|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ**\_\_\_\_***ИУК «Информатика и управление»*\_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_

**КАФЕДРА** **\_\_\_\_*ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***\_\_\_\_\_\_

**О Т Ч Е Т**

**УЧЕБНАЯ ПРАКТИКА**

**«Проектно-технологическая практика»**

Студент гр. ИУК4-42Б \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Карельский М.К.)

(подпись) (Ф.И.О.)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Пчелинцева Н.И.)

(подпись) (Ф.И.О.)

Оценка руководителя \_\_\_\_\_ баллов \_\_\_\_\_\_\_\_\_\_\_

30-50 (дата)

Оценка защиты \_\_\_\_\_ баллов \_\_\_\_\_\_\_\_\_\_\_

30-50 (дата)

Оценка практики \_\_\_\_\_ баллов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка по пятибалльной шкале)

Комиссия: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (         Гагарин Ю.Е.       )

(подпись) (Ф.И.О.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (          Глебов С.А.        )

(подпись) (Ф.И.О.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (     Пчелинцева Н.И.    )

(подпись) (Ф.И.О.)

*Калуга, 2022Калужский филиал федерального государственного бюджетного образовательного учреждения высшего образования*

***«Московский государственный технический университет имени Н.Э. Баумана*** *(национальный исследовательский университет)»* ***(КФ МГТУ им. Н.Э. Баумана)***

УТВЕРЖДАЮ

Заведующий кафедрой **\_\_ИУК4\_\_\_**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Гагарин Ю.Е.)

« 04 » июля 2022 г.

**ЗАДАНИЕ**

**на УЧЕБНУЮ ПРАКТИКУ, Проектно-технологическая практика**

За время прохождения практики студенту необходимо:

1. Ознакомиться со структурой инфокоммуникационной системы вуза, получить доступ к данной системе путем создания соответствующих учетных записей, рассмотреть принципы и особенности применения инфокоммуникационной системы на примере работы приемной комиссии вуза.
2. Ознакомиться со структурой Федеральной Информационной Системы, рассмотреть взаимодействие ФИС и инфокоммуникационной системы вуза.
3. Осуществить занесение информации в инфокоммуникационную систему вуза при подаче документов и зачислении студентов, осуществить перенос информации из инфокоммуникационной системы вуза в Федеральную Информационную Систему.
4. Разработать API веб-сервиса ведения истории инцидентов.
5. Подготовить отчет и защитить результаты практики.

Дата выдачи задания « 04 » июля 2022 г.

|  |  |  |
| --- | --- | --- |
| Руководитель практики |  | Пчелинцева Н.И. |
|  |  |  |
| Задание получил |  | Карельский М.К. |

**CОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc113111734)

[1. ОБЩИЕ СВЕДЕНИЯ 5](#_Toc113111735)

[1.1. Исследование предметной области задачи и постановка задачи 5](#_Toc113111736)

[1.2. Обоснование выбора средства реализации 7](#_Toc113111737)

[1.3. Актуальность решаемой проблемы и возможные области применения данной разработки 8](#_Toc113111738)

[1.4. Определение входных и выходных характеристик 8](#_Toc113111739)

[2. ПРОЕКТИРОВАНИЕ КОМПОНЕНТОВ ПРОГРАММНОГО 9](#_Toc113111740)

[ПРОДУКТА 9](#_Toc113111741)

[2.1. Общие сведения о программе 9](#_Toc113111742)

[2.2. Проектирование иерархии классов 12](#_Toc113111743)

[2.3. Описание программных модулей 13](#_Toc113111744)

[2.4. Обработка исключений 14](#_Toc113111745)

[2.5. Обзор использованных инструментов 15](#_Toc113111746)

[ЗАКЛЮЧЕНИЕ 19](#_Toc113111747)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20](#_Toc113111748)

[Основная литература 20](#_Toc113111749)

[Дополнительная литература 20](#_Toc113111750)

[ПРИЛОЖЕНИЯ 22](#_Toc113111751)

[Листинг 22](#_Toc113111752)

**ВВЕДЕНИЕ**

Целью учебной практики является реализация API веб-сервиса ведения истории инцидентов в рамках совместной работы над модулем оповещения о недоступности сервиса сайта.

Для достижения поставленной цели ставятся следующие задачи:

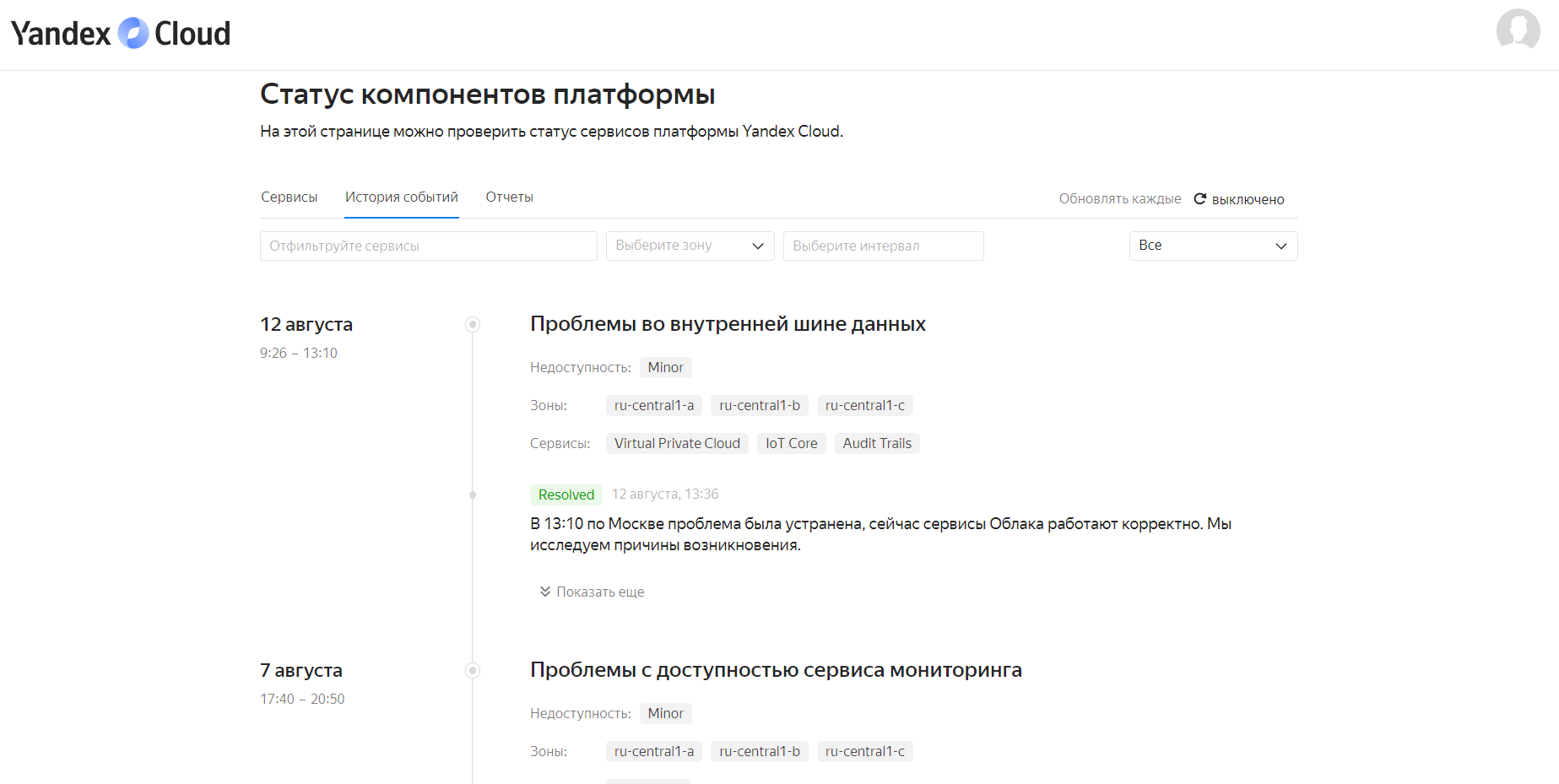
1. Проанализировать предметную область;
2. Создать и интегрировать репозиторий Git;
3. Интегрировать контейнеризатор приложений;
4. Разработать и реализовать необходимые модели;
5. Реализовать требуемый функционал;
6. Подключить базу данных.
7. **ОБЩИЕ СВЕДЕНИЯ**

# 1.1. Исследование предметной области задачи и постановка задачи

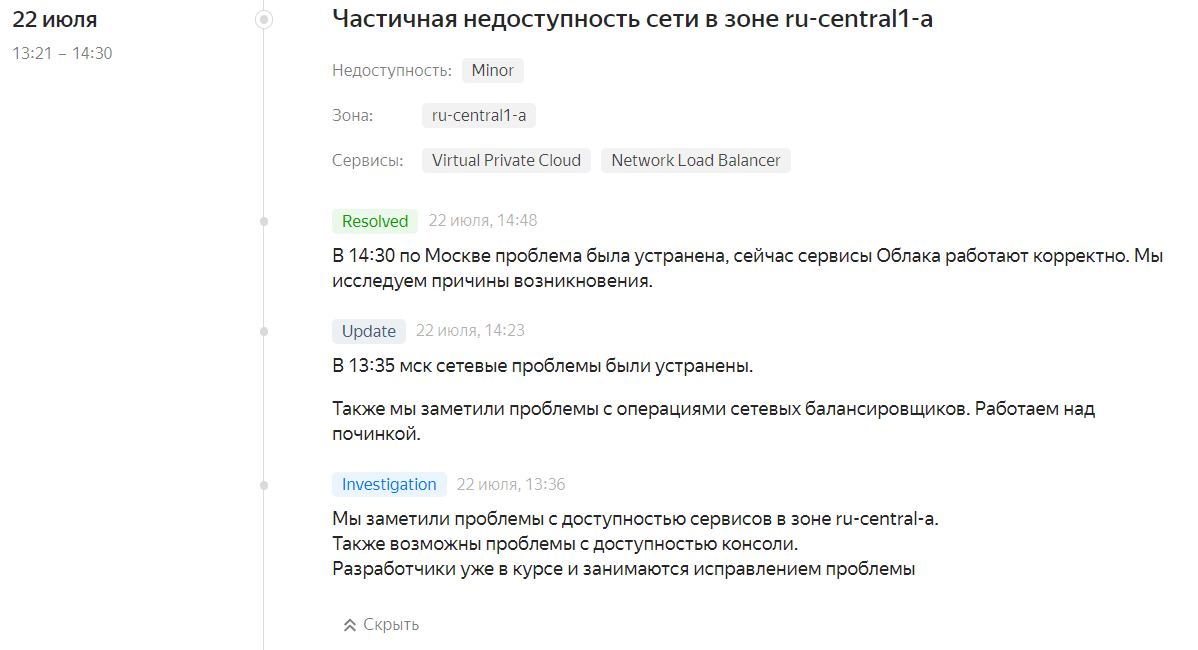
Сайт – это интернет-ресурс, состоящий из нескольких виртуальных страниц, которые связаны между собой ссылками и обычно объединены общей темой или задачей. Веб-сервис – идентифицируемая уникальным веб-адресом (URL-адресом) программная система со стандартизированными интерфейсами, единица модульности.

Периодически те или иные веб-сервисы могут оказываться временно недоступными. Причинами этому могут быть различные сбои или запланированные заранее профилактические работы. Таким образом, перед группой разработчиков ставится задача разработать 2 сервиса: ведения истории инцидентов и профилактических работ. В данной работе описывается решение первой задачи.

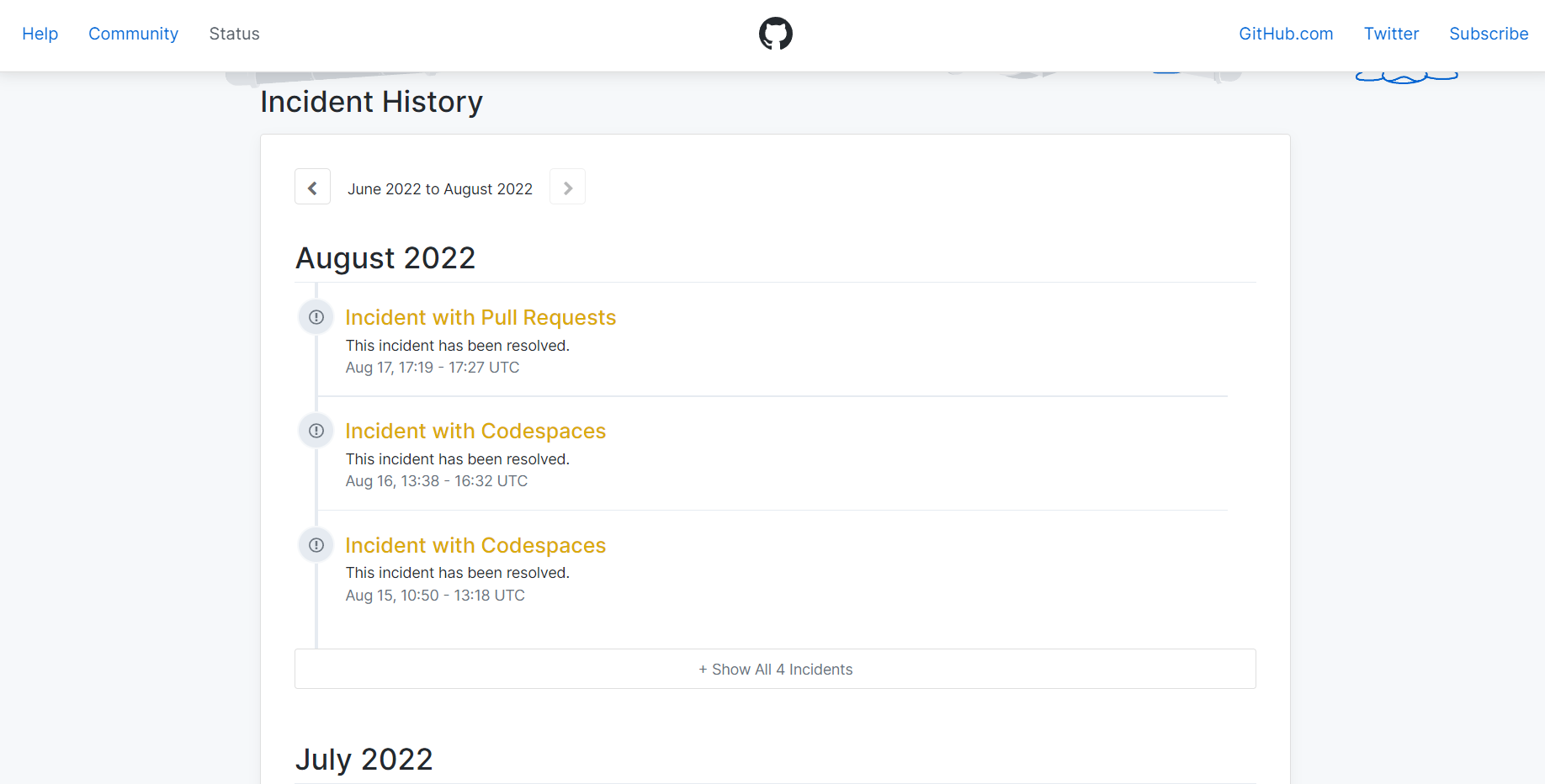
За пример были взяты два сервиса: Yandex Cloud Status (см. рис. 1.1-1.2) и GitHub Status (см. рис. 2.1-2.2).



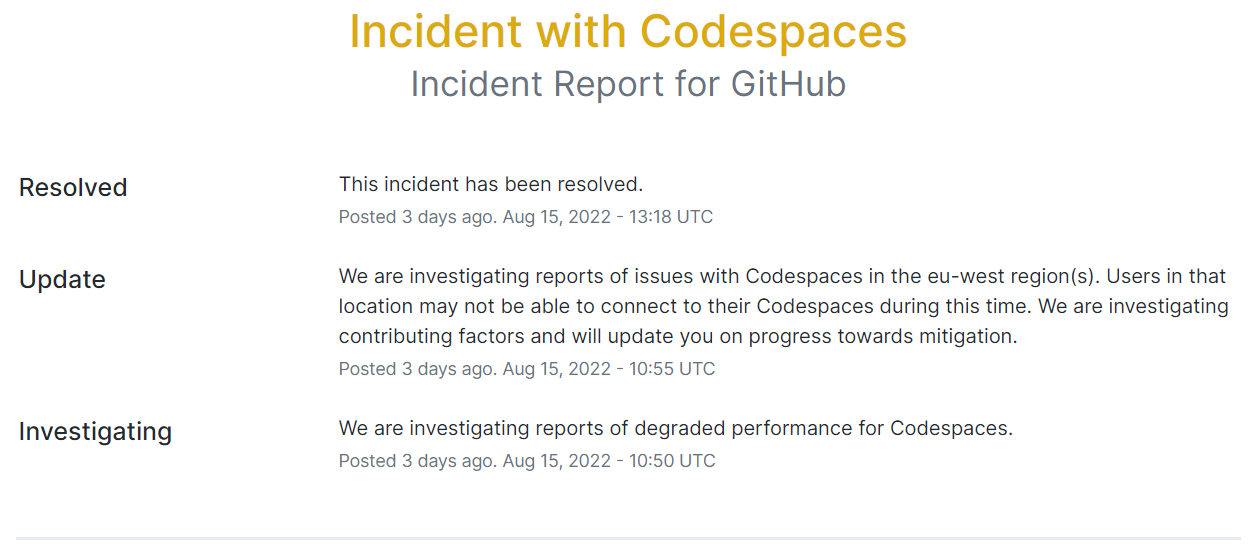
**Рис. 1.1** – История инцидентов Yandex Cloud



**Рис. 1.2** – Пример инцидента Yandex Cloud



**Рис. 2.1** – История инцидентов GitHub



**Рис. 2.2** – Пример инцидента GitHub

Таким образом, необходимо реализовать возможность создания списка инцидентов, имеющих заголовок, недоступные зоны и сервис, дополнительные теги, промежуток во времени, когда сервис был недоступен, а также включающие в себя отметки хода их решения, содержащие дату, комментарий и тег.

# 1.2. Обоснование выбора средства реализации

В качестве средства ведения репозитория Git был выбран GitHub, так как он удобен и наиболее распространен. Работа с ним предполагала использование commit convention.

Для реализации сервиса использовался фреймворк ASP.NET. Он достаточно популярен, стабильный и надежный. За язык программирования был взят широко известный и используемый C#. Средой разработки выступила Visual Studio 2022, ведь она имеет поддержку последней версии ASP.NET 6.0 и технологии IntelliSense.

Чтобы отделить приложение от инфраструктуры, требовалось использование платформы контейнеризации, в роли которой выступил Docker. Он имеет большое комьюнити, его контейнеры легковесны и производительны, в нем удобно обновлять версии продукта.

В качестве реляционной СУБД был выбран PostgreSQL, так как он доступный, имеет большое распространение, совместим с Docker и работает с Entity Framework Core.

Так как выполнение задачи не подразумевает работы над фронтэндом, для тестирования и демонстрации реализованного API использовался Swagger.

# 1.3. Актуальность решаемой проблемы и возможные области применения данной разработки

Разработанный сервис позволяет посетителям сайтов узнавать о недоступности каких-либо его частей, а также следить за ходом решения проблемы, тем самым получать представление о том, когда ресурс станет вновь доступным. Владельцы сайта же способны таким образом удобно поддерживать обратную связь с клиентами.Так как работа большинства сайтов сопряжена с неизбежным и непредсказуемым возникновением ошибок и проблем, созданный модуль способен найти свое применение на каждом из них.

# 1.4. Определение входных и выходных характеристик

Разрабатываемый продукт представляет собой сервис HTTP RESTful, позволяющий вести и просматривать историю инцидентов, которые привели к недоступности различных частей сайта.

Необходимо реализовать следующие методы:

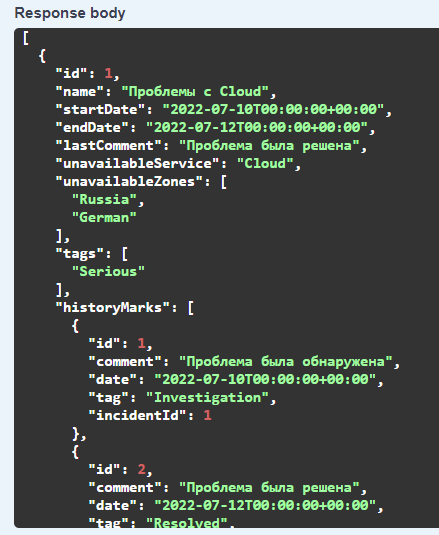
* Получение полного списка инцидентов вместе с их индивидуальной историей;
* Получение инцидента по ID с его историей;
* Добавление нового инцидента с указанием заголовка, недоступных зон и сервиса, тегов;
* Добавление отметки в историю взятого по ID инцидента с указанием комментария, даты и тега.

# ПРОЕКТИРОВАНИЕ КОМПОНЕНТОВ ПРОГРАММНОГО

# ПРОДУКТА

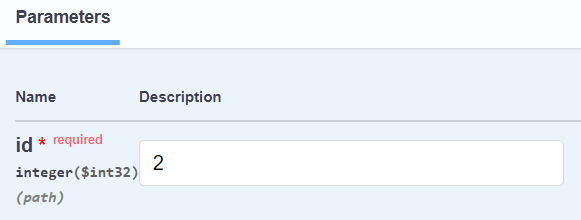
# 2.1. Общие сведения о программе

Программа представляет собой веб-сервис, имеющий 4 API-метода. Для их демонстрации используется Swagger.



**Рис. 3** – GET-метод без параметров

Первый из них – GET-метод без параметров (см. рис. 3). После его вызова возвращается JSON, содержащий список инцидентов, в каждый из которых включен свой список исторических отметок.

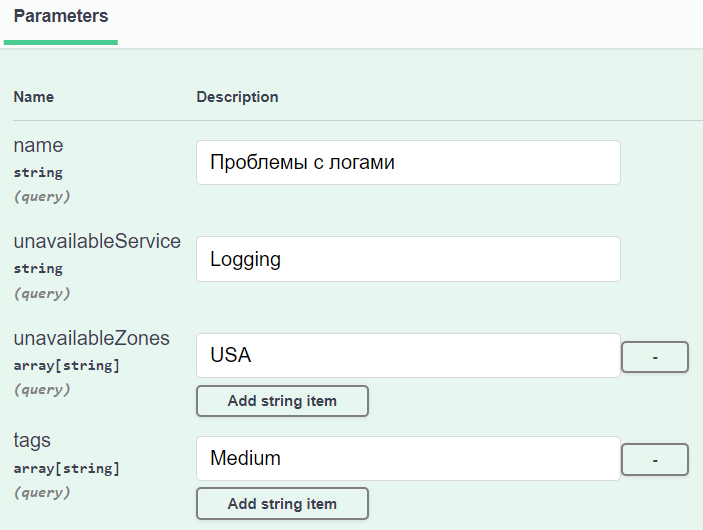


**Рис. 4.1** – Параметры второй версии GET-метода

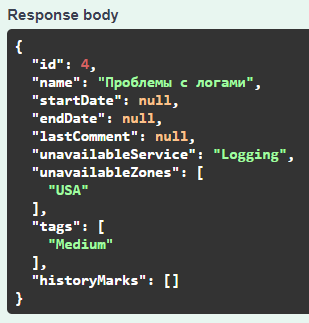


**Рис. 4.2** – Возвращаемые GET-методом с параметром данные

Вторая версия GET-метода (см. рис. 4.1-4.2) содержит один параметр – ID искомого инцидента. Также возвращает JSON с информацией о найденном инциденте, внутри которого указывается история его решения.



**Рис. 5.1** – Параметры POST-метода



**Рис. 5.2** – Возвращаемые POST-методом данные

POST-метод (см. рис. 5.1-5.2) отвечает за добавление новых инцидентов. Для его вызова необходимо указать 4 параметра: заголовок, недоступный сервис, список недоступных зон и список тегов. В итоге возвращается JSON с созданным инцидентом, который добавляется в БД.



**Рис. 6.1** – Параметры PUT-метода



**Рис. 6.2** – Возвращаемые PUT-методом данные

Последний PUT-метод (см. рис. 6.1-6.2) позволяет добавлять отметки в историю какого-либо инцидента. Для этого необходимо указать ID нужного инцидента, комментарий, дату отметки и тег. Результат аналогичный добавлению инцидента: возвращается JSON с созданной отметкой, которая добавляется в БД.

# 2.2. Проектирование иерархии классов

Проект состоит из 4 классов (см. рис. 7): предметные сущности инцидента (Incident) и отметки в его истории (HistoryMark), сеанс работы с базой данных (ApplicationContext) и основной контроллер (IncidentController).



**Рис. 7** – UML-диаграмма классов

# 2.3. Описание программных модулей

Модули сервиса имеют следующую структуру:

1. Класс Incident – сущность инцидента – события, которое привело к недоступности какой-то части сайта

* Свойство Id – уникальный идентификатор
* Свойство Name – заголовок, описывающий возникшую проблему
* Свойство StartDate – дата начала истории инцидента, возвращает дату самой старой отметки
* Свойство EndDate – дата текущего конца истории инцидента, возвращает дату самой новой отметки
* Свойство LastComment – последний комментарий из истории инцидента, работает аналогично EndDate: возвращает комментарий самой новой отметки
* Свойство UnavailableService – ставший недоступным вследствие инцидента сервис сайта
* Свойство UnabailableZones – ставшие недоступными вследствие инцидента зоны сайта
* Свойство Tags – теги, описывающие инцидент (например, уровень серьезности проблемы)
* Свойство HistoryMarks – история ведения инцидента, представляющая собой массив исторических отметок

1. Класс HistoryMark – сущность отметки в истории ведения инцидента

* Свойство Id – уникальный идентификатор
* Свойство Comment – комментарий о том, что было сделано в ходе решения возникшей проблемы
* Свойство Date – дата отметки
* Свойство Tag – тег, описывающий отметку (например, указание на обнаружение проблемы – investigation)

1. Класс ApplicationContext – сеанс работы с базой данных, позволяет изменять записи вышеописанных сущностей в БД

* Свойство Incidents – набор инцидентов, сопоставляемый с соответствующей таблицей в БД
* Свойство HistoryMarks – набор исторических отметок, сопоставляемый с соответствующей таблицей в БД
* Метод OnConfiguring(optionsBuilder) – настраивает базу данных в соответствии с указаниями во внешнем файле и подключает логирование

1. Класс IncidentController – основной контроллер, управляющий входящими запросами

* Поле \_cultureInfo – используется для корректной работы с датой
* Метод Get() – возвращает список инцидентов вместе с индивидуальной историей их ведения
* Метод Get(id) – возвращает инцидент с указанным Id вместе с его историей
* Метод Post(name, unavailableService, unavailableZones, tags) – вносит новый инцидент в БД
* Метод Put(incidentId, comment, date, tag) – вносит новую отметку истории инцидента в БД

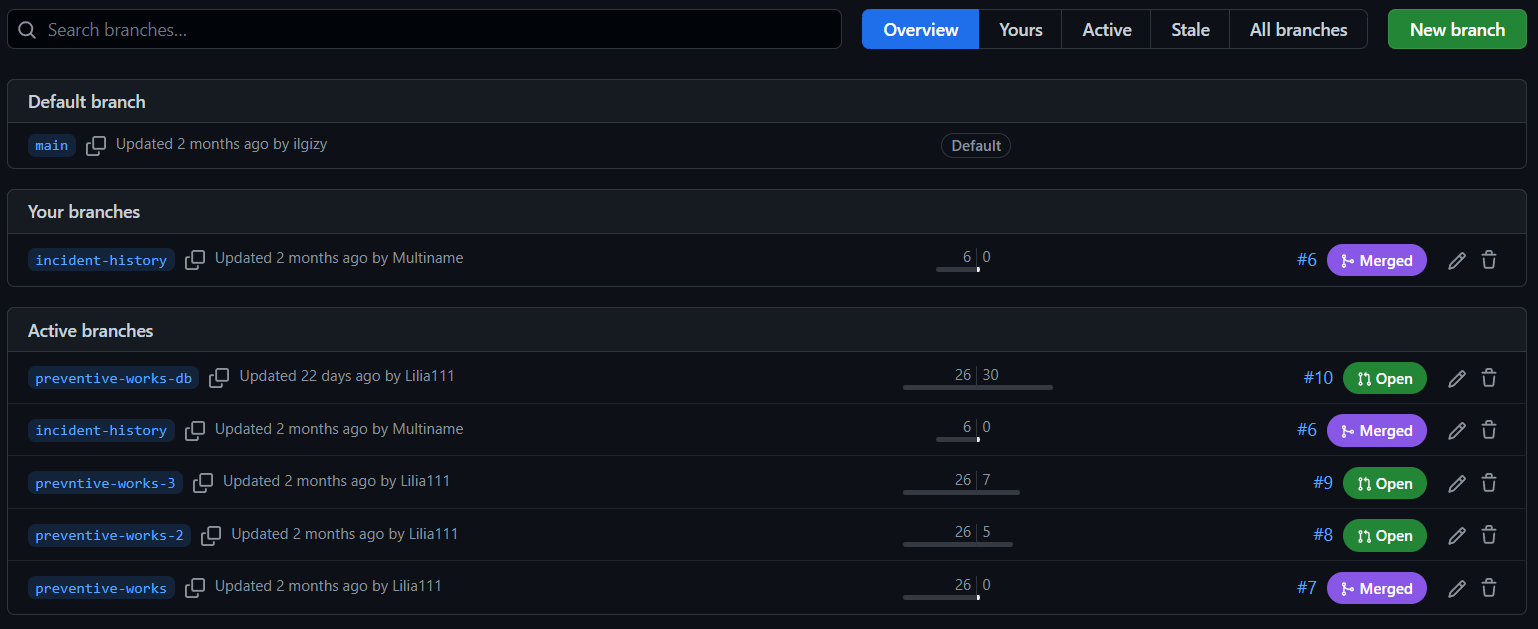
# 2.4. Обработка исключений

Для указания статуса результата запроса используются специальные коды:

* 200 – запрос прошел успешно (вместе с этим возвращается соответствующий запросу JSON);
* 400 – возвращается PUT-методом при попытке добавления исторической отметки с указанием несуществующего ID инцидента;
* 404 – возвращается GET-методом, если не удается найти инцидент с указанным ID.

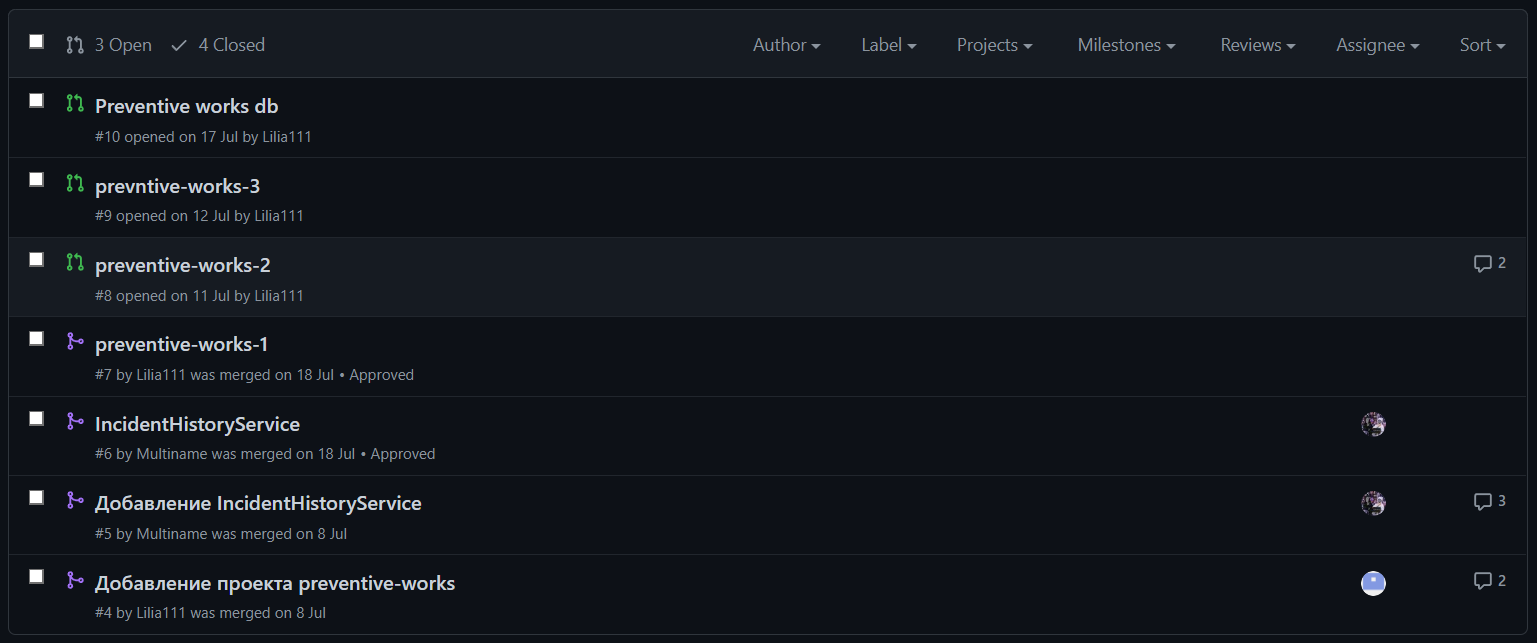
# 2.5. Обзор использованных инструментов

Реализация продукта подразумевала совместное использование общего репозитория GitHub при условии работы над разными компонентами. Вследствие этого была очевидна необходимость использования индивидуальных веток, стилем наименования которых был выбран kebab case (см. рис. 8).

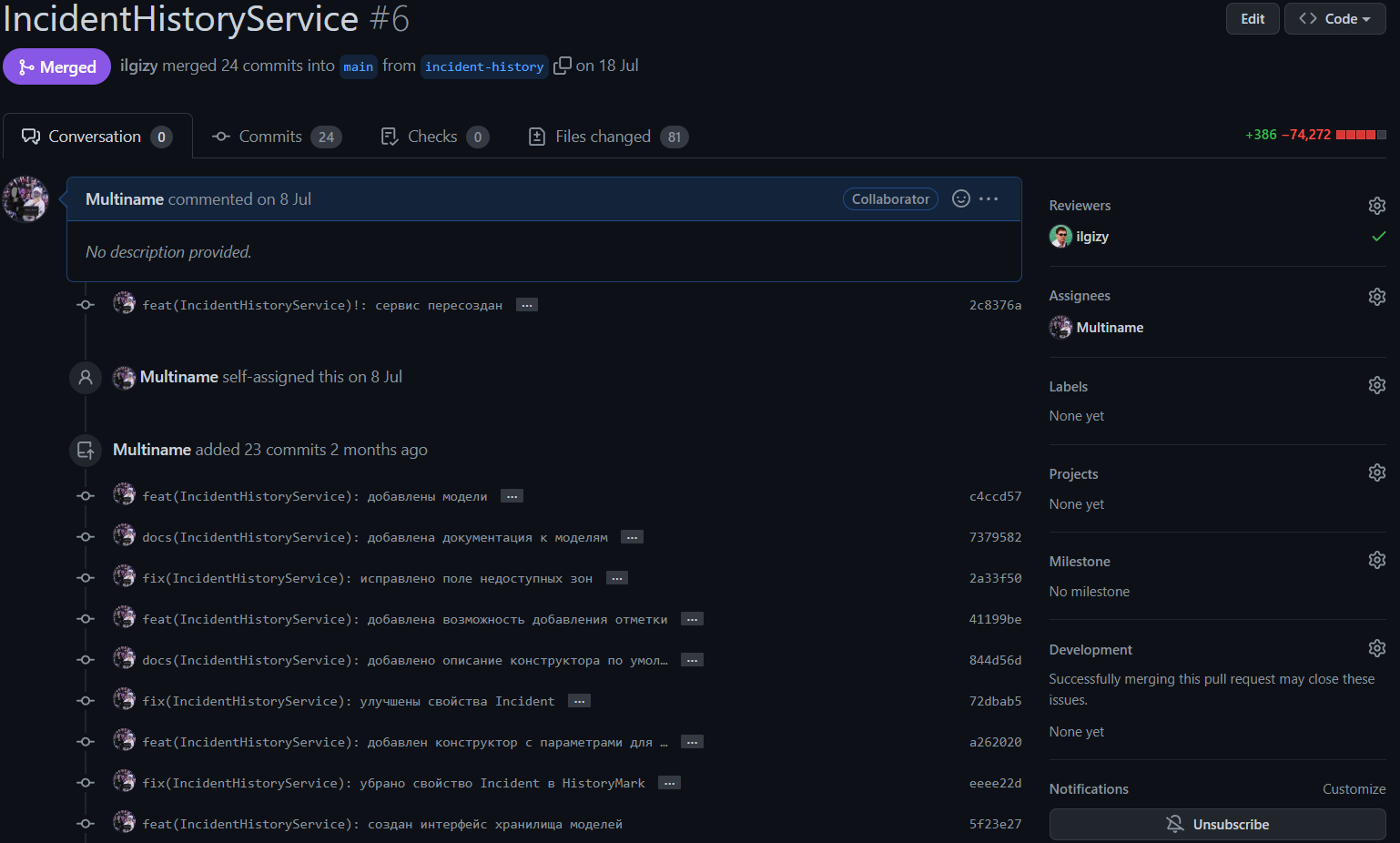


**Рис. 8** – Использовавшиеся ветки GitHub

Для проведения ревью проделанной работы использовалось открытие pull request (см. рис. 9.1-9.2).



**Рис. 9.1** – Pull requests

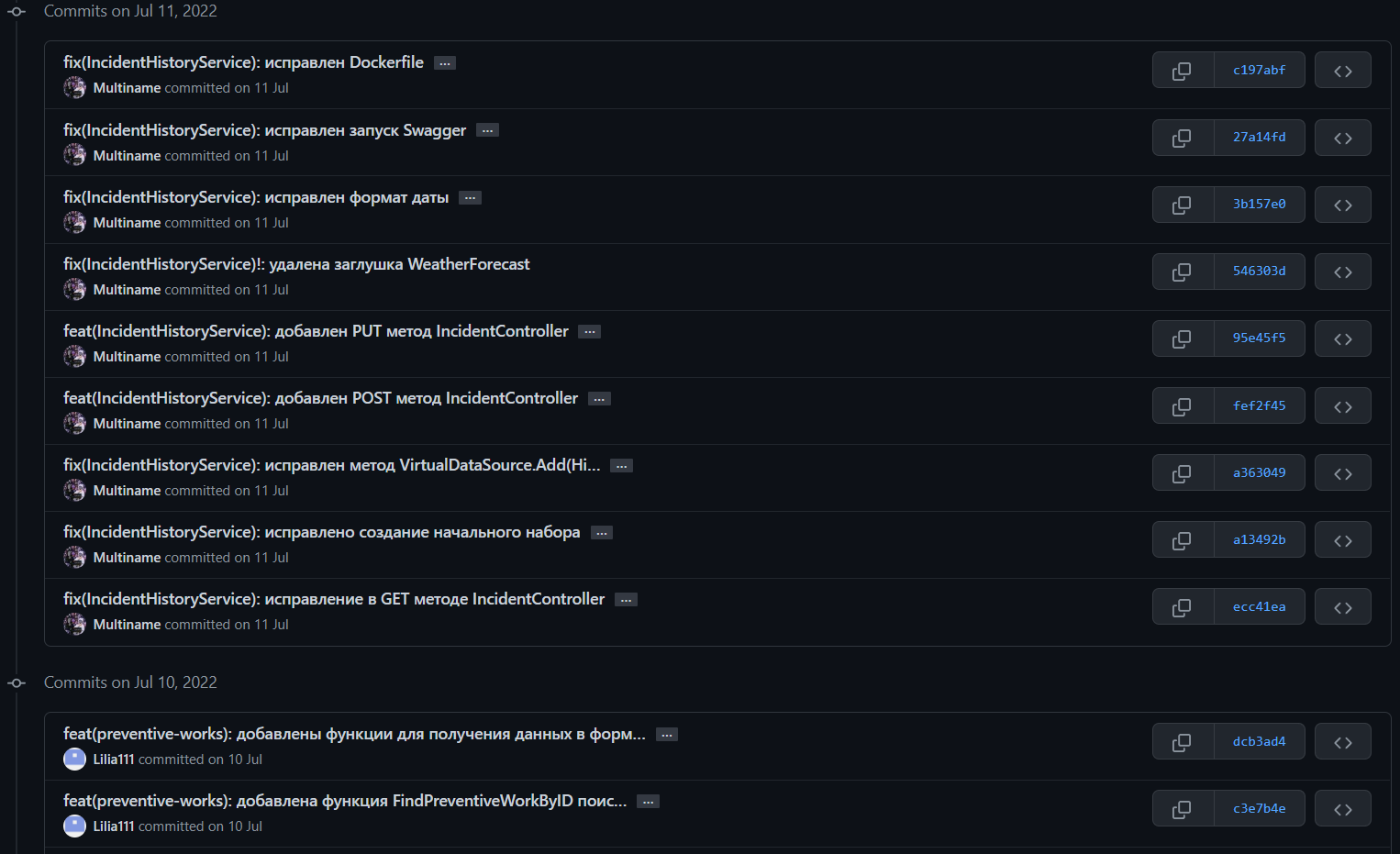


**Рис. 9.2** – Пример содержания pull request

Написание коммитов также происходило в рамках commit convention (см. рис. 10), предлагающей использование следующей формы: *<тип>(компонент): краткое описание*. Возможные типы:

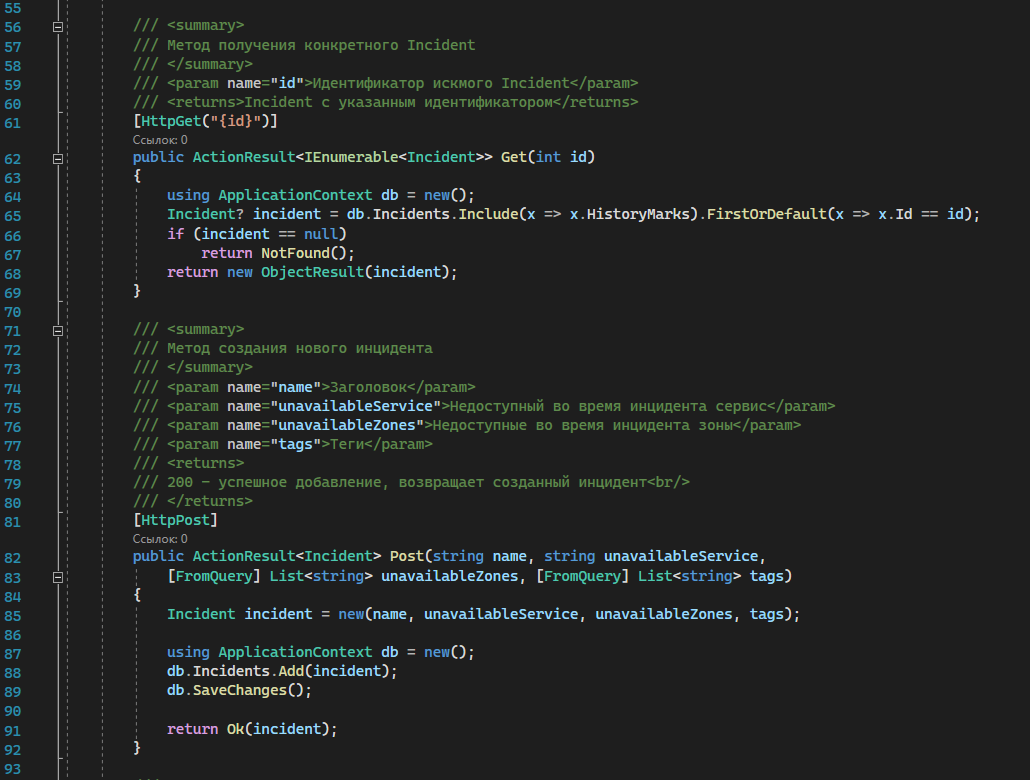
* feat – расширение, добавление новых возможностей;
* fix – различные исправления;
* conf – работа над конфигурациями;
* docs – работа с документацией.

При возникновении изменений, приводящих к несовместимости нового API со старым, после названия компонента ставится восклицательный знак.



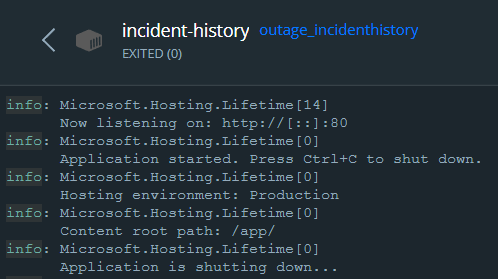
**Рис. 10** – Пример отправленных коммитов

Для удобной документации кода в Visual Studio 2022 использовались summary-комментарии (см. рис. 11).



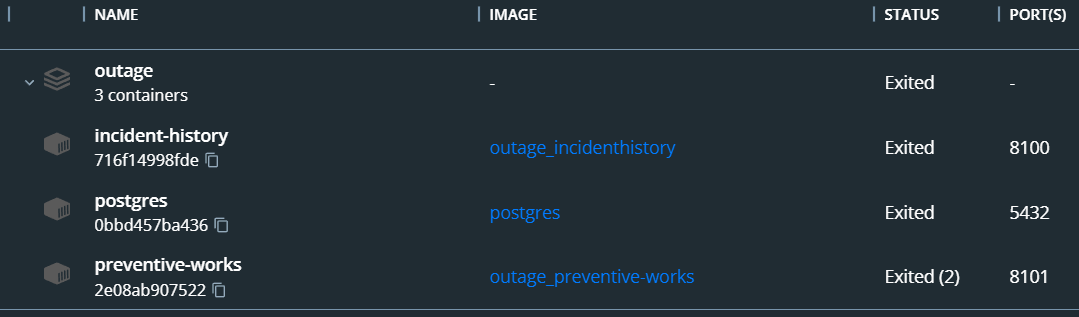
**Рис. 11** – Пример использования summary-комментариев

При настройке dockerfile сервиса ведения инцидентов был установлен порт 80 (см. рис. 12).



**Рис. 12** – Внутренний порт сервиса ведения инцидентов

В итоговом контейнере присутствовали образы двух требуемых сервисов – ведения инцидентов с портом 8100 и ведения профилактических работ с портом 8101, – а также необходимый образ PostgreSQL с портом 5432 (см. рис. 13).



**Рис. 13** – Итоговый контейнер Docker

**ЗАКЛЮЧЕНИЕ**

В ходе работы над проектом был успешно реализован веб-сервис ведения истории инцидентов, а также получены следующие навыки:

* Работа с C# ASP.NET 6.0;
* Использование summary-комментариев для документации кода;
* Совместная работа в общем репозитории GitHub: создание собственных веток, использование commit convention для их наименования, открытие pull request;
* Написание и настройка docker- и docker-compose файлов, создание контейнеров и образов Docker;
* Работа с PostgreSQL, подключение его в контейнер Docker;
* Интеграция и использование Swagger для удобной работы с запросами.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

## ***Основная литература***

1. Вайнштейн, М. З. Основы научных исследований [Электронный ресурс: учебное пособие / М. З. Вайнштейн, В. М. Вайнштейн, О. В. Кононова. — Йошкар-Ола: Марийский государственный технический университет, Поволжский государственный технологический университет, ЭБС АСВ, 2011. — 216 c. — Режим доступа: <http://www.iprbookshop.ru/22586.html>
2. Гутман Г.Н. Объектно-реляционная СУБД PostgreSQL: Учебное пособие. – Самара: АСИ СамГТУ, 2016
3. Коваленко, Ю.В. Информационно-поисковые системы [Электронный ресурс]: учебно-методическое пособие / Ю.В. Коваленко, Т.А. Сергиенко. — Омск: Омская юридическая академия, 2017. — 38 c.— Режим доступа: <http://www.iprbookshop.ru/66817.html>
4. Лок Э. ASP.NET Core в действии. – М.: ДМК Пресс, 2021
5. Маюрникова, Л. А. Основы научных исследований в научно-технической сфере [Электронный ресурс]: учебно-методическое пособие / Л. А. Маюрникова, С. В. Новосёлов. — Кемерово: Кемеровский технологический институт пищевой промышленности, 2009. — 123 c. — Режим доступа: <http://www.iprbookshop.ru/14381.html>
6. Мокий, М.С. Методология научных исследований [Текст]: учебник / М.С. Мокий, А.Л. Никифоров, В.С. Мокий. - М.: Юрайт, 2015. - 255 с.
7. Моделирование информационных ресурсов [Электронный ресурс]: учебно-методический/ Составитель Огнев Э.Н. - Кемерово: Кемеровский государственный университет культуры и искусств, 2013. - 36 с.: ил., табл. - URL: <http://biblioclub.ru/index.php?page=book&id=274218>
8. Рогов, В.А. Методика и практика технических экспериментов [Текст]: учеб.пособие / В.А. Рогов, А.В. Антонов, Г.Г. Поздняк. – М.: Академия, 2005. – 288 с.
9. Садыков А.М. Методы разработки веб-приложений: учебно-методическое пособие. – Иваново: ИГЭУ, 2019
10. Сейерс Э.Х., Милл А. Docker на практике. – М.: ДМК Пресс, 2020
11. Фишерман Л.В. Git. Практическое руководство. Управление и контроль версий в разработке программного обеспечения. – СПб.: Наука и Техника, 2021

## ***Дополнительная литература***

1. Моделирование систем [Текст]: учебник для вузов / С.И. Дворецкий, Ю.Л. Муромцев, В.А. Погонин, А.Г. Схиртладзе. – М.: Академия, 2009. – 320 с.
2. Порсев, Е. Г. Организация и планирование экспериментов [Электронный ресурс]: учебное пособие / Е. Г. Порсев. — Новосибирск : Новосибирский государственный технический университет, 2010. — 155 c. — Режим доступа: http://www.iprbookshop.ru/45415.html
3. Щербаков, А. Интернет-аналитика [Электронный ресурс]: поиск и оценка информации в web-ресурсах: практическое пособие / А. Щербаков. - М.: Книжный мир, 2012. - 78 с. - URL: <http://biblioclub.ru/index.php?page=book&id=89693>.

# ПРИЛОЖЕНИЯ

# *Листинг*

**Incident.cs:**

namespace IncidentHistoryService.Models

{

/// <summary>

/// Класс инцидента

/// </summary>

public class Incident

{

/// <summary>

/// Идентификатор инцидента

/// </summary>

public int Id { get; set; }

/// <summary>

/// Заголовок инцидента

/// </summary>

public string Name { get; set; }

/// <summary>

/// Время начала инцидента

/// </summary>

/// <returns>

/// Время первой отметки из HistoryMarks<br/>

/// Если History не содержит элементов, вернет null

/// </returns>

public DateTimeOffset? StartDate

{

get

{

if (HistoryMarks.Count == 0) return null;

return HistoryMarks.MinBy(x => x.Date).Date;

}

}

/// <summary>

/// Время окончания инцидента

/// </summary>

/// <returns>

/// Время последней отметки из HistoryMarks<br/>

/// Если History не содержит элементов, вернет null

/// </returns>

public DateTimeOffset? EndDate

{

get

{

if (HistoryMarks.Count == 0) return null;

return HistoryMarks.MaxBy(x => x.Date).Date;

}

}

/// <summary>

/// Последний комментарий

/// </summary>

/// <returns>

/// Комментарий последней отметки из HistoryMarks<br/>

/// Если History не содержит элементов, вернет null

/// </returns>

public string? LastComment

{

get

{

if (HistoryMarks.Count == 0) return null;

return HistoryMarks.MaxBy(x => x.Date).Comment;

}

}

/// <summary>

/// Недоступный во время инцидента сервис

/// </summary>

public string UnavailableService { get; set; }

/// <summary>

/// Недоступные во время инцидента зоны

/// </summary>

public List<string> UnavailableZones { get; set; }

/// <summary>

/// Список тегов

/// </summary>

public List<string> Tags { get; set; }

/// <summary>

/// Список отметок для ведения истории инцидента

/// </summary>

public List<HistoryMark> HistoryMarks { get; set; }

/// <summary>

/// Конструктор с параметрами<br/>

/// Ориентирован на работу с БД

/// </summary>

/// <param name="name">Заголовок</param>

/// <param name="unavailableService">Недоступный во время инцидента сервис</param>

/// <param name="unavailableZones">Недоступные во время инцидента зоны</param>

/// <param name="tags">Список тегов</param>

public Incident(string name, string unavailableService, List<string> unavailableZones, List<string> tags)

{

Id = default;

Name = name;

UnavailableService = unavailableService;

UnavailableZones = unavailableZones;

Tags = tags;

HistoryMarks = new List<HistoryMark>();

}

}

}

**HistoryMark.cs:**

using System.Text.Json.Serialization;

namespace IncidentHistoryService.Models

{

/// <summary>

/// Отметка в истории инцидента

/// </summary>

public class HistoryMark

{

/// <summary>

/// Идентификатор отметки

/// </summary>

public int Id { get; set; }

/// <summary>

/// Комментарий отметки

/// </summary>

public string Comment { get; set; }

/// <summary>

/// Время отметки

/// </summary>

public DateTimeOffset Date { get; set; }

/// <summary>

/// Тег отметки

/// </summary>

public string Tag { get; set; }

/// <summary>

/// Идентификатор инцидента, к которому относится отметка

/// </summary>

public int IncidentId { get; set; }

/// <summary>

/// Инцидент, к которому относится отметка

/// </summary>

[JsonIgnore]

public Incident Incident { get; set; }

/// <summary>

/// Конструктор с параметрами

/// </summary>

/// <param name="id">Идентификатор</param>

/// <param name="comment">Комментарий</param>

/// <param name="date">Время</param>

/// <param name="tag">Тег</param>

/// <param name="incidentId">Идентификатор инцидента, к которому относится отметка</param>

public HistoryMark(int id, string comment, DateTimeOffset date, string tag, int incidentId)

{

Id = id;

Comment = comment;

Date = date;

Tag = tag;

IncidentId = incidentId;

}

/// <summary>

/// Конструктор с параметрами<br/>

/// Ориентирован на работу с БД

/// </summary>

/// <param name="comment">Комментарий</param>

/// <param name="date">Время</param>

/// <param name="tag">Тег</param>

/// <param name="incident">Инцидент, к которому относится отметка</param>

public HistoryMark(string comment, DateTimeOffset date, string tag, Incident incident)

{

Id = default;

Comment = comment;

Date = date;

Tag = tag;

Incident = incident;

}

}

}

**ApplicationContext.cs:**

using Microsoft.EntityFrameworkCore;

namespace IncidentHistoryService.Models

{

public class ApplicationContext : DbContext

{

public DbSet<Incident> Incidents { get; set; } = null!;

public DbSet<HistoryMark> HistoryMarks { get; set; } = null!;

public ApplicationContext()

{

AppContext.SetSwitch("Npgsql.EnableLegacyTimestampBehavior", true);

Database.EnsureCreated();

}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.EnableSensitiveDataLogging();

var configuration = new ConfigurationBuilder().AddJsonFile("appsettings.json").AddEnvironmentVariables().Build();

optionsBuilder.UseNpgsql(configuration.GetConnectionString("DefaultConnection"));

}

}

}

**IncidentController.cs:**

using IncidentHistoryService.Models;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Globalization;

namespace IncidentHistoryService.Controllers

{

/// <summary>

/// Контроллер Incident

/// </summary>

[Route("api/[controller]")]

[ApiController]

public class IncidentController : ControllerBase

{

/// <summary>

/// Настройка для российского формата даты

/// </summary>

private static CultureInfo \_cultureInfo = new CultureInfo("ru-RU", false);

/// <summary>

/// Конструктор по умолчанию<br/>

/// Создает начальные сущности

/// </summary>

public IncidentController()

{

using ApplicationContext db = new();

if (!db.Incidents.Any())

{

Incident incident\_1 = new("Проблемы с Cloud", "Cloud", new() { "Russia", "German" }, new() { "Serious" });

HistoryMark mark\_1 = new("Проблема была обнаружена", DateTimeOffset.Parse("10.07.2022", \_cultureInfo), "Investigation", incident\_1);

HistoryMark mark\_2 = new("Проблема была решена", DateTimeOffset.Parse("12.07.2022", \_cultureInfo), "Resolved", incident\_1);

Incident incident\_2 = new("Проблемы с DNS", "DNS", new() { "France", "Spain" }, new() { "Small" });

HistoryMark mark\_3 = new("Проблема была обнаружена", DateTimeOffset.Parse("11.07.2022", \_cultureInfo), "Investigation", incident\_2);

db.Incidents.Add(incident\_1);

db.Incidents.Add(incident\_2);

db.HistoryMarks.Add(mark\_1);

db.HistoryMarks.Add(mark\_2);

db.HistoryMarks.Add(mark\_3);

db.SaveChanges();

}

}

/// <summary>

/// Метод получения списка Incident

/// </summary>

/// <returns>JSON со всеми Incident</returns>

[HttpGet]

public ActionResult<IEnumerable<Incident>> Get()

{

using ApplicationContext db = new();

return new ObjectResult(db.Incidents.Include(x => x.HistoryMarks).ToList());

}

/// <summary>

/// Метод получения конкретного Incident

/// </summary>

/// <param name="id">Идентификатор искмого Incident</param>

/// <returns>Incident с указанным идентификатором</returns>

[HttpGet("{id}")]

public ActionResult<IEnumerable<Incident>> Get(int id)

{

using ApplicationContext db = new();

Incident? incident = db.Incidents.Include(x => x.HistoryMarks).FirstOrDefault(x => x.Id == id);

if (incident == null)

return NotFound();

return new ObjectResult(incident);

}

/// <summary>

/// Метод создания нового инцидента

/// </summary>

/// <param name="name">Заголовок</param>

/// <param name="unavailableService">Недоступный во время инцидента сервис</param>

/// <param name="unavailableZones">Недоступные во время инцидента зоны</param>

/// <param name="tags">Теги</param>

/// <returns>

/// 200 - успешное добавление, возвращает созданный инцидент<br/>

/// </returns>

[HttpPost]

public ActionResult<Incident> Post(string name, string unavailableService,

[FromQuery] List<string> unavailableZones, [FromQuery] List<string> tags)

{

Incident incident = new(name, unavailableService, unavailableZones, tags);

using ApplicationContext db = new();

db.Incidents.Add(incident);

db.SaveChanges();

return Ok(incident);

}

/// <summary>

/// Метод добавления отметки в историю инцидента

/// </summary>

/// <param name="incidentId">Идентификатор инцидента</param>

/// <param name="comment">Комментарий метки</param>

/// <param name="date">Время метки</param>

/// <param name="tag">Тег метки</param>

/// <returns>

/// 200 - успешное добавление, возвращает созданную метку<br/>

/// 400 - добавление не удалось

/// </returns>

[HttpPut]

public ActionResult<HistoryMark> Put(int incidentId, string comment, string date, string tag)

{

using ApplicationContext db = new();

Incident? incident = db.Incidents.Include(x => x.HistoryMarks).FirstOrDefault(x => x.Id == incidentId);

if (incident == null)

return BadRequest();

HistoryMark historyMark = new(comment, DateTimeOffset.Parse(date, \_cultureInfo), tag, incident);

incident.HistoryMarks.Add(historyMark);

db.Update(incident);

db.SaveChanges();

return Ok(historyMark);

}

}

}

**Program.cs:**

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

app.UseSwagger();

app.UseSwaggerUI();

app.UseAuthorization();

app.MapControllers();

app.Run();

**Dockerfile:**

FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base

WORKDIR /app

EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build

WORKDIR /src

COPY ["IncidentHistoryService/IncidentHistoryService/IncidentHistoryService.csproj", "IncidentHistoryService/"]

RUN dotnet restore "IncidentHistoryService/IncidentHistoryService.csproj"

COPY ["IncidentHistoryService/IncidentHistoryService/.", "IncidentHistoryService/"]

WORKDIR "/src/IncidentHistoryService"

RUN dotnet build "IncidentHistoryService.csproj" -c Release -o /app/build

FROM build AS publish

RUN dotnet publish "IncidentHistoryService.csproj" -c Release -o /app/publish

FROM base AS final

WORKDIR /app

COPY --from=publish /app/publish .

ENTRYPOINT ["dotnet", "IncidentHistoryService.dll"]

**Docker-compose.yaml:**

services:

postgres:

image: postgres:14.4

container\_name: postgres

ports:

- 5432:5432

environment:

- POSTGRES\_USER=\*\*\*

- POSTGRES\_PASSWORD=\*\*\*

- POSTGRES\_DB=IncidentHistoryServiceDB

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U postgres"]

interval: 5s

timeout: 5s

retries: 5

incidenthistory:

container\_name: incident-history

build:

context: .

dockerfile: IncidentHistoryService/Dockerfile

ports:

- 8100:80

environment:

- ConnectionStrings\_\_DefaultConnection=Host=postgres;Port=5432;Database=IncidentHistoryServiceDB;Username=\*\*\*;Password=\*\*\*;Include Error Detail=true

depends\_on:

postgres:

condition: service\_healthy

preventive-works:

container\_name: preventive-works

build: ./preventive-works

ports:

- 8101:8080