Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент: Турсунов Жасурбек Рустамович
(Фамилия, имя, отчетсво)

Группа: ИУ7-66Б

Тип практики: производственная

Название предприятия: ООО «ПолисСофт»

Студент _____ (Подпись, дата) Ж.Р.Турсунов (И.О.Фамилия)

Руководитель практики _____ (Подпись, дата) Н.Б.Толпинская (И.О.Фамилия)

Руководитель предприятия _____ (Подпись, дата) Д.Е.Бекасов (И.О.Фамилия)

Оценка:_____

Москва, 2021 г.

Содержание

Введение **4**

Основная часть **5**

1 Декомпозиция функциональной части **5**

2 Командная разработка **6**

3 Архитектура программно-аппаратного комплекса **6**

4 Сервис хранения данных маршрутов **7**

5 База данных **8**

6 Используемые инструменты и технологии веб-приложения **9**

7 Системные характеристики **10**

8 Модульное тестирование **10**

9 Нагрузочное тестирование **11**

Заключение **14**

Список литературы **15**

Введение

Картинг приобретает все большую популярность и как развлечение, и как вид спорта. Во многих городах появляются картодромы, где можно получить дозу адреналина, улучшить свои навыки вождения, снять стресс и просто весело провести время. Спортивный картинг является основой гоночного вида спорта. И как для любого пилота, периодически нужно выполнять анализ пройденного маршрута, для улучшения навыков вождения. Для получения такого анализа достаточно подключить необходимые приборы к карту и специальное программное обеспечение поможет сделать вывод о пройденном мрашруте.

Цель данной работы - разработка программно-аппаратного комплекса для оценки качества прохождения маршрутов.

Задачи проекта:

1. построение траектории пройденного маршрута;
2. оценка качества прохождения маршрута пилотом;
3. визуализация полученной оценки.

Индивидуальные задачи:

1. получить навыки командной разработки;
2. разобраться уже в существующей реализации сервиса и в используемых технологиях;
3. написать документацию к новым функциям для API;
4. добавить новые функции в API;
5. провести модульное тестирование;
6. провести нагрузочное тестирование и сделать анализ полученных результатов.

Основная часть

1 Декомпозиция функциональной части

На рисунке 1 показан нулевой уровень функциональной модели IDEF0 для разрабатываемого программно-аппаратного комплекса, а на рисунке 2 подробно описан первый уровень данной модели. Для проведения оценки качества прохождения маршрута нужно знать маршрут трека и маршрут который был пройден фактически картом. В качестве выходных данных должна выступать визуализация оценки качества прохождения маршрута.

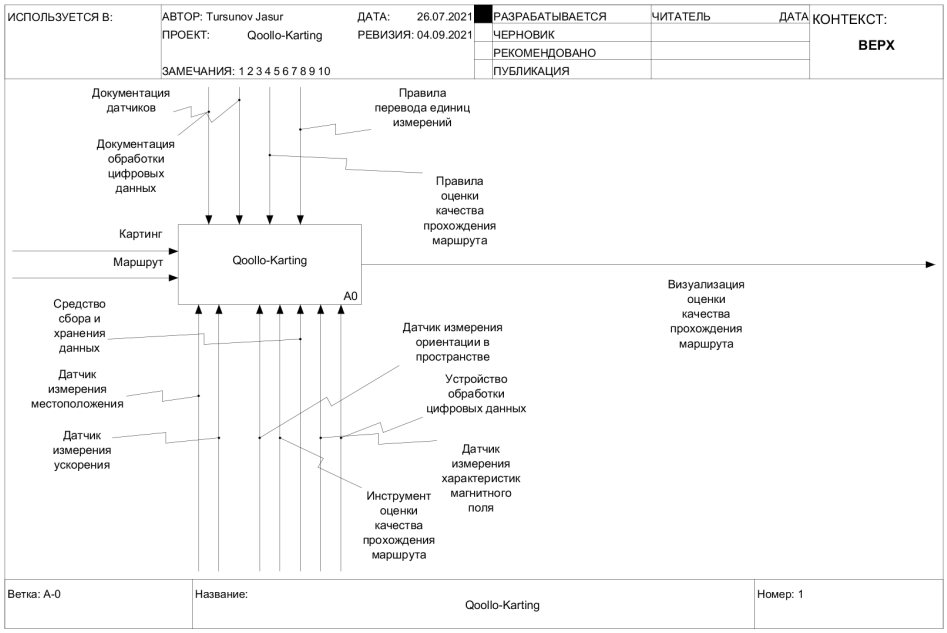

Рисунок 1 – Декомпозиция функциональной модели IDEF0

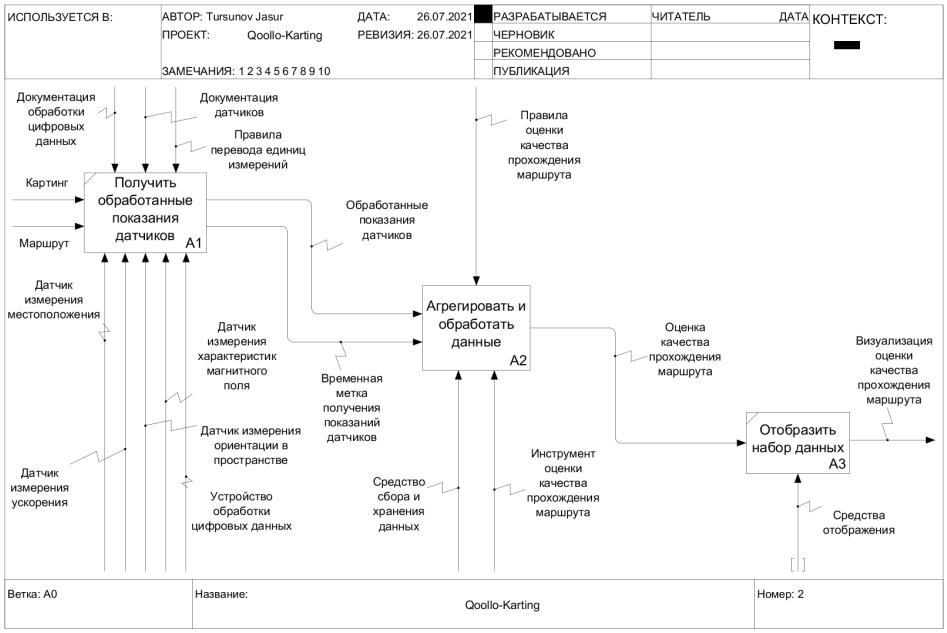

Рисунок 2 – Декомпозиция функциональной модели IDEF0

2 Командная разработка

Во главе ментора и четырех разработчиков была организована командная работа. Для контроля статуса выполнения задач и четкого понимания сроков выполнения поставленных задач перед каждым участником, был использован Redmine. В качестве системы управления версиями в рамках разрабатываемого проекта использовался Git.

3 Архитектура программно-аппаратного комплекса

В ходе совместной работы, командой была разработана архитектура программно-аппаратного комплекса, которая представлена на рисунке 3. В качестве типа архитектуры предпочтение было отдано микросервисной архитектуре, так как было четко ясно, что программно-аппаратный комплекс будет разбит на небольшие автономные компоненты, изменение которых не будет влиять на работоспособность других частей. Проект имеет архитектуру, с единой точной входа, базой данных, сервисом хранения данных маршрутов, сервисом оценки маршрута, сервисом тайлов. Для взаимодействия компонентов была использована архитектура REST. Выбор стиля взаимодействия основывался на таких качествах как: простота интерфейсов, надежность, масштабируемость, возможность использовать разные языки программирования для разных компонент. В общем случае REST является простым интерфейсом управления информацией без использования каких-то дополнительных внутренних настроек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. URL - это адрес, который выдан уникальному ресурсу в интернете. Каждая URL в свою очередь имеет строго заданный формат[1].

Управление информацией сервиса – полностью основывается на протоколе передачи данных. Наиболее распространенный протокол - HTTP. Для HTTP действие над данными задается с помощью методов: GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить). Таким образом, действия CRUD (Create-Read-Updtae-Delete)[2] могут выполняться как со всеми 4-мя методами, так и только с помощью GET и POST.

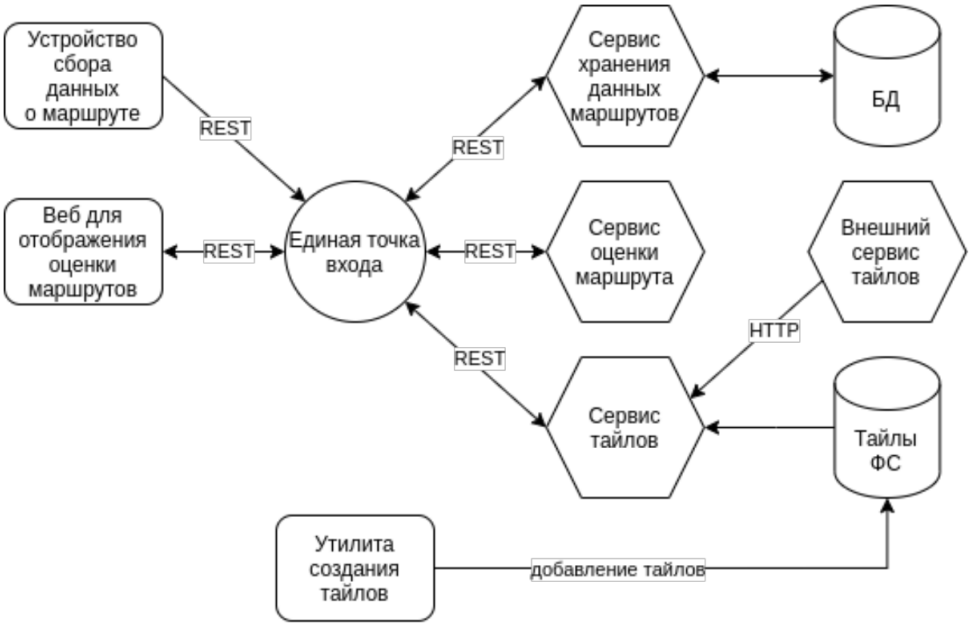

Рисунок 3 – Архитектура проекта

Начальная версия сервиса уже была написана ранее. Одной из поставленных задач было разработка и добавление новых функций в уже существующую реализацию.

4 Сервис хранения данных маршрутов

Первая реализация сервиса была написана с использованием микрофреймворка Flask. Исходя из этого новый функционал продолжил использовать этот фреймворк. Flask — микрофреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Перед тем как начать создавать новый функционал, необходимо было разобраться в уже существующем коде, чтобы не разрушать уже проделанную работу. Для разрабатываемого сервиса была написана документация, с подробным описанием всех POST и GET запросов. Ниже показаны все запросы обрабатываемые сервисом. Последние 4 запроса были реализованы в новой версии проекта.

1. создать маршрут;
2. добавить координаты маршрута;
3. получить список маршрутов;
4. добавить показания датчиков;
5. получить список показаний датчиков;
6. список координат маршрута.

На рисунке 4 показана функциональная модель IDEF0 второго уровня, в которой подробно описана схема работы сервиса хранения данных маршрута.

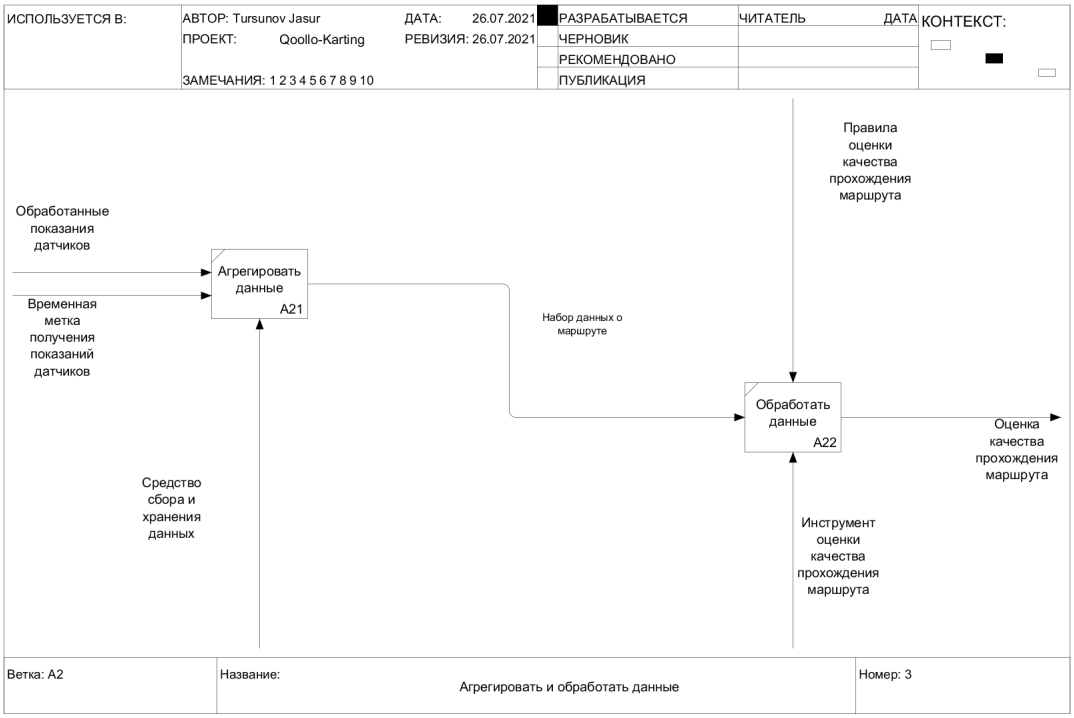

Рисунок 4 – Декомпозиция функциональной модели IDEF0

5 База данных

Не смотря на то, что существует разные типы баз данных, в данной работе была выбрана реляционная модель. Эта модель была выбрана из вида наших данных к которым применимы следующие достоинства:

1. доступность для понимания пользователем. Единственной используемой информационной конструкцией является "таблица";
2. полная независимость данных. Изменения в прикладной программе при изменении реляционной БД минимальны;
3. для организации запросов и написания прикладного ПО нет необходимости знать конкретную организацию БД во внешней памяти.

Для упрощения процесса сохранения объектов в реляционную базу данных и их извлечения была использована ORM. ORM - это технология программирования, которая позволяет преобразовывать несовместимые типы моделей в ООП, в частности, между хранилищем данных и объектами программирования. ORM сама заботится о преобразовании данных между двумя несовместимыми состояниями. Большинство ORM-инструментов в значительной мере полагаются на метаданные базы данных и объектов, так что объектам ничего не нужно знать о структуре базы данных, а базе данных — ничего о том, как данные организованы в приложении. ORM обеспечивает полное разделение задач в хорошо спроектированных приложениях, при котором и база данных, и приложение могут работать с данными каждый в своей исходной форме. Использование ORM не ограничивает выбор БД. Можно в любой момент поменять на подходящую БД.

На рисунке 5 показана ER-диаграмма, описывающая концептуальные схемы разрабатываемой

предметной области. База данных состоит из 3 таблиц: track, coordinate, sensor.

В таблице track хранится уникальная номер о маршруте. Каждый маршрута имеет неограниченное число значений датчиков, в следствие чего, создана таблица coordinate, которая направлена на хранение информации о местоположение карта. Эта модель связана с таблицой sensor, в которой хранится данные с акселерометра, магнитометра, гироскопа.

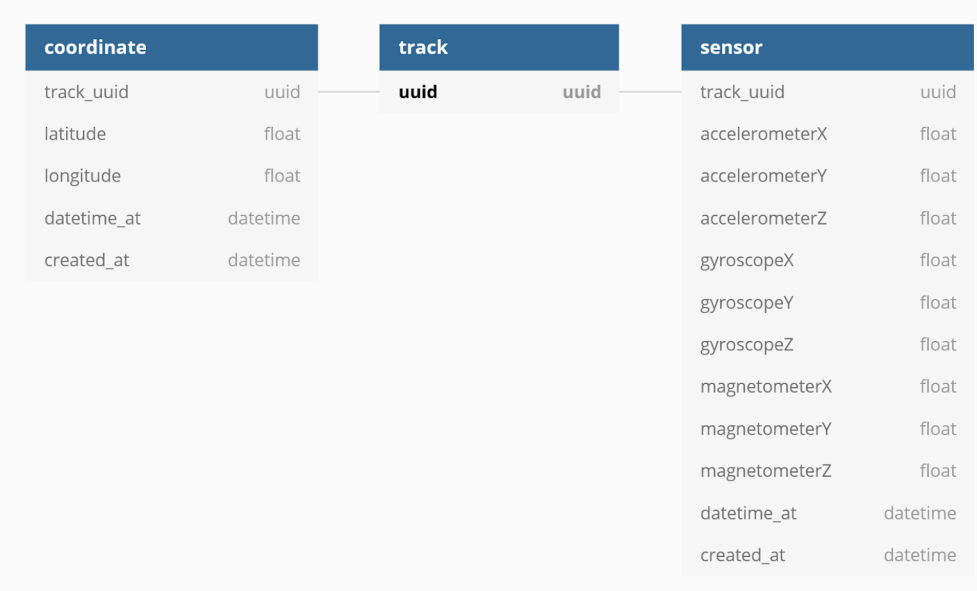

Рисунок 5 – ER-модель структуры БД

6 Используемые инструменты и технологии веб-приложения

Для облегчения разработки веб-приложения нужно использовать фреймворки, которые являются набором шаблонов или заготовок, облегчающий разработку и объединение разных модулей программного проекта.

Для разработки API был использован универсальный микрофреймворк с открытым исходным кодом Flask[3]. Для работы с базой данных использовалась ORM SQLArchemy, в которой модель данных описывается классами Python, и по ней генерируется схема базы данных. В качестве среды разработки выбран редактор PyCharm[4].

Для удобства проверки запросов с клиента на сервер и получения обратного ответа было использовано приложение Postman[5].

Для тестирования производительности, сбора показателей и определения производительности и времени отклика программно-технической системы был использован инструмент Яндекс.Танк[6].

В качестве инструмента для хранения и управления репозиториями Git был выбран GitLab[7]. GitLab дает возможность выполнять совместную разработку силами нескольких программистов, применять обновления кода и откатывать изменения, если это необходимо.

Для управления задачами проектами, учёта временных затрат, отслеживания статуса выполнения задачи было решено использовать веб-приложение Qoollo Redmine.

7 Системные характеристики

Характеристики компьютера на котором проводился эксперимент:

1. операционная система - Windows 10(версия 10.0.19043);
2. процессор - Intel(R) Core(TM) i7-10510U CPU @1.80GHz 2.30GHz, количество ядер - 4, количество логических процессов - 8;
3. объем оперативной памяти - 16 ГБ, тип оперативной памяти - DDR4, скорость оперативной памяти - 2133 МГц, количество планок оперативной памяти - 2, режим работы оперативной памяти - двухканальный;
4. скорость записи и чтения жесткого диска - 193 MB/s.

8 Модульное тестирование

В рамках данного проекта были написаны 15 модульных тестов, направленные на проверку корректного добавления, получения данных из БД. Тестирование было написано с помощью фреймворка unittest, входящий в стандартную библиотеку языка Python[8]. На рисунке 6 показаны результаты тестирования.

```
$ python tests.py -v
test_add_coordinate (__main__.TracksTest) ... ok
test_add_coordinate_incorrect_date (__main__.TracksTest) ... ok
test_add_coordinates (__main__.TracksTest) ... ok
test_add_sensors_data (__main__.TracksTest) ... ok
test_add_sensors_data_empty_track (__main__.TracksTest) ... ok
test_create_new_track (__main__.TracksTest) ... ok
test_get_empty_track_all_coordinates (__main__.TracksTest) ... ok
test_get_empty_track_all_sensors (__main__.TracksTest) ... ok
test_get_one_track (__main__.TracksTest) ... ok
test_get_one_track_empty_info (__main__.TracksTest) ... ok
test_get_range_coordinates (__main__.TracksTest) ... ok
test_get_sensors_data (__main__.TracksTest) ... ok
test_get_sort_coordinates (__main__.TracksTest) ... ok
test_get_track_all_coordinates (__main__.TracksTest) ... ok
test_get_track_list (__main__.TracksTest) ... ok
----------------------------------------------------------------------
Ran 15 tests in 0.864s
OK
```

Рисунок 6 – Результаты тестиривания

9 Нагрузочное тестирование

Для анализа работоспособности сервиса было проведено нагрузочное тестирование с помощью инструмента Yandex-Tank. Были проведены следующие тесты:

1. для определения максимального количества запросов, которые может обработать сервис, был реализован тест с ступенчатым увеличением запросов, где в каждом запросе добавлялось по одной координате. Промежуток количества запросов был от 0 до 180, с шагом 10 и по 30 секунд для каждого запроса. На рисунке 7 показан полученный результат. Можно заметить, что сервис может отработать без ошибок до 140 запросов в секунду. Для увеличения числа запросов можно оптимизировать запросы к БД или проставить индексы при обращении к базе данных;

Рисунок 7 – Результаты измерения пиковой нагрузки

2. чтобы определить сможет ли сервис обработать до 140 запросов, если добавление координат за каждый запрос будет не по одной, а по несколько - 5. Все параметры теста идентичны с предыдущим тестом. На рисунке 8 представлены результаты моделирования. Как видно из рисунка работоспособность сервиса не изменились даже при добавлении 5 координат за раз;

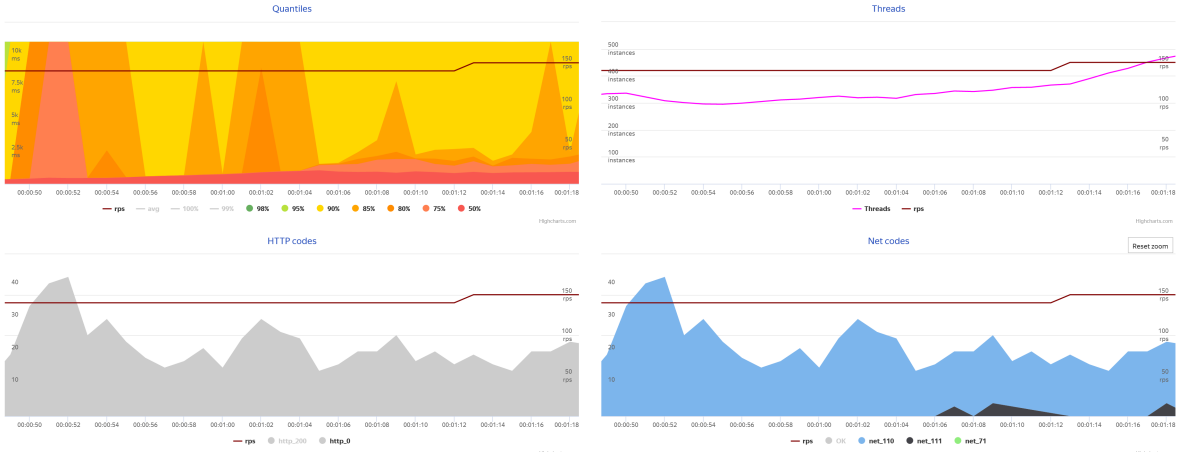

Рисунок 8 – Результаты измерения пиковой нагрузки при добавлении 5 координат за раз

3. восстановление сервиса после кратковременной пиковой нагрузки добавления координат. Данный тест был направлен на то, чтобы узнать сколько времени понадобится сервису чтобы восстановится после кратковременной нагрузки. В ходе тестирования сначала сервис нагружался в пол нагрузки, затем вдвое больше, а затем снова в пол оборота. На рисунке 9 показано, сервис не смог восстановится в течение 30 секунд после пиковой нагрузки. А вот на рисунке 10 при таких же условиях видно, что 60 достаточно чтобы сервич смог нормально заработать поле перегрузки;

Рисунок 9 – Результаты восстановления сервиса после пиковой нагрузки в течение 30 секунд

Рисунок 10 – Результаты восстановления сервиса после пиковой нагрузки в течение 60 секунд

4. чтобы узнать как будет вести себя сервис в течение долгого промежутка времени был проведен тест в котором, добавление координат происходило с постоянной нормальной нагрузкой в течение 60 минут. На рисунке 11 представлены полученные результаты. Можно сделать вывод, что сервис успешно обработал только 69% всех запросов;

Рисунок 11 – Результаты тестирования в течении 60 минут

Заключение

В рамках практики был реализован программно-аппаратный комплекс для оценки качества прохождения маршрутов. С помощью разработанной программы можно получить подробный анализ пройденного маршрута пилотом. В рамках текущего проекта были решены все ранее поставленные индивидуальные задачи:

1. умение работать в команде;
2. разбор уже существовашей реализации сервиса и используемых в ней технологиях;
3. написание документации к новым функциям для API;
4. добавление новых функций в API;
5. проведение модульного тестирования;
6. проведение нагрузочного тестирования и анализ полученных результатов.

Целью каждого программного обеспечения является уменьшение влияние человеского фактора в разных аспектах, тем самым внедряя машинное обучение мы получаем человеконезависимый продукт. Этот программный комплекс не исключение. Развивая этот проект можно находить оптимальный маршрут, определять физическое, психологическое состояние пилота на разных этапах маршрута и многое другое.

Список литературы

[1] Архитектура REST // Habr URL: https://habr.com/ru/post/38730 (дата обращения: 04.07.2021).

[2] CRUD // MaxSite URL: https://maxsite.org/page/restful-crud (дата обращения: 15.07.2021).

[3] Flask // Pallet Projects URL: https://flask.palletsprojects.com/en/2.0.x/ (дата обращения.

[4] PyCharm // JetBrains URL: https://www.jetbrains.com/ru-ru/pycharm/ (дата обращения: 25.07.2021).

[5] Postman // Postman URL: https://www.postman.com/ (дата обращения: 25.07.2021).

[6] Нагрузочное тестирование // Яндекс Танк URL: https://yandex.ru/dev/tank/ (дата обращения: 20.07.2021).

[7] GitLab URL: https://gitlab.com/ (дата обращения: 15.07.2021).

[8] Python // Python Documentation URL: https://www.python.org/ (дата обращения: 03.07.2021).