализации модуля Python3.*
   - , являющийся фасадом над несколькими базами данных, позволяющим проводить валидацию, фильтрацию и отображать текущий статус хранящейся информации.

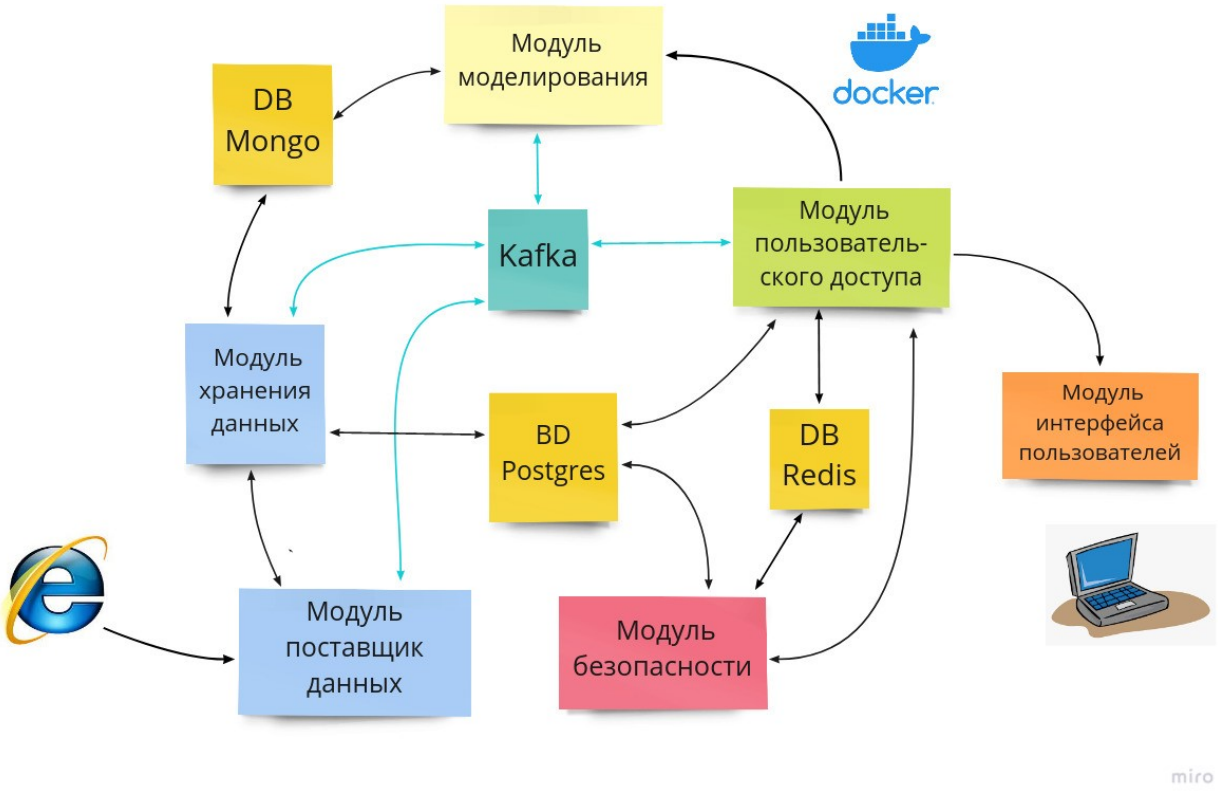
Модуль в свою очередь состоит из двух частей, предназначенных для:
    - хранения исходных данных
    - хранения результатов моделирования и прогнозирования
  - **Модуля моделирования,** выполняющего в асинхронном режиме расчёт прогнозных показателей, которые в свою очередь передаются в модуль хранения. Модуль состоит из множества подмодулей, которые могут произвольно собираться в последовательно-параллельные цепочки обработки данных. Сбор подмодулей выполняется пользователями с уровнем доступа «исследователь» с целью организации различных алгоритмов обработки информации. Информацию о текущих настройках модуль хранит в базе данных NoSQL (MongoDB).
  - *Языки реализации модуля Python3, Java, C#.*
  - **Модуль пользовательского доступа** к данным и инструментам прогнозирования и администрирования системы. Модуль представляет собой непосредственно API для доступа к вышеупомянутым модулям и поддерживает три уровня доступа:
    - администратора
    - исследователя
    - пользователя.
  - **Модуль безопасности.** Осуществляет аутентификацию и авторизацию для всех остальных модулей системы.
  - **Модуль интерфейса пользователей** (интерфейс адаптируется под уровень доступа)**,** включая **дашборды визуализации данных** (как исходных так и прогнозных). Если все предыдущие модули были серверной частью архитектуры, то данный модуль представляет собой программный продукт, непосредственно выполняемый в браузере.
  - *Языки реализации модуля JavaScript, Vue2.js.*
2. **Слабое связыванием модулей**, позволяющее выполнять замену и расширение:
    1. Каждый модуль автономен и существует в виде отдельной программы (группы программ), выполняемы в виртуальной машине или контейнере.
    2. Модули общаются между собой по сети с помощью API, открытого для всех модулей, но закрытого для внешнего мира, Только модуль **пользовательского доступа** открыт и является шлюзом с внешним миром.

3. Документируется и фиксируется только API модулей, внутренняя реализация может быть изменена.
4. Протоколы взаимодействия модулей между собой HTTP, REST и формат JSON. Возможна миграция на Protobuf при увеличении масштабов системы.
5. *Инструменты реализации Docker, Kubernetes.*

3. **Цикл обработки информации осуществляется в модулях асинхронно:**
   1. Модули запроса данных начинают работу по запрограммированному расписанию (или команде администратора) и по завершению работы над очередными блоками данных передают сообщения системе о том, что данные доступны и сохранены в базах данных.
   2. Модули прогнозирования, основанные на сложной логике, включая машинное обучение, выполняют обработку данных в достаточно больших временных рамках. Кроме того данные модули могут быть изменены, подключены в различной последовательности (pipeline) или модифицированы пользователями группы «исследователи».
   3. Как только очередные результаты прогнозирования и моделирования становятся доступными и сохранены в базах данных, **модули обработки** направляют соответствующее сообщение в систему. Сообщение подтверждает готовность для модулей поставщиков, которые в противном случае привели бы систему в состояние затопления.
   4. *Инструмент асинхронной передачи сообщений и потоков данных Kafka.*
4. **Масштабируемость под нагрузкой**
   1. Модули допускают вертикальное и горизонтальное масштабирования за счёт увеличения мощности машин и/или увеличения количества параллельно работающих контейнеров (или виртуальных машин) с модулями.
   2. Работу модулей в параллельном режиме и их масштабирование организовывают инструменты контейнеризации.
   3. Диагностика здоровья и нагруженности системы осуществляется инструментами мониторинга.
   4. Нагрузка между параллельными модулями распределяется балансировщиками (loadbalancer).
   5. Используется кеширование некоторых повторяющихся запросов между отдельными модулями. Политика

кеширования настраивается исходя из логики процессов.

6. Исходные внешние запросы, поступающие на **модуль пользовательского доступа** проходят через реверсионный прокси и также кешируются и балансируются.
7. *Иснтрументы организации: Nginx, Memcach, Redis, Prometheus, Graphana.*



5. **Облачность.** Предполагается размещение кода, конфигураций и производственной системы (production) на нескольких виртуальных серверах в контейнерах и использование соответствующих инструментов, среди которых:
   1. Хранение исходного кода и конфигураций— Bitbucket, GitHub
   2. Непрерывная интеграция — Jenkins.
   3. Тестирование интегральное — Postman.
   4. Размещение в облаках — AWS (возможны и другие варианты, в зависимости от бюджета проекта).
6. **Безопасность**. Планируется использование концепции *Zero-trust*, в приложении к данной системе состоящей в следующем:

1. Все сервера и виртуальные машины/контейнеры системы находится под защитой firewall и посторонние запросы допускаются только на **модуль пользовательского доступа**.
2. Тем не мене каждый модуль рассматривает запросы, пришедшие к нему даже от других модулей системы, как небезопасные и осуществляет проверку авторизации.
3. Для проверки используются инструменты криптографической защиты. Реализацией являются токены, передаваемые с каждым запросом. Только валидность токена позволяет пропустить запрос.
4. Токены генерируется специальным менеджером безопасности внутри системы, находящимся в данный момент в **модуле безопасности.** Возможен переход на внешний, например KeyCloak.
5. Все внешние запросы возможны лишь к модулю пользовательского доступа. Они проходят стандартные барьеры защиты, соответствующие требованиям OWASP.
6. Внешний запрос к модулю пользовательского доступа проходит парольную аутентификацию. В случае успеха генерируется **токен**, который передаётся **в модуль интерфейса пользователя** — в клиент браузера.
7. Данный токен сберегается лишь в программе **модуля пользовательского интерфейса** и не сохраняется ни в cookies, ни в памяти браузера.
8. Данный токен имеет ограниченный срок жизни в течение которого клиент — программа интерфейса пользователя (из браузера) может вести диалог с серверными модулями, а модули передают этот токен по цепочке между собой. Каждый запрос к серверным модулям, естественно подписывается данным токеном. Токен передаётся в заголовке HTTP/HTTPS.
9. Генерацией данных токенов занят модулю пользовательского доступа с возможным объединением с внешними системами, как OAuth, sms-OTP.
10. Доступ к администрированию системных и виртуальных ресурсов осуществляется через стандартные Linux терминалы через ssh. Доступ защищён паролями, криптопарами ключей, а также ограничение множества IP адресов , с которых может входить администратор.
11. Все пароли будут валидироваться на соответствие требованиям устойчивости до начала применения.

12. Внутренние секреты шифрования, необходимые модулям безопасности сохраняются в конфигурационных файлах с ограниченным доступом или в соответствующих закрытых хранилищах облачных сервисов.
13. Модуль внешнего доступа и модуль интерфейса пользователя использует протокол HTTPS.
14. Дополнительно используются защиты от подбора паролей, токенов и DDoS атак.
15. *Используемые инструменты и технологии: Spring Security, KeyCloack, OAuth.*

7. **Соотвествие соверменным подходам** обеспечивается использованием следующих инструментов:
   1. Программирование Java 17, Sprong Boot 2.3, Python3.9, Scikitlearn.
   2. Контейнеризация Doker, Kubernetes.
   3. Тестирование JUnit, Spring Boot, Postman.
   4. Непрерывная интеграция Jenkins.
   5. Хранение версий, кода и конфигураций Bitbuket, Git? GitHub.
   6. Командная работа Jira, Confluence.
   7. Базы данных Postgres 12, Redis, Mongo.
   8. Нагрузка, прокси и кеширование Nginx, Redis, Memcash.
   9. Мониторинг Prometheus, Graphana, Logstech, Kibana.
   10. Безопасность JWT, KeyClock, OAuth, Spring Security, ssh.
   11. Асинхронность Kafka.
   12. Интерфейс пользователи и визуализация данных, включая BI — Vue2.js, Devextreme.

Замечание: Все перечисленные продукты являются свободными для некоммерческого использования. Однако, при переходе системы в режим платного доступа необходимо будет определятся с соответствующими лицензиями и нести затраты на их получение.

Выполненная программная реализация

На данный момент разработаны следующие модули полностью или частично. Ниже приведено описание работающего функционала.

1. Модуль пользовательского доступа
Функционирует, как Spring Boot, Java приложение, реализующее REST API (документация прилагается). Данное приложение пакуется в один файл формата JAR, развёртывается на встроенном сервере Tomcat. Запуск осуществляется в отдельном Docker контейнере, Java версии 17. Конфигурации Java и контейнера (ресурсы вычисления и памяти, сборка мусора и тп.) будут оптимизироваться в условиях близких к производству (production). Пока остаются дефолтными.

Модуль представляет собой трёх-слойную систему:

- доступа к базам данных, реализующего CRUD и поддерживающего транзакционность.
- бизнес логики обработки, валидации данных
- получения/предачи данных сериализованных в формате JSON в строну к/от клиента – модуля интерфейса пользователя.

Модуль **пользовательского доступа** предоставляет незакрытый доступ к:

- формам входа пользователя.
- регистрации новых пользователей.
- смены пароля.

Остальной вход только для пользователей, представивших в своём REST запросе валидный токен, полученный из модуля безопасности. Данным модулем кешируются ответы не запросы пользователей. Модуль предоставляет доступ в зависимости от уровня доступа:

- пользователю доступна наименьшая часть функций.
- исследователю — пользовательский, плюс возможности конфигурации моделей сбора данных и прогнозирования.
- администратору — исследовательские, плюс возможность управлять правами пользователей и осущствлять некоторые изменения настроек системы.

Запрос данных за пределами авторизации приводит к возврату HTTP ошибки *401 — Unauthorized.*

Для увеличения производительности данный модуль помещён за фасадом Nginx, который работает в качестве прокси реверсного сервера и осуществляет:

- переход в HTTPS
- кеширование статического контента
- балансировку нагрузки.

Модуль непосредственно общается с базой данных PostgreSQL для запроса данных, но может делать это и через посредничество модуля доступа к данным.
На данный момент разработана вся затребованная часть функционала (с учётом отставания в разработке модулей прогнозирования). Модуль покрыт тестированием на 100%. Тестирование подтверждает корректность работы.

2. Модуль безопасности

В основе модуля безопасности лежит использование **самодостаточных токенов JWT**. Самодостаточность говорит о том, что токены не хранятся в базе данных как пароли, а проверяются на ходу с помощью криптографии. Это позволяет использовать их сквозным образом во всей системе и снизить нагрузку на базу данных. Это также позволяет избежать сессий со статусом и соответствующих рисков похищения данных сессии, данных coocies и тп.
Токены содержат информацию об пользовательских правах, IP адресе отправителя и времени истекания полномочий токена (на этапе тестирования установлены одни сутки).
При входе пользователя и его парольной авторизации модуль безопасности генерирует токен, подписывает его криптографически и возвращает клиенту, в нашем случае модулю интерфейса пользователя, то есть клиенту в браузере. Для каждого запроса к модулю пользовательского доступа (REST запрос) модуль интерфейса должен передать в заголовке запроса данный токен. Если токен не валиден, просрочен или скомпрометировано IP, модуль вернет ошибку *401 — Unauthorized.*
Если пользователь разлогинился (вышел из системы), то его токен будет сохранен в блок -исте на срок дейсвия токена. Для блок листа используется тот же инструмент что и для высокопроизводительного кеширования, в нашем случае Redis. Кроме этого модуль безопасности отслеживает частоту запросов от одного IP, блокируя попытки подбора также внося IP такого пользователя в блок лист.

Помимо этого модуль безопасности предоставляет пользователю возможность смены забытого пароля, отправляя ему письмо со ссылкой, содержащей токен, перейдя по которой пользователь сможет изменить свой пароль.
Новый пользователь зарегистрированный в системе получает письмо по электронной почте, также содержащее ссылку с токеном, переход по которой позволяет подтвердить электронный адрес. После подтверждения вход пользователя в систему разрешён. Но его полномочия могут устанавливаться администратором в ручную или автоматически, допустим по факту оплаты за услуги системы (автоматизация данных бухгалтерии).
Хранение паролей и прочих данных пользователя осуществляется в базе данных PostgreSQL.
Модуль безопасности и модуль пользовательского доступа разворачиваются совместно в четырёх контейнерах:

- nginx
- java
- redis
- postgres.

Функционал модуля разработан полностью и покрыт тестами на 100%. Тестирование подтверждает корректность работы.

3. Модуль интерфейса пользователя

Модуль интерфейса пользователя включает:

- формы входа, выхода, смены пароля, регистрации нового пользователя.
- стандартное меню навигации сайта в виде расположенной слева «выдвижной» панели инструментов.
- экранных форм работы с данными, в частности, для администратора — управления уровнем доступа пользователей.
- дашбордов в стиле **Business Intelligence** для интеллектуальной визуализации данных и прогнозов.

Система полностью интерактивная, анимированная, разработана в стиле материального дизайна с использованием современных библиотек клиентского интерфейса Vue2.js, DevExtrem.

Модуль является асинхронным приложением, позволяющим получать данные в фоновом режиме, выполняя запросы или автоматические функции без задержек для пользователя.
Как было сказано выше, данный модуль запрашивает и получает в обмен на пароль и логин пользователя токе и хранит его в недоступном для похищения виде для подписания своих запросов к серверной части системы.
Хранение распределенных рабочих данных (для редактирования, отражения в виде графиков и таблиц различными частями системы) осуществляется в транзакционном хранилище, реализованном библиотекой Vuex.
Передача REST запросов — через библиотеку Anxios.
Внешний вид интерфейса адаптируется в зависимости от полномочий пользователя, предоставляя все функции меню и экранные формы только администратору, остальным пользователям ограничено.
Модуль сам обрабатывает ошибки сети и полученные коды ошибок от сервера преобразуя их в всплывающие экранные информаторы.
Интернационализация интерфейса осуществляется именно данным модулем в зависимости от используемой локали.
Интерфейс оптимизирован под любой размер монитора и мобильные устройства.

На данный момент разработана вся затребованная часть функционала (с учётом отставания в разработке модулей прогнозирования). Модуль покрыт тестированием на 100%. Тестирование подтверждает корректность работы.

3. Инструменты командной работы, написания кода и тестирования

Для работы над созданием, тестирование и запуска в производство системы осуществлены:

- зарезервирована электронная почта для проекта.
- создан репозиторий Bitbuket.
- создан проект рабочей документации Confluence.
- активизирована и используется система управления разработкой Jira и задачами Trello.
- приобретена лицензия для Java разработки на IntelliJ IDEA.
- создана среда контейнерной разработки Docker.
- настроена среда тестирования Postman.

4. Доступны исходные файлы и скомпилированный продукт вместе со скриптом его развёртывания

- ссылка скачивания: «https://bitbucket.org/denwolpertinger/backend/src/master/»
- предпочтительно скачать командой *git clone <ссылка>*
- скрипт развёртывания *to_run_site*

Минимальные требования к системе для изучения и тестирования:

- ОС и приложения:
  - Linux или MacOS
  - Docker
  - Docer-compose
- Техника
  - >2G RAM
  - >10GB HDD
  - >Intel 3
- Клиентская часть
  - Современный браузер (версии от 2020)
- Вход в систему после запуска по скрипту
  - набрать в браузере IP сервера где было развёртывание
  - например http://127.0.0.1