Решение тестового задания Danone

Автор: Николай Буше  
nbushe@rambler.ru

Оглавление

[Введение 2](#_Toc190957813)

[Общие сведения 3](#_Toc190957814)

[Список сокращений 3](#_Toc190957815)

[Цель проекта 3](#_Toc190957816)

[Реализация 3](#_Toc190957817)

[Функциональные возможности 3](#_Toc190957818)

[Границы проекта 4](#_Toc190957819)

[Архитектурное решение 4](#_Toc190957820)

[Схема архитектуры 4](#_Toc190957821)

[Возможности 5](#_Toc190957822)

[Технологии 5](#_Toc190957823)

[Инструкции по установке и запуску 6](#_Toc190957824)

[1. Запуск с использованием Docker 6](#_Toc190957825)

[2. Локальный запуск без Docker 6](#_Toc190957826)

[Требования к техническому обеспечению 7](#_Toc190957827)

[Общие требования 7](#_Toc190957828)

[В случае запуска без Docker: 7](#_Toc190957829)

[Состав проекта и описание работы модулей 7](#_Toc190957830)

[Эмулятор (C#) 7](#_Toc190957831)

[API (ASP.NET Core) 7](#_Toc190957832)

[Фронтенд (React) 7](#_Toc190957833)

[Сервис базы данных 8](#_Toc190957834)

[Настройки 8](#_Toc190957835)

[Примечание 8](#_Toc190957836)

[Описание работы основных модулей 9](#_Toc190957837)

[Документация 13](#_Toc190957838)

# Введение

Данный документ описывает программный продукт созданный, как решение тестового задания Danone.

# Общие сведения

Программный продукт представляет собой систему мониторинга данных с виртуальных датчиков, реализованную с использованием архитектуры на основе 4 контейнеров Docker.

# Список сокращений

**MVP (Minimum Viable Product) —** Минимально жизнеспособный продукт – базовая версия продукта, содержащая только ключевые функции, необходимые для запуска и тестирования на рынке.

**API (Application Programming Interface) —** Интерфейс программирования приложений – набор правил и инструментов, позволяющих различным программным компонентам взаимодействовать друг с другом.

**Фронтенд (Frontend) —** Клиентская часть приложения, отвечающая за пользовательский интерфейс и взаимодействие с пользователем.

**Бэкенд (Backend) —** Серверная часть приложения, обеспечивающая обработку данных, бизнес-логику и взаимодействие с базой данных.

**БД (База данных) —** Организованная структура для хранения, управления и извлечения данных.

# Цель проекта

Создание комплексной системы мониторинга данных с виртуальных датчиков с возможностью:

* Эмуляции данных в реальном времени
* Хранения и анализа данных
* Визуализации через веб-интерфейс
* Интеграции с внешними системами через XML

# Реализация

Реализацией проекта является MVP микросервисного фуллстек приложения, совмещающего базу данных, бэкенд, фронтенд и эмулятор данных датчиков.  
Приложение развертывается на сервере с поддержкой docker compose.

# Функциональные возможности

* API:
  + Прием XML-файлов с последующей валидацией по XSD-схеме.
  + Прием данных от эмулятора датчиков.
  + Возврат данных за указанный период времени.
  + Агрегация данных (среднее, максимум, минимум).
* Эмуляция данных:
  + Генерация случайных значений от 0 до 100 для трех датчиков каждую секунду.
  + Отправка данных в базу через API.
* База данных:
  + Хранение данных, полученных от эмулятора датчиков.
  + Поддержка PostgreSQL/MSSQL.
* Фронтенд:
  + Отображение данных в виде таблицы с фильтрацией по времени.
  + Визуализация данных на графике с линейной интерполяцией.
  + Автоматическое обновление данных каждые 5 секунд.
  + Выгрузка данных в XML-файл и отправка его на валидацию.

# Границы проекта

При создании проекта очень важен вопрос о его границах.  
В связи с *ограничением времени на разработку* было решено создать MVP, имеющий следующие ограничения:

* Нет поддержки аутентификации пользователей
* Ограничение на 3 фиксированных датчика
* Нет механизма горизонтального масштабирования
* Только базовые метрики (среднее, мин/макс)
* Локализация только на русский язык
* Данные подключения к серверам включены в git-репозиторий
* Нет юнит-тестов в реализации. Тестирование проводится на уровне ручных интеграционных тестов через Swagger и DevTools фронтенда.

# Архитектурное решение

Система состоит из следующих компонентов:

* **Сервис эмуляции данных (эмулятор)** : Генерирует данные с виртуальных датчиков.
* **База данных**: Хранит все собранные данные.
* **API**: Предоставляет интерфейсы для работы с данными.
* **Фронтенд**: Веб-интерфейс для пользователей.

Каждый компонент развертывается в отдельном Docker-контейнере.

### Схема архитектуры

[Эмулятор] → [API] ↔ [PostgreSQL]

↓↑

[Frontend]

4-компонентная Docker-система:

Сервис эмуляции данных (C#)

API (ASP.NET Core/C#)

База данных (PostgreSQL)

Фронтенд (React/TypeScript)

# Технологии

* **Бэкенд**:
  + ASP.NET Core 8
  + Entity Framework Core
  + Npgsql
* Фронтенд:
  + React 19
  + Redux Toolkit
  + Material-UI
  + Recharts
* **БД**:
  + PostgreSQL
* **Инфраструктура**:
  + Docker
  + Docker Compose
* **Вспомогательные**:
  + Swagger
  + XSD-валидация

# Состав проекта и описание работы модулей

## Эмулятор (C#)

Генерирует случайные значения датчиков, в диапазоне от 0 до 100 и отправляет на бэкенд с интервалом 1 с.

## API (ASP.NET Core)

Осуществляет взаимодействие с базой данных:

Принимает данные датчиков и сохраняет их в базе данных (одна запись на показание каждого датчика)

Выдает по запросу данные о показаниях датчиков за указанный период, группируя их по времени.

Возвращает агрегированные данные о показаниях датчиков (среднее, максимум, минимум) за указанный период.

Принимает XML-файл, валидирует его по XSD-схеме и сохраняет его в базе данных.

## Фронтенд (React)

Позволяет выбрать желаемый диапазон времени.

Отображает данные в виде таблицы с фильтром по диапазону времени.

Отображает данных на графике (временной график с линейной интерполяцией).

Отображает агрегированные данные за указанный период.

Выгружает отображенные значения в виде XML на сервер.  
Кеширует данные с сервера и загружает их порциями для ускорения работы.

## Сервис базы данных

База данных Postgres, для накопления данных.

## Настройки

|  |  |
| --- | --- |
| **Файл** | **Описание** |
| `docker-compose.yml` | Оркестрация контейнеров. Определяет процесс запуска приложения. |
| `init.sql` | Инициализация БД. Создание пользователя. |
| `.env` | Конфигурация окружения эмулятора – управляет логированием |
| `schema.xml` | XSD-схема валидации |

## Примечание

Для простоты в данном тестовом задании все данные доступа к ресурсам включены в проект. В реальной системе они должны быть вынесены в отдельные .env файлы и передаваться лично ответственным лицам при развертывании системы.

# Описание работы основных модулей

|  | **Файл** | **Описание** |
| --- | --- | --- |
| **API** | | |
| 1 | Controllers/SensorController.cs | Контроллер для обработки запросов к API, связанных с данными датчиков. |
| 2 | Controllers/XmlController.cs | Контроллер для обработки запросов к API, связанных с XML-данными. |
| 3 | Model/Sensor.cs | Модель данных для хранения информации о датчиках. |
| 4 | Model/SensorData.cs | Модель данных для хранения данных датчиков. |
| 5 | Model/XmlData.cs | Модель данных для хранения XML-данных. |
| 6 | Model/SensorsContext.cs | Контекст базы данных для работы с данными датчиков. |
| 7 | Components/SensorDataRepository.cs | Репозиторий для доступа к данным датчиков из базы данных. |
| 8 | Components/XmlDataRepository.cs | Репозиторий для доступа к XML-данным из базы данных. |
| 9 | Components/XmlValidator.cs | Валидатор для проверки XML-данных по заданной схеме. |
| 10 | Components/CustomLogger.cs | Логгер для логирования событий и ошибок в приложении. |
| 11 | appsettings.json | Файл конфигурации приложения. |
| 12 | appsettings.Development.json | Файл конфигурации приложения для режима разработки. |
| 13 | Program.cs | Основной файл приложения. |
| 14 | Startup.cs | Файл инициализации приложения. |
| 15 | out/appsettings.json | Файл конфигурации приложения. |
| 16 | out/appsettings.Development.json | Файл конфигурации приложения для режима разработки. |
| 17 | out/api.deps.json | Файл зависимостей приложения. |
| 18 | out/api.runtimeconfig.json | Файл конфигурации runtime приложения. |
| **База данных** | | |
| 1 | init.sql | Файл инициализации базы данных. |
| 2 | Dockerfile | Файл конфигурации Docker для базы данных. |
| 3 | data | Каталог для хранения данных базы данных. |
| **Эмулятор** | | |
| 1 | SensorEmulator.cs | Класс для эмуляции данных датчиков. |
| 2 | SensorData.cs | Модель данных для хранения данных датчиков. |
| 3 | LoggingHandler.cs | Класс для логирования запросов и ответов в API. |
| 4 | CustomConsoleFormatter.cs | Класс для форматирования логов в консоли. |
| 5 | Exceptions.cs | Класс для обработки исключений в приложении. |
| 6 | appsettings.json | Файл конфигурации приложения. |
| 7 | Program.cs | Основной файл приложения. |
| 8 | .env | Файл конфигурации окружения. |
| **Фронтенд** | | |
| 1 | src/components/DataTable.tsx | Компонент для отображения данных датчиков в таблице. |
| 2 | src/components/LineChart.tsx | Компонент для отображения данных датчиков на графике. |
| 3 | src/components/Summary.tsx | Компонент для отображения суммарных данных датчиков. |
| 4 | src/components/UploadForm.tsx | Компонент для загрузки XML-файлов на сервер. |
| 5 | src/components/NoData.tsx | Компонент для отображения сообщения о отсутствии данных. |
| 6 | src/components/Sidebar.tsx | Компонент для отображения боковой панели. |
| 7 | src/components/IntervalPicker.tsx | Компонент для выбора интервала времени. |
| 8 | src/components/AppLayout.tsx | Компонент для отображения основной структуры приложения. |
| 9 | src/App.tsx | Основной компонент приложения. |
| 10 | src/index.tsx | Файл инициализации приложения. |
| 11 | src/main.tsx | Основной файл приложения. |
| 12 | src/services/api.ts | Файл для работы с API. |
| 13 | src/misc/SensorData.ts | Модель данных для хранения данных датчиков. |
| 14 | src/misc/maxSensors.ts | Функция для определения максимального количества датчиков. |
| 15 | src/misc/webColors.ts | Массив цветов для использования в приложении. |
| 16 | src/redux/intervalSlice.ts | Файл для работы с состоянием интервала времени. |
| 17 | src/redux/store.ts | Файл для работы с хранилищем состояния. |
| 18 | public/index.html | Основной файл HTML приложения. |
| **Конфигурации** | | |
| global.json |  | Файл конфигурации global. |
| docker-compose.yml |  | Файл конфигурации Docker Compose. |
| README.MD |  | Файл README. |
| TZ.MD |  | Файл технического задания. |
| api/schema.xml |  | Файл схемы XML. |

# Инструкции по установке и запуску

## 1. Запуск с использованием Docker

**mkdir test**

**cd test  
git clone. <адрес проекта> .**  
<установка docker compose>

**# Сборка и запуск проекта**

**docker compose up --build**

## 2. Локальный запуск без Docker

**mkdir test**

**cd test  
git clone. <адрес проекта> .**

<установка .net SDK>  
<установка nodejs>

<установка yarn>

**cd database**

<в отдельном терминале:>

**cd test/api**

<в файле конфигурации проекта appsettings.json изменить строку:

"DefaultConnection":"Host=db;Port=5432;Database=sensor\_data;…

на

"DefaultConnection":"Host=localhost;Port=5432;Database=sensor\_data;…

>

**dotnet run**

<в отдельном терминале:>

**cd test/emulator**

**dotnet run**

<в отдельном терминале:>

**cd test/frontend**

**corepack enable**

**yarn set version berry**

**yarn install**

**yarn dev**

# Требования к техническому обеспечению

## Общие требования

* Операционная система: Linux/Windows/macOS.
* Docker и Docker Compose для развертывания контейнеров.

## В случае запуска без Docker:

* Node.js и yarn для работы с фронтендом.
* .NET SDK для работы с API.
* PostgreSQL для базы данных.

# Документация

В комплекте с системой поставляется следующая документация:

* TZ.MD Техническое (тестовое) задание на систему
* README.md в репозитории проекта.
* Описание системы (данный документ)