MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
Sumy State University
Academic and Research Institute of Business, Economics and Management
Department of Economic Cybernetics

«Admitted to the defense»
Head of Department
_____Vitaliia KOIBICUK
(signature)
_____ _____ 2023 year

QUALIFICATION WORK

to obtain an educational degree bachelor_

from the specialty 051 Economics,

educational-professional programs Business Analytics

on the topic: Development of an API Prototype for the Module of Visits, Routes and Tasks Using Modern Frameworks

Winner(s) of the group _AB-91.a.an Kocherezhchenko Roman Dmytrovych

The qualification work contains the results of own research. The use of ideas, results and texts of other authors are linked to the corresponding source

_____
_____
(signature) (Name and SURNAME of the acquirer)

Head: Candidate of Economics, Associate Professor Koibichuk V.V. _____
(position, academic degree, academic title, Name and SURNAME) (signature)

Consultant_____ _____
(position, academic degree, academic title, Name and SURNAME) (signature)

Sumy – 2023

SUMMARY

of Bachelor's level degree qualification thesis on the theme
"Development of an API Prototype for the Module of Visits, Routes and Tasks Using Modern Frameworks"

Student: Roman Kocherezhchenko

The task of optimizing the work of the IT department is multidimensional and complex, there are many solutions and solution options. The very concept of efficiency is based on many factors. The level of management, the education of employees, the coherence of teamwork, the effectiveness of communications, the level of technological equipment. Effective work is impossible without significant optimization of those processes that can be optimized, but those processes that at first glance are impossible or very difficult to automate should also be considered as one of the ways to improve work. Such processes, as a rule, have a complex nature of organization, requirements for creative and creative aspects. One of these processes is development. There are many ways and methods of optimization: generation of types, hints in the editor. But the generation of the entire API with documentation, clear contracts and the possibility of rapid changes in the models is a task that is very difficult to solve. Solving this problem allows you to significantly improve all development processes, because the speed of creating routine tasks will give you the flexibility to delve into more detailed aspects of business logic, architecture, requirements, etc.

The purpose of the qualification work is to develop a project for the automatic generation of an application software interface using current technologies to optimize the creation of prototypes for server parts of applications.

The object of research is methods of creating APIs, technologies and frameworks that provide the possibility of auto-generation of APIs with the necessary methods for creating, reading, editing and deleting data.

The subject of the research is a prototype API module for working with task, visit, and route data, namely a database model diagram, software code for configuring generation parameters, and a script for filling the database with test data.

The objectives of the research are:

1. Explore the needs of today's digital business environment and identify areas where digitization can be a key solution for optimizing work.
2. Gather requirements for the API, taking into account the needs of the business environment and the functionality required to manage routes, visits and tasks.
3. Investigate modern and relevant technologies for the development of the backend part of the program, which ensure the efficiency and productivity of the system.
4. Develop a database schema that will be the basis for API operation and will provide efficient data storage and management.
5. Develop application code that implements API functionality and provides interaction with the route, visit and task management system.
6. Develop an API (application programming interface) capable of automating and optimizing business processes, including CRUD operations (Create, Read, Delete, Update) for the API prototype. Create centralized control over business operations and provide structured information for strategic decision-making.

As a result, develop a subsystem that allows you to quickly create an API taking into account the database schema.

To achieve the set goal and tasks of the research, the following were used: fundamental concepts of theoretical and methodological research on ways of

developing applied software interfaces. A set of general practices for database design.

The information base of the qualifying bachelor thesis was made up of the results of the pre-diploma practice.

The main scientific results of the bachelor's work are a developed project that allows you to get a flexible, auto-generated API that allows you to quickly show the working server part.

Practical development was done in VS Code and DataGrip.

The obtained results can be used by economic and analytical and IT departments.

Keywords: development, application programming interface, generation, database, GraphQL, development optimization, MVP, prototyping.

The content of the master's thesis is presented on 38 pages. The list of used sources from 24 names, placed on 3 pages.

The year of Bachelor’s thesis fulfillment is 2023.

The year of Bachelor’s thesis defense is 2023.

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

SUMY STATE UNIVERSITY

Academic and Research Institute of Business, Economics and Management

Department of Economic Cybernetics

APPROVED BY

Head of Department Candidate

of Economics, Associate Professor

_____ Koibichuk V. V.

“__” _____ 2023.

TASKS FOR BACHELOR’S LEVEL DEGREE QUALIFICATION THESIS

(specialty 051 “Economics” (Study Programme “Business Analytics”))
Student of IV course, group’s code AB-91.a.an.

KOCHEREZHCHENKO ROMAN DMYTROVYCH
(student’s full name)

1. The theme of the work is “Development of an API Prototype for the Module of Visits, Routes and Tasks Using Modern Frameworks” approved by the order of the university from “_12__”___June___2023 year № 0659-VI
2. The term of completed paper submission by the student is “16” June 2023 year.
3. The purpose of the qualification work is to develop a project for the automatic generation of an application software interface using current technologies to optimize the creation of prototypes for server parts of applications.
4. The object of research is methods of creating APIs, technologies and frameworks that provide the possibility of auto-generation of APIs with the necessary methods for creating, reading, editing and deleting data.
5. The subject of the research is a prototype API module for working with task, visit, and route data, namely a database model diagram, software code for configuring generation parameters, and a script for filling the database with test data.
6. The qualification paper is carried out on materials of pre-diploma practice
7. The indicative plan of qualification work, terms of submission of the chapters

to the research advisor, and the content of tasks for the performance of the set purpose is as follows:

Chapter 1. Theoretical and methodological aspects of building API

In chapter 1 it is necessary to discuss the theoretical and methodological aspects that are associated with the development of application programming interfaces. Recent research and publications are reviewed in order to assess the relevance of the topic. Technologies are also considered, such as programming languages, transports for data transfer, frameworks for the practical part of the work.

Chapter 2. Development of an API.

Chapter 2 covers all aspects related to the practical part of API development. Namely, consider the architecture of the application, create a physical database, develop application code for working with the database; to provide an interpretation of the results.

8. Supervision on work:

Chapter	Full name and position of theadvisor	Date, signature	
		task issued by	task accepted by
1			
2			

9. Date of issue of the tasks: “03” 04 2023 year

Research Advisor _____ __ _Koibichuk V.V._____
(signature) (full name)

The tasks has been received _____ _Kocherezhchenko R.D.
(signature) (full name)

CONTENT

CONTENT 6
INTRODUCTION 7
1. CHAPTER 1. THEORETICAL AND METHODOLOGICAL ASPECTS OF THE BUILDING API 8
1.1 Overview of recent research and publications 8
1.2 The purpose and objectives of the research 11
1.3 Choice of implementation methods 12
1.4 Database model design 17
1.5 Comparison of approaches to writing an API 19
CHAPTER 2. DEVELOPEMNT OF AN API 20
2.1 API architecture 20
2.2 API implementation 21
2.3 The result of the introduction of the API 26
CONCLUSION 28
REFERNCES 29
APPENDICES 32

INTRODUCTION

In the context of the modern digital business environment, which is formed due to the digitalization of most processes, it becomes obvious that there is a need for effective management. A similar task is relevant for many areas of business, and among them there are many places where digitalization becomes a key solution for optimizing work.

If you look at an approach without a certain degree of use of the possibilities of modern communication methods, then the control and optimization of the implementation of such tasks is almost impossible. After all, keeping records of data on paper is a task that does not have reliability in itself. It requires a lot of human resources, it takes a lot of time, processes are slowed down and this can have a negative impact on the entire enterprise.

To solve such problems, automate and optimize business processes, an example of an API was developed that solves these problems.

The API, the development of which is described in this paper, is a comprehensive solution that allows you to effectively manage and coordinate tasks, visits, and routes in real time. It includes key features such as: CRUD(Create, Read, Delete, Update) operations for the prototype API. This end-to-end solution provides centralized control over business operations and provides structured information to support strategic decision making.

The main function of the subsystem is the ability to quickly create an API from a database schema.

This system enables a new level of development speed, because the automation of complex tasks of design, development and support of program code provides a significant improvement in work efficiency, and hence the overall improvement of all processes.

1. CHAPTER 1. THEORETICAL AND METHODOLOGICAL ASPECTS OF THE BUILDING API

1.1 Overview of recent research and publications

With the development of digitization processes in all areas, API development methods are developing and improving. Thus, the analysis of scientific publications indexed by the Scopus database over the past 22 years (from 2000 to 2022) shows the great interest of scientists in the specifics of API development methods, the study of their advantages and disadvantages (Fig. 1.1.1). The total number of publications found for this period is 4172 units. The dynamics of publication activity is steadily growing.

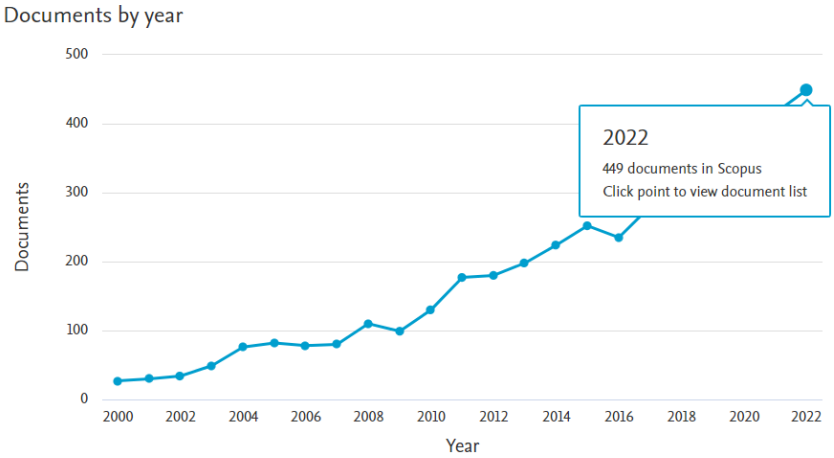

Figure 1.1.1 - Dynamics of publications devoted to the topic "API development methods"

Further bibliometric analysis of the first 2,000 publications from the total array of publications found using the Vosviwer 1.6.18 software allowed them to be divided into four clusters based on the use of keywords in these publications, where each cluster characterizes a certain specificity of API development methods and applied areas of their application. The total number of keywords is 19,538 units, of which the same 1,416 were used by scientists in their publications, and at least 5 in one publication. The number of connections

between clusters is 83926, the total number of connections is 178170. In particular, the keywords included in the cluster (Fig. 2), marked in red, characterize the application programming interface for design, creating an open map, data processing, dissemination of information, regression analysis, computer simulation, standardization of processes, use of API for development of oil production, oil industry. A group of keywords marked in green describes the application of API in the field of pharmaceuticals and medicine, chemistry (drug formulation, drug stability, pharmaceutical preparation, ph measurement, drug compounding). The group of keywords, included in the cluster marked in yellow, characterizes the methods of API development for conducting experiments, diagnostic tests of research accuracy, classification, cluster analysis, risk assessment, comparative research. The fourth group of keywords, marked in blue, characterizes API development methods for high-quality performance of certain tasks, validation of results, use for geometric calculations. That is, as the analysis shows, the scope of application of the API is quite wide and diverse.

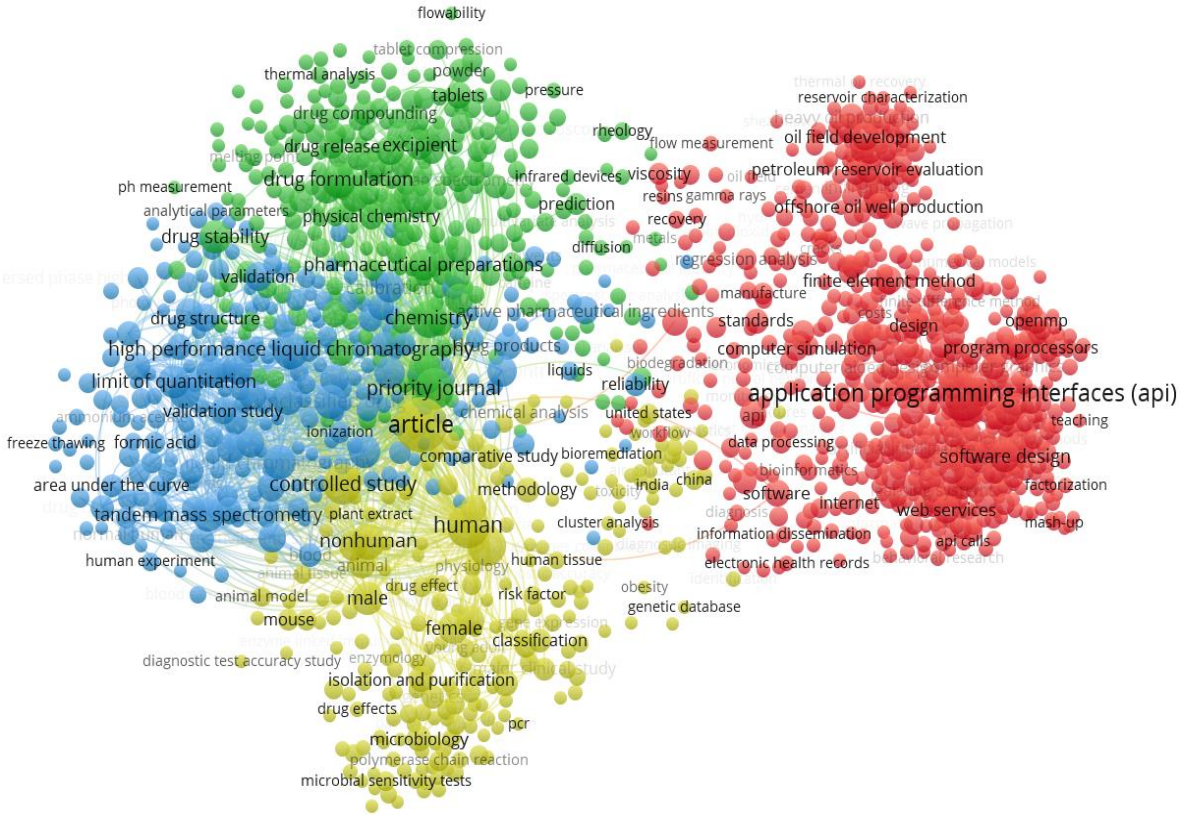

Figure 1.1.2. A map of keywords characterizing API development methods

In addition, two basic principles of application programming interface development that work with web services should be particularly focused on - Representational State Transfer (REST) and GraphQL. After all, service-oriented architecture became the basis for large-scale integration between various platforms, applications, modules, programs and led to the modern state of using cloud services with their advantages and disadvantages [15].

Why should you stop at them? What are their features? Give references to sources. REST is an architectural style used to create scalable, lightweight, and easily extensible web services. Also, REST is a standard protocol for companies to implement and use their services and their flexible integration on various business platforms using application programming interfaces.

In particular, work [6] notes the relationship between Industry 4.0 and cyber-physical systems, which require easy access to data, the need to control and optimize the production process. In order to achieve a unified open platform communication architecture, the authors suggest using RESTful (Representational State Transfer) and, as an alternative to REST, GraphQL. And also perform a comparative analysis of the characteristics of the REST and GraphQL interfaces for the Open Platform Communications Unified Architecture (OPC UA) and perform data reading and writing measurements. Measurements show that GraphQL offers better performance than REST when multiple values are read or written, while REST is faster with a single value [6]. In the work [3], developers-scientists offer a solution that automates the creation of tests for REST APIs based on their specifications, and besides the automatic generation of tests, provides the user with the opportunity to influence the process of creating tests. To do this, the researchers used the latest version of OpenAPI 3.x and a wide range of coverage metrics to analyze the functionality and performance of the generated tests, as well as non-functional metrics to analyze the performance of the API. Experiments confirmed the effectiveness [3]. And in the work [20], the

authors emphasize the inadequacy of the review of the current state of RESTful API testing and conduct a study of 16 sources, classify various problems and solutions related to RESTful API testing and the creation of module tests. In [19], the authors conduct a comparative analysis of REST vs GraphQL on the example of a controlled experiment of 22 students (10 bachelors and 12 masters), who were asked to implement eight requests to access a web service using GraphQL and REST. The obtained results show that GraphQL requires less effort to implement remote service requests compared to REST (9 vs. 6 minutes, average time). These benefits increase when REST requests include more complex endpoints with multiple parameters. Also interesting is the fact that GraphQL outperforms REST even among more experienced participants (graduate students) and among participants with prior REST experience but no GraphQL experience [19].

1.2 The purpose and objectives of the research

In this work, it is necessary to develop an API for the management of visits, routes and assignments of sales representatives for visiting retail outlets.

When performing the work, it is necessary to analyze the subject area, determine the degree of relevance of the development, research the existing methods of creating APIs, determine the requirements and plan the time of the work.

Definition of requirements is one of the first and necessary stages of development, as the creation of a product that will be useful to users requires a clear idea of what needs to be done to create a quality product.

For effective work and implementation of the module, it is necessary to determine the tools for technical implementation. Conduct an analysis between them and choose what will be optimal for development, and in the future for

support. When choosing tools, it is important to consider the team and developers who will work on the development and support of the project in the future.

Often, the list of tools is formed only based on established frameworks and languages within the company or team. To implement this module, you can choose any tools that will be able to cover the necessary tasks, at the same time, they must be quite popular, relatively simple, that is, without unnecessary complications and have an open source code or a specification that guarantees the stability of work.

The design of the API should first of all correspond to the functionality and specifics of the software modules for which it is created in order to create a good user experience from using the service. The interface for interacting with the API, namely auxiliary frontends for convenient documentation and communication, should be both clear and informative, and the sequence of steps for executing processes should be clear.

1.3 Choice of implementation methods

To develop a web application, you need to develop 2 services - a business logic service for interacting with the database - a backend and the database itself around which everything mentioned will be built. To solve these problems, there are their own languages, approaches, frameworks and platforms.

When choosing languages and frameworks for back-end development, there are criteria that will allow you to choose the right technology for the task. Among these, the main ones are:

Performance: Evaluating the performance of the language and framework is important to ensure efficient query processing and high system responsiveness.

Scalability: The choice of language and framework should be based on their ability to scale to handle increasing load and ensure application high availability.

Community and Ecosystem: Having an active community of developers and a developed ecosystem of tools, libraries and modules facilitates the development, problem solving and support of the project.

Security: The language and framework should provide mechanisms to secure the application, including protection against vulnerabilities such as code injection and session forgery.

Ease of development: The choice should take into account the simplicity and ease of use of the language and framework, the availability of tools for debugging, testing and deploying the application.

Integration: It is important to evaluate the possibility of integrating the chosen language and framework with other systems, databases, third-party services and APIs to create complex solutions.

Speed of development: The language and framework should provide high speed of development, provide convenient means for creating and maintaining code, and promote the reuse of components and modules.

Technical support: Having documentation, manuals, support forums, and available consultants for the chosen language and framework will facilitate development and troubleshooting.

However, the most important is the experience and preferences of the team, this factor and others directly affect all development efficiency metrics, such as ROI (Returns on Investments), TTM (Time to Market), CSAT (Customer Satisfaction) and others.

Based on the above factors, the choice of language for developing the back-end part using the Typescript programming language and the Node.js platform. Before a detailed consideration of the frameworks and tools that will also be used, we should dwell in more detail on the language and platform itself. TypeScript is

a programming language that is a superset of the JavaScript language. It adds static typing, the ability to define interfaces, enums, and other entities that help developers build more robust and scalable applications, and it compiles to Javascript.

That is, in this case, it allows you to get one layer of security, because static typing has its advantages. such as: compile-time error detection, code readability improvement, performance improvement, code refactoring and code maintenance simplification, security improvement. These advantages are enough to choose Typescript as the language for implementing business logic on the backend [23].

Node.js is a JavaScript runtime built on the V8 engine (the same one used in the Google Chrome browser)[2]. It allows you to develop server applications and execute JavaScript on the server side. There are also other options like Deno or Bun, but they don't have the same infrastructure as Node.js, so they won't be considered as options [5, 9, 12].

However, using only the language and platform is not enough to build a prototype quickly and efficiently, you need a framework. A framework is a set of out-of-the-box tools, rules, and patterns that help developers build applications more efficiently and quickly. It provides a foundation and structure for software development, simplifies routine tasks, and provides out-of-the-box solutions to common problems.

In order to choose a framework, you need to decide on the architecture for transferring data from the backend to possible clients, it can be like other backend microservices and the frontend. In general, it is possible to single out REST(HTTP) architecture and non-REST. REST is an architecture style that defines the rules and conventions for building web services and APIs. It is based on simple principles and offers a way to organize communication between a client and a server. REST uses various HTTP methods such as GET, POST, PUT, and DELETE to denote different types of operations on resources. For example, GET is used to get data, POST is used to create new resources, PUT is used to update

existing ones, and DELETE is used to delete [21]. Often, this architectural style simply refers to HTTP transport, so in this case, you can simply generalize such a method as an HTTP API. Other architectural approaches and transports include GraphQL, gRPC, SOAP, JSON-RPC, and other variations of these tools [1, 14].

In the process of researching API requirements, it was decided to use GraphQL as the basis for information exchange [11]. GraphQL is a query language and runtime for APIs. Unlike REST, which follows the "one URL, one resource" principle, GraphQL allows clients to request only the data they need and receive it in a single response. GraphQL is also strongly typed and provides flexibility for clients to define the structure of the data they receive [8].

Having decided on the transport, you can go to the database, however, to achieve the best result in development, you should first choose a tool for abstracting over the database in order to be able to quickly switch between them, which is especially useful at the stage, design, prototyping or even choosing technologies. To do this, there are ORM (Object-Relational Mapping) - a technology that allows programmers to work with the database using an object-oriented approach. It allows tables in a database to be linked to objects in code, providing a convenient way to interact with data. It is quite enough of it for abstraction over a DB [22, 25].

In the process of researching available tools, Prisma was chosen, as it provides a wide range of available databases, and convenient interfaces for interacting with data, as well as numerous integrations with other tools, which will also speed up development [16, 17].

During the research, useful tools were also identified that will speed up prototyping many times over. One such tool is “TypeGraphQL Prisma”, as stated on the site itself, it is “Prisma generator to emit TypeGraphQL type classes and CRUD resolvers from your Prisma schema”. This tool will allow you to automatically generate CRUD operations, taking into account the incoming and

outgoing parameters of methods, as well as entity relationships. That will allow you to get a working version of the API prototype only from the database [18].

As a server for GraphQL, Apollo Server was chosen as one of the most popular solutions[13, 10].

There are various databases for storing data. A database is an organized collection of structured data that is designed to store, manage, and process information. It serves as a virtual repository where data can be efficiently organized, structured, and accessed by multiple users or applications.

Among the main types of databases that are widely used are: Relational DBs, Document-oriented DBs, Key-value DBs, Graph DBs and Time-series DBs. They should be considered in more detail for a competent choice of storage, since the technical features of the selected database will need to be taken into account when developing the remaining parts of the program module.

One of the main and popular types of databases are Relational DBs, which are based on the relational algebra model and are used to store and manage data in the form of tables consisting of rows and columns. Also, to work with such databases, SQL (Structured Query Language) is used - this is a declarative language for querying and manipulating data. Examples of such DBs are PostgreSQL, MySQL.

Document-oriented DBs are designed to store, organize, and manage documents in a format such as JSON or XML. These databases allow you to work efficiently with semi-structured hierarchical data and have flexibility in terms of data schema. Examples of document databases are MongoDB and Couchbase.

Key-value DBs - Data is stored as key-value pairs. This type of database is commonly used for caching, storing user sessions, managing counters, and other simple scenarios. Key-value databases are highly performant and scalable. Examples include Redis and Riak.

Graph DBs are designed to store and process graph structures, where data is represented as nodes and edges. Graph databases work effectively with data

related to social networks, recommender systems, link analysis, and other scenarios that require handling relationships between objects. An example is Neo4j.

There are also other types that are more specific to certain tasks. However, in the case of developing such a program module, without specific database requirements, you can take the most popular and reliable solution among others - the Postgres relational database.

1.4 Database model design

In order to understand what data the module will use, you first need to design a database in order to build relationships based on requirements and restrictions.

Data modeling is the process of representing the logical structure and relationships of entities without technical conditions and restrictions to understand the general form of data. For modeling, you can use different tools, one of the main conditions is the information content within the team. As part of the program module, the main elements are just tasks, visits and routes. Based on the hierarchical model where tasks are part of the visit, and the visit is part of the route, you can start building tables and relationships between them. The figure shows a schematic representation for these entities, taking into account the limitations and requirements for the operation of the module(Fig. 1.4.1).

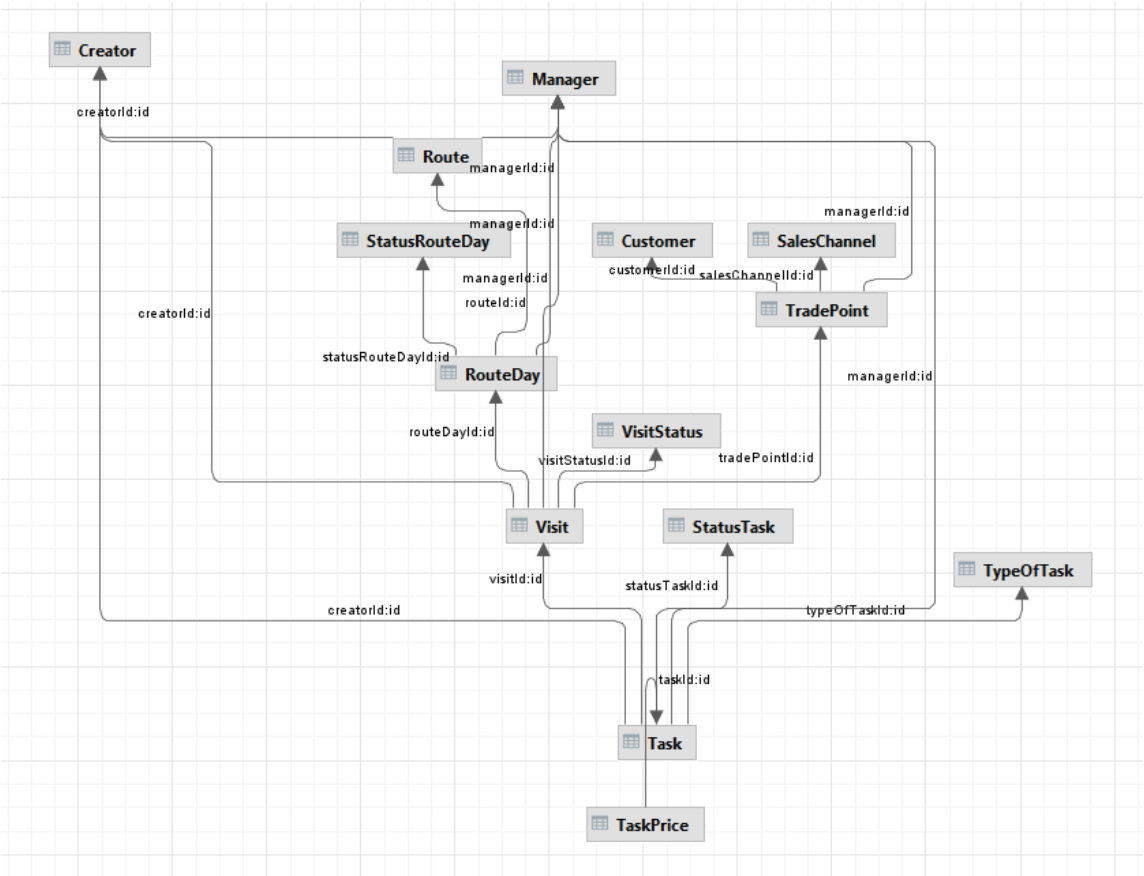

Figure 1.4.1 – Scheme of physical database model

To create such a database model, Prisma was used. To do this, it provides a special model description syntax that is easier to read than SQL queries. For example, this code describes the Tradepoint and SalesChannel models, as you can see this is a one-to-many relationship, one SalesChannel can have many Tradepoint entities(Fig. 1.4.2).

```
model SalesChannel {
  id         Int          @id @default(autoincrement())
  name       String
  TradePoint TradePoint[]
}

model TradePoint {
  id               Int           @id @default(autoincrement())
  code             Int
  name             String
  dayCredit        Int
  sumCredit        Int
  dayCreditSc      Int
  sumCreditSc      Int
  dayCreditC       Int
  sumCreditC       Int
  address          String
  lat              String
  lng              String
  addressGeo       String
  lastVisitDate    DateTime
  lastDeliveryDate DateTime
  Customer         Customer?     @relation(fields: [customerId], references: [id])
  customerId       Int?
  SalesChannel     SalesChannel? @relation(fields: [salesChannelId], references: [id])
  salesChannelId   Int?
  Manager          Manager?      @relation(fields: [managerId], references: [id])
  managerId        Int?

  Visit Visit[]
}
```

Figure 1.4.2 – Code to create a part of entities

1.5 Comparison of approaches to writing an API

In fact, even using almost the entire toolkit that was provided as an example for technologies that are used in development does not help to quickly develop a software interface through autogeneration. In the general case, there are 2 ways to develop, write all the handlers by hand, set up auto-generation [4]. Both of these approaches have pros and cons. However, in order to evaluate the impact of auto-generation of code on the speed of development, one should consider the stages in the development of everything manually using the example of GraphQL: project configuration, database modeling, writing types, writing queries, mutations and resolvers for them. With autogeneration, there are only 2 stages: configuration and database modeling. This is quite enough to consider that autogeneration is suitable for the API prototype.

CHAPTER 2. DEVELOPMENT OF AN API

2.1 API architecture

Architecture is the structure of the interconnection of system components and processes [7]. Competent architecture directly affects the quality of program development and support. In this case, architecture is also a variation of classical architecture.

The software module consists of 2 main components - the server part - business logic, which is responsible for data management and database storage. The software module has a monolithic architecture, that is, the server part is not divided into several component parts, for example, a separate microservice for tasks, a separate one for visits, etc., but a holistic structure, which, taking into account the features of the system, makes it a good choice for development. The figure shows a schematic model of the application architecture(Fig. 2.1.1).

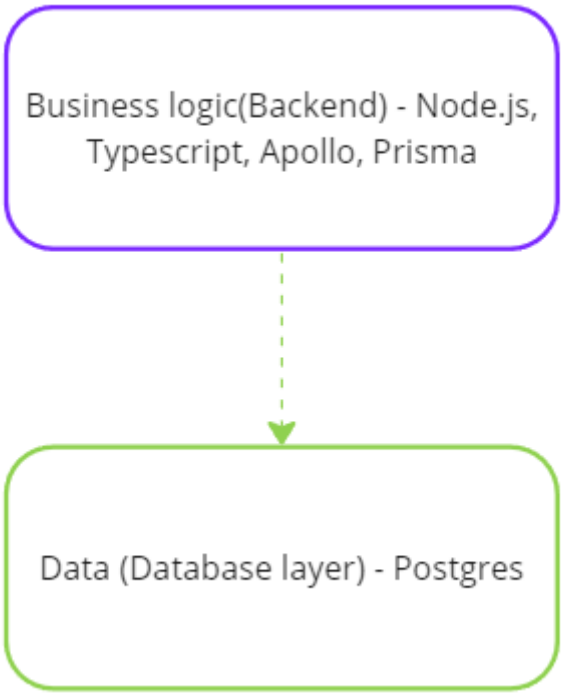

Figure 2.1.1 – API architecture

At the infrastructure level, possible architecture options can be divided into monolithic and microservice.

In a monolithic architecture, all application components are placed in a single executable file or package. They interact directly with each other and are usually deployed on the same server. Monolithic applications are easier and faster to develop, run, and deploy because there is no complexity in managing microservices or services.

In the case of an API prototype task, a monolithic architecture may be preferable, as it allows you to quickly create a foundation of functionality and test the concept. In a microservice architecture, an application is divided into a set of independent services, each of which is responsible for a specific functionality. These services communicate with each other through network calls, usually using an API. The microservice architecture provides more flexible scaling, component isolation, and the ability to deploy and update independently. However, creating a full-fledged microservice application requires more time and resources than a monolith. Since the task is rapid prototyping, a monolithic architecture was chosen.

2.2 API implementation

In order to reproduce the project from scratch, you need to clone it from the project's public repository - https://github.com/romakoch/proto_proj/tree/master.

First you need to define the database schema, in this case Prisma is used and its syntax for defining the schema, all the code can be found in appendix B1.

In order for Prisma to generate its models into true link tables, run the following command in a terminal in the root directory of the project (Fig. 2.2.1).

```
npx prisma migrate dev
```

Figure 2.2.1 – Command to migrate database

Run the following command to configure the server and application (Fig. 2.2.2).

```
npx prisma generate
```

Figure 2.2.2 – Command to generate Prisma client and methods of an API

Also, to fill the database with test data, you need to write and run the following command(Fig. 2.2.3).

```
npx prisma db seed
```

Figure 2.2.3 – Command to fill the database with the test data

And the index.ts code(Fig. 2.2.4).

src > TS index.ts > ...

```ts
import "reflect-metadata";

import { Prisma, PrismaClient } from "@prisma/client";
import { ApolloServer, BaseContext } from "@apollo/server";
import { startStandaloneServer } from "@apollo/server/standalone";
import { resolvers } from "../prisma/generated/type-graphql";
import { buildSchema } from "type-graphql";

const prisma = new PrismaClient();

interface MyContext {
  prisma: PrismaClient;
}

const runGraphql = async () => {
  const schema = await buildSchema({
    resolvers,
    validate: false,
  });

  const graphqlserver = new ApolloServer<MyContext>({
    schema,
  });

  const { url } = await startStandaloneServer(graphqlserver, {
    context: async ({ req }) => ({ prisma }),
    listen: { port: 4000 },
  });
  console.log(`🚀 Server ready at: ${url}`);
};

runGraphql();

```

Figure 2.2.4 – Code to run server

The code needs to be run with the command(Fig. 2.2.5).

```
npm run dev
```

Figure 2.2.5 – Command to run server

After successfully running the project, the following message should appear in the terminal(Fig.2.2.6).

```
> rest-express@1.0.0 dev
> ts-node ./src/index.ts

🚀 Server ready at: http://localhost:4000/
```

Figure 2.2.6 – Message about successful starting of the server

After the successful launch of the project, the page opens on the host http://localhost:4000/ and looks like this(Fig. 2.2.7).

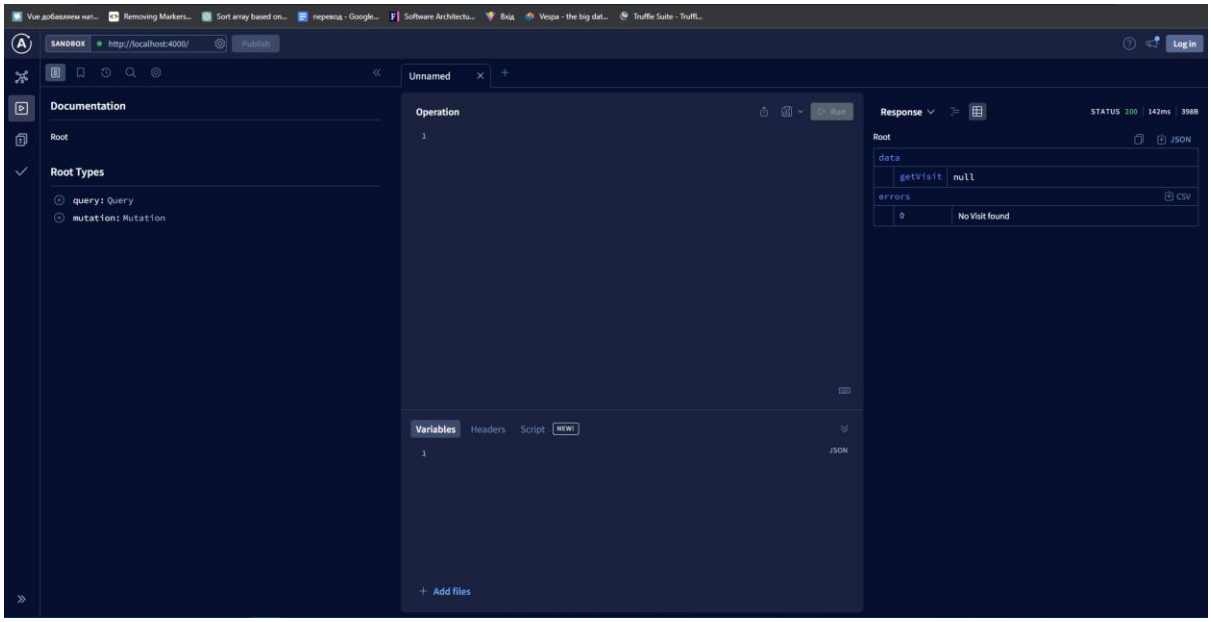

Figure 2.2.7 – Documentation page

This page contains documentation and the ability to test against generated query queries to find the necessary entities(Fig.2.2.8).

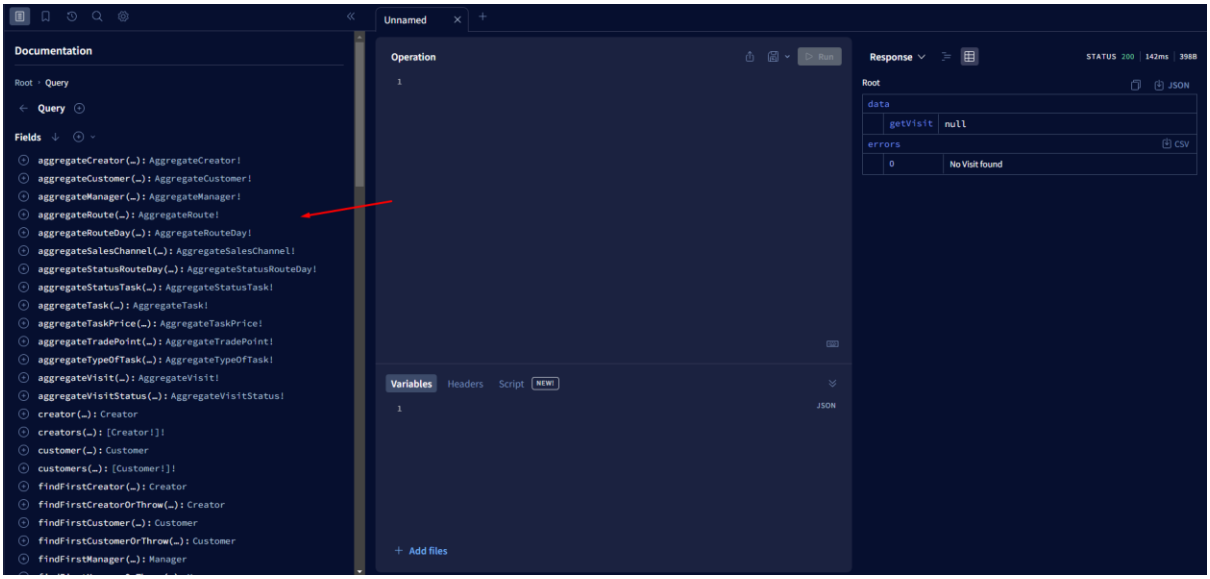

Figure 2.2.8 – Query documentation page

And also a page with mutations for creating, deleting or changing data(Fig. 2.2.9).

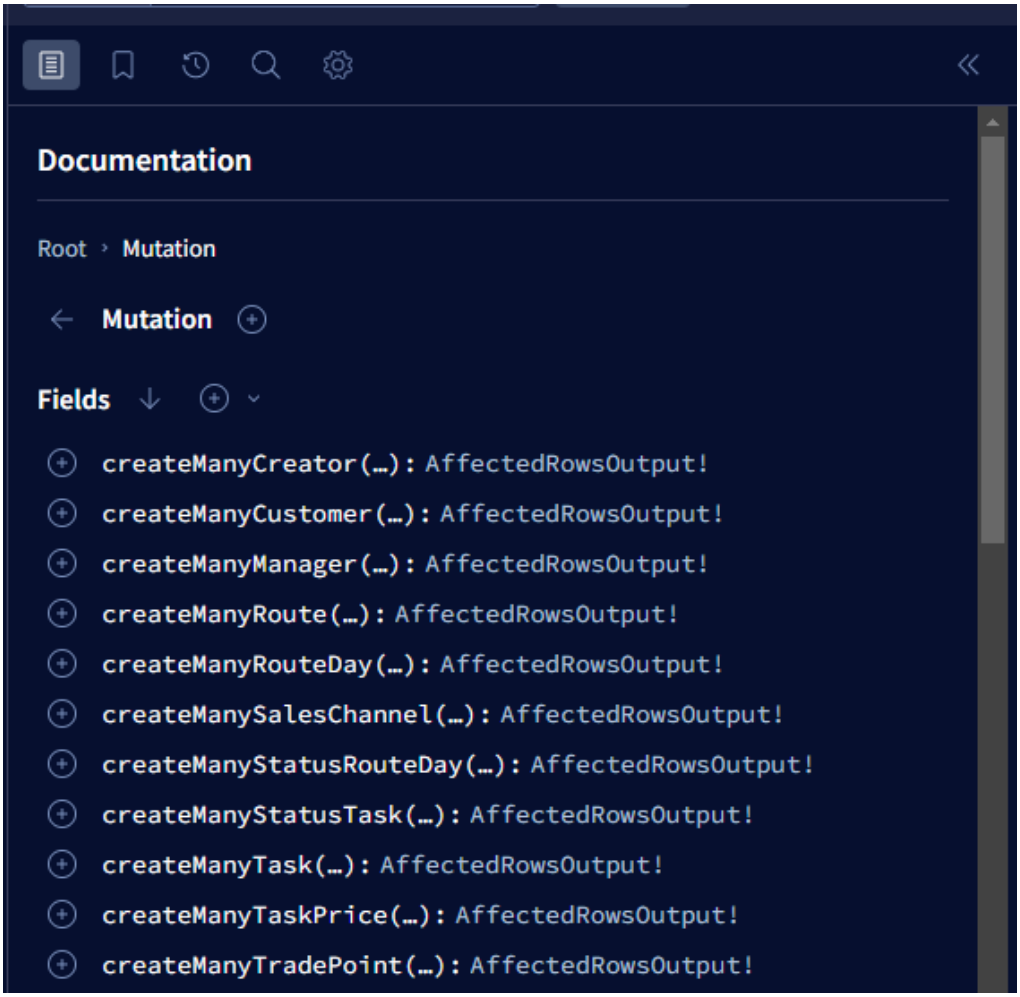

Figure 2.2.9 – Mutation documentation page

2.3 The result of the introduction of the API

Since the purpose of this work was to create an API prototype for the module of visits, tasks and routes, then, taking into account the development features and the proposed approach, it can be considered completed. Thanks to auto-generation, a prototype was obtained in a very short time; it would take man months to develop a similar prototype by hand. The proposed approach made it possible to build some processes differently. For example, instead of working directly with SQL, GraphQL is now used. In general, such a proposed approach was able to optimize the work of the IT department, because the time to show

how the backend will work has been reduced many times, thanks to the rapid prototype, there is time for other things, such as approving decisions, developing requirements, developing the front-end part.

CONCLUSION

In the conclusion of this study, the set tasks related to the development of an application programming interface (API) for managing routes, visits and tasks in today's digital business environment were successfully completed. During the study, modern methods of automating and optimizing business processes were studied. Relevant technologies were selected and applied to develop the backend part of the application, ensuring the efficiency and performance of the system. A database schema was developed that serves as the basis for the operation of the API, provides efficient storage and management of data on routes, visits and tasks. The created Application Programming Interface (API) provides the functionality of CRUD (Create, Read, Delete, Update) operations, error handling and other necessary functions for efficient management of business operations. During testing, the functionality, reliability, and performance of the API was verified, and bugs and issues were fixed. As a result, this study has successfully developed and proposed a comprehensive application programming interface (API) solution that can optimize and automate business processes in a digital business environment. The results of the study will significantly improve the efficiency and overall processes in organizations using the developed solution.

REFERNCES

1. About gRPC // GRPC: [Website]. URL: https://grpc.io/about/ (viewed on: 21.04.2023).
2. About Node.js // Node.js: [Website]. URL: https://nodejs.org/en/about (viewed on: 21.04.2023).
3. Automated Specification-Based Testing of REST APIs / O. Banias et al. Sensors. 2021. Vol. 21, no. 16. P. 5375. URL: https://doi.org/10.3390/s21165375 (date of access: 12.06.2023).
4. Building a GraphQL API – GraphQL API example // APOLLO BLOG: [Website]. URL: https://www.apollographql.com/blog/graphql/examples/building-a-graphql-api/ (viewed on: 25.04.2023).
5. ChakraCore // Github: [Website]. URL: https://github.com/chakra-core/ChakraCore#chakracore (viewed on: 22.04.2023).
6. Comparison of REST and GraphQL interfaces for OPC / Ala-laurinaho R. et al. . Uknown, 2022. 5 p.
7. Dharani R. Web API Design: Crafting Interfaces that Developers Love. Uknown, 2017. 67 p.
8. Giroux M. Production Ready GraphQL. Uknown, 2020. 169 p.
9. GraalVM JavaScript Implementation // GraalVM: [Website]. URL: https://www.graalvm.org/latest/reference-manual/js/ (viewed on: 22.04.2023).
10. GraphQL Yoga // The guild: [Website]. URL: https://the-guild.dev/graphql/yoga-server (viewed on: 25.04.2023).
11. GraphQL: [Website]. 2023. URL: https://graphql.org/ (viewed on: 24.04.2023).

12. Introduction // Deno: [Website]. URL: https://deno.com/manual@v1.34.1/introduction (viewed on: 22.04.2023).
13. Introduction to Apollo Server // APOLLO DOCS: [Website]. URL: https://www.apollographql.com/docs/apollo-server/ (viewed on: 25.04.2023).
14. Jacobson D. , Brail G. , Woods D. APIs: A Strategy Guide: book. Sebastopol: O'Reilly, 2011. 8 p.
15. Koibichuk V., Kocherezhchenko R., Zhovtonizhko I. (2023). Overview of the economic activities of cloud providers and research of theoretical basics of cloud computing. Visnyk ekonomiky – Herald of Economics, 2, P. 80-91.
16. Next-generation Node.js and TypeScript ORM // Prisma: [Website]. URL: https://www.prisma.io/ (viewed on: 20.04.2023).
17. Node.js vs Deno vs Bun: Serving images performance comparison // Medium: [Website]. URL: https://medium.com/deno-the-complete-reference/node-js-vs-deno-vs-bun-a-re-look-at-the-performance-when-serving-images-87a972c9257 (viewed on: 18.04.2023).
18. Repository Pattern // DevIQ: [Website]. URL: https://deviq.com/design-patterns/repository-pattern (viewed on: 25.05.2023).
19. REST vs GraphQL: A controlled experiment / Brito, G. 2020. REST vs GraphQL: A controlled experiment. Paper presented at the *Proceedings - IEEE 17th International Conference on Software Architecture, ICSA*. URL: http://doi.org/10.1109/ICSA47634.2020.00016 (date of access: 12.06.2023).
20. RESTful API testing methodologies: rationale, challenges, and solution directions / A. Ehsan et al. Applied sciences. 2022. Vol. 12, no. 9. P.

4369. URL: https://doi.org/10.3390/app12094369 (date of access: 12.06.2023).

21. Richardson L. , Amundsen M. , Ruby S. RESTful Web APIs: Services for a Changing World: book. Sebastopol: O'Reilly, 2013. 112 p.
22. Should I Or Should I Not Use ORM ? // Medium: [Website]. URL: https://medium.com/@mithunsasidharan/should-i-or-should-i-not-use-orm-4c3742a639ce (viewed on: 25.05.2023).
23. Top 10 Reasons Why Node js is better? // Moreyeahs: [Website]. URL: https://www.moreyeahs.com/top-10-reasons-why-node-js-is-better/ (viewed on: 21.04.2023).
24. What is an API? // Red Hat: [Website]. 2022. URL: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces (viewed on: 18.04.2023).
25. What is an ORM – The Meaning of Object Relational Mapping Database Tools // FreeCodeCamp: [Website]. 2022. URL: https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/ (viewed on: 31.05.2023).

APPENDICES

Appendix A
ABSTRACT OF QUALIFICATION WORK

Summary

Kocherezhchenko Roman Dmytrovych “Development of an API Prototype for the Module of Visits, Routes and Tasks Using Modern Frameworks” – Bachelor's thesis. Sumy State University, Sumy, 2023.

The main goal of the work is the rapid development of a flexible API prototype for working with visits, tasks and routes data.

The paper reflects the process of research, choice of technologies, development algorithm to achieve a practical result. The result is a flexible, auto-generated system that goes beyond the original requirements and provides methods for aggregation, nested structures, and automatic method documentation.

Keywords: development, application programming interface, generation, database, GraphQL, development optimization, MVP, prototyping.

АНОТАЦІЯ

Кочережченко Роман Дмитрович «Розробка прототипу API для модуля візитів, маршрутів і завдань з використанням сучасних фреймворків» - Кваліфікаційна робота бакалавра. Сумський державний університет, Суми, 2023.

Основна мета роботи – швидка розробка гнучкого прототипу API для роботи з даними відвідувань, завдань і маршрутів.

У роботі відображено процес дослідження, вибір технології, алгоритм розробки для досягнення практичного результату. Результатом є гнучка автоматично створена система, яка виходить за рамки початкових вимог і також надає методи для агрегації, дає можливість для роботи з вкладеними структурами і автоматичної документації.

Ключові слова: розробка, прикладний програмний інтерфейс, генерація, база даних, GraphQL, оптимізація розробки, MVP, прототипування.

Appendix B

(informational)

Listing B1 - Prisma scheme code

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

generator typegraphql {
  provider           = "typegraphql-prisma"
  output             = "../prisma/generated/type-graphql"
  // emitTranspiledCode = true
}

model Manager {
  id         Int          @id @default(autoincrement())
  name       String
  Visit      Visit[]
  Task       Task[]
  Route      Route[]
  TradePoint TradePoint[]
  RouteDay   RouteDay[]
}

model Creator {
  id    Int     @id @default(autoincrement())
  name  String
  Visit Visit[]
  Task  Task[]
  Route Route[]
```

```
}

model Customer {
  id         Int          @id @default(autoincrement())
  name       String
  TradePoint TradePoint[]
}

model VisitStatus {
  id         Int     @id @default(autoincrement())
  name       String
  orderBy    Int
  color      String
  accessName String
  Visit      Visit[]
}

model Visit {
  id            Int          @id @default(autoincrement())
  dateIn        DateTime
  datePlan      DateTime
  dateStart     DateTime
  dateEnd       DateTime
  orderBy       Decimal
  isUpdate      Int
  tasks         Task[]
  Manager      Manager?    @relation(fields: [managerId], references:
[id])
  managerId     Int?
  Creator      Creator?    @relation(fields: [creatorId], references:
[id])
  creatorId     Int?
  TradePoint        TradePoint?    @relation(fields:  [tradePointId],
references: [id])
  tradePointId  Int?
  VisitStatus      VisitStatus?  @relation(fields:  [visitStatusId],
references: [id])
```

```
  visitStatusId Int?
  RouteDay          RouteDay?       @relation(fields:  [routeDayId],
references: [id])
  routeDayId    Int?
}

model TypeOfTask {
  id         Int    @id @default(autoincrement())
  name       String
  orderBy    Int
  accessName String
  Task       Task[]
}

model StatusTask {
  id         Int    @id @default(autoincrement())
  name       String
  orderBy    Int
  color      String
  accessName String
  Task       Task[]
}

model TaskPrice {
  id           Int     @id @default(autoincrement())
  brand        String
  height       String
  manufacturer String
  price        Decimal
  Task         Task?   @relation(fields: [taskId], references: [id])
  taskId       Int?
}

model Task {
  id           Int         @id @default(autoincrement())
  name         String
  description  String
```

```
  datePlan     DateTime
  Visit        Visit?      @relation(fields: [visitId], references:
[id])
  visitId      Int?
  Manager      Manager?    @relation(fields: [managerId], references:
[id])
  managerId    Int?
  status             StatusTask    @relation(fields:  [statusTaskId],
references: [id])
  statusTaskId Int
  Creator      Creator?    @relation(fields: [creatorId], references:
[id])
  creatorId    Int?
  TypeOfTask        TypeOfTask?   @relation(fields:   [typeOfTaskId],
references: [id])
  typeOfTaskId Int?
  TaskPrice    TaskPrice[]
}

model StatusRouteDay {
  id         Int        @id @default(autoincrement())
  name       String
  orderBy    Int
  accessName String
  RouteDay   RouteDay[]
}

model RouteDay {
  id               Int             @id @default(autoincrement())
  dateIn           DateTime
  isOnlyOne        Int
  dateStart        DateTime
  dateEnd          DateTime
  comment          String
  isUpdate         Int
  Route               Route?            @relation(fields: [routeId],
references: [id])
```

```
  routeId          Int?
  StatusRouteDay               StatusRouteDay?     @relation(fields:
[statusRouteDayId], references: [id])
  statusRouteDayId Int?
  Manager            Manager?         @relation(fields: [managerId],
references: [id])
  managerId        Int?
  Visit            Visit[]
}

model Route {
  id        Int        @id @default(autoincrement())
  name      String
  dateIn    DateTime
  dateEnd   DateTime
  RouteDay  RouteDay[]
  Manager    Manager?    @relation(fields: [managerId], references:
[id])
  managerId Int?
  Creator    Creator?    @relation(fields: [creatorId], references:
[id])
  creatorId Int?
  isDay1    Int
  isDay2    Int
  isDay3    Int
  isDay4    Int
  isDay5    Int
  isDay6    Int
  isDay7    Int
}

model SalesChannel {
  id         Int          @id @default(autoincrement())
  name       String
  TradePoint TradePoint[]
}
```

```
model TradePoint {
  id               Int           @id @default(autoincrement())
  code             Int
  name             String
  dayCredit        Int
  sumCredit        Int
  dayCreditSc      Int
  sumCreditSc      Int
  dayCreditC       Int
  sumCreditC       Int
  address          String
  lat              String
  lng              String
  addressGeo       String
  lastVisitDate    DateTime
  lastDeliveryDate DateTime
  Customer           Customer?      @relation(fields: [customerId],
references: [id])
  customerId       Int?
  SalesChannel     SalesChannel? @relation(fields: [salesChannelId],
references: [id])
  salesChannelId   Int?
  Manager             Manager?        @relation(fields: [managerId],
references: [id])
  managerId        Int?

  Visit Visit[]
}
```