

Contents

Chapter 1. Introduction.....	1
1.1. Scope and context of the theme approach	1
1.2. Proposed subject.....	1
Chapter 2. Theoretical Background.....	2
2.1. Alternative solutions.....	2
2.2. Web Services.....	3
2.2.1. RESTful services	3
2.2.2. Web API Services.....	4
2.3. Technologies.....	5
2.3.1. MAS	5
2.3.2. .NET	6
2.3.3. ASP.NET Core	7
2.3.4. Fluent Migrations Framework	7
2.3.5. OpenTripMap API.....	8
2.3.6. POSTMAN.....	8
2.3.7. AJAX.....	9
2.3.8. SQL and PostgreSQL	10
Chapter 3. Application design	11
3.1. Application architecture	11
3.2. Application levels.....	12
3.2.1. Presentation tier	13
3.2.2. Data tier	14
3.3 Application block diagram	15
3.3.1. Web application block diagram	16
3.3.2. Web API block diagram	17
3.3.3. Database	23
Chapter 4. The application development.....	25
4.1. Database	25
4.2. Data tier	27
4.3. Presentation tier	31
Chapter 5. Application testing	35
Chapter 6. Demonstration and scenarios	39
Chapter 7. Conclusions and future directions of development.....	45
Bibliography	46
Annexes	48

Search for a complete vacation package using multi-agent negotiation

Mihailov Emilian

Summary

This paper describes a recommender system that has a multi-agent system as its processing core. When the client, using the graphical interface, requests a complete holiday package, a call is sent to the server, which is interpreted as an API. The user's preferences are sent to the multi-agent system, which aims to provide the best possible recommendation. Agents will be able to communicate with each other as needed if they do not have enough knowledge to solve the proposed task.

A particularity of this system is that there are two types of agents: the *coordinator* and the *tourism agents*.

The *coordinator* is the one who informs the travel agents that a request has just arrived and is asked to provide recommendations for four services: finding a departure flight, a return flight, a place to stay during the holiday and some touristic attractions. This agent will also monitor the status of the tasks the agents have to complete and keep track of which services have been completed in order to be provided to the customer. When all services are collected into a complete package, the coordinator will notify the API services that the recommendation is ready and can be sent to the client.

Tourism agents are the ones who do the actual work. This means that they are responsible for providing a solution to the user in accordance with their preferences. Agents have their own knowledge, which can actually be interpreted as a personal database in which agents have stored data about tourist services (flights, properties, attractions). However, this can be incomplete, which is why communication between agents is vital.

Teamwork is a defining trait of a multi-agent system. Agents choose their tasks individually, without talking to others, but when they need help, they can ask others. In the proposed architecture agents become experts in certain types of holiday services, and this makes the recommendation process more efficient. The degree of expert is determined by the ratings the user provides. The more positive the rating, the better the agent is considered an expert, deducing that the local database the agent contains best fits the user's wishes.

The whole purpose of this process is to make it as easy as possible for the user to find what best matches them. It is no longer necessary to search on different websites for generic holiday packages or to plan a vacation by searching on flight websites then renting a space. This project aims to provide all the necessary vacation services to suit the exact needs of the user.

In addition to development ideas, this project also offers the possibility of including algorithms to analyze the user's profile and offer holiday packages based not only on the preferences displayed in the graphical interface, but also from the profile analysis.

The project architecture will apply two important actors: the client and the server. The first one will represent the graphical interface from where the preference calls will be made, and the server will be another resource that will be in charge of calculating all the data and providing the response to the user. The multi-agent system will obviously be stored on the server side.

ActressMAS [14], which is inspired by the actor model, was used as the technology for the multi-agent system. It is quite easy for a programmer to use it, plus all the necessary documentation can be found from official resources.

.NET was used as programming platform. It offered the possibility to create the web application using ASP.NET Core MVC, and for the server ASP.NET Core Web API.

It can also be mentioned that a separate project has been created for the database, which will generate it using migrations. From the same project the populating of the database with information about holiday services (in particular flights, properties, and tourist attractions) will also be performed. As database environment was used PostgreSQL.

Chapter 1. Introduction

1.1. Scope and context of the theme approach

After most recent global disease outbreak Covid-19, the economic crisis that is felt in Europe, but also on other continents caused massive travel demands. The prices on flights or stays are rising as well, particularly in the high season when the demand is increased. The travel agencies are overloaded with requests, and it seems that a real travel boom is looming, but the industry may not be ready.

The UNWTO's latest World Tourism Barometer shows an increase of 182% for international tourism in the first three months of 2022 compared to the previous year [1]. As long as physical travel agencies are overloaded, a future perspective would be the creation of virtual agents that are able to perform the same functions, or even more.

When it comes to planning a vacation, people start browsing dozens of travel sites. Each service is searched separately, whether talking about flight tickets, hotels, or attractions. This search often takes quite a long time, but their efficiency is also low if people are not experienced travelers. Obviously, there are travel agencies that offer full services, but it must be taken into consideration that the agencies' services are included in the final price and are often not cheap.

The aim of the project is to help customers find the most suitable trips according to their preferences by accessing a website. An individuality of the proposed system is the fact that the user can indicate a necessary minimum of information to receive what he needs. The more explicit the preferences are, the closer the result will be to his requirements.

1.2. Proposed subject

The technology of agent-based systems has increased popularity in recent years due to its characteristic of new paradigm for creating a concept, designing and implementation of a software system. That's why a best fit for solving a problem was developing of a multiagent system (MAS) that will compose a recommender system¹. This type of system not only offers the possibility to find a solution to a problem, but also to allow the extensibility of services. In our case, is made reference to the possibility of integrating a system for analyzing the client's profile and providing recommendations based on this factor.

The project consists in realization of an application that will provide to the user a complete vacation package for each request. In this context, a full vacation package includes departure and return flight, a property to stay and some attractions. The required information from the customer, to send the request, are minimal: departure and destination city, the date of check-in and holidays period. Of course, here are more optional customized filters and preferences as flight companies, part of the day most preferred for flight (e.g., morning or night), type of place to stay, some attraction preferences and others. The more preferences are set, the better result customer will get.

The main component that will process the user request and bring a response back to the customer is a multi-agent system. The principle is simple: "*Divide et impera*". Each agent will be responsible for a single vacation service (flight, property, or attractions). When all the agents are done with their task, a full vacation package is constructed based on their results. They act like real life tourism agents. Each have his own responsibilities, knowledges, rating, and the ability to communicate with other agents to complete his task. Last one is essential in a multi-agent system. Agents should be able to ask for some help from the others, to complete their recommendation, or in turn to offer some.

¹ A recommender system, or a recommendation system is a subclass of information filtering system that provide suggestions for items that are most pertinent to a particular user. See Francesco Ricci, Lior Rokach & Bracha Shapira, "Recommender Systems: Techniques, Applications, and Challenges", 2021.

Chapter 2. Theoretical Background

2.1. Alternative solutions

Multi-agent systems are new in software industry, but still, several projects have already been created related to the topic of vacation packages using a multi-agent system. One of them is the MAPWEB [2].

The system of MAPWEB consists of 4 agents: the agent that is responsible for communication between system and the user, another one plans the travel, only one that is searching for information (in this case only on Internet) and the coach agent that will manage the entire system of agents. As in a usual MAS system, in MAPWEB the agents register each recommendation as a case and on next request he will use it to improve his results. The biggest disadvantage of this system is the existence of the “coach agent”, because the communication between agents is strongly managed by him. He tells agents to who they should ask for help. That is a big waste of resources, and a better idea would be to make agents independent when they communicate with each other. A good practice of this system is that agents split the tasks in common agreement based on their service expert rate.

Compared to MAPWEB, our system has two types of agents: a *coordinator* agent that maintain the relation between customer and other agents and the *tourism agents* that are responsible for bringing solutions as the result of customer request.

Another multi-agent system related to recommendation system is the CASIS [3]. This application study the user preferences with each request and improves its suggestions over time. In this case, the multi-agent system is seen as swarm of agents that negotiate to release a result. As in MAPWEB, here is also used the recommendation case-based, to bring to the user the best travel results.

The recommendation process works as follows: the user elicits her preferences; the bees visit all cases in the case base and when they find the best case (according to the user’s preferences) they dance to that case, recruiting other bees to that case; and the case with the greatest number of bees dancing for it is the one recommended to the user.[4]

The advantage of CASIS is that the system has always something to offer to the customer, even the rate of match-cases is low. That means the user will get a recommendation regardless of his preferences or the knowledge that the agents possess. The main problem of this system is the lack of communication between agents if they need to check the cases that the other agents hold.

The information exchange in the MAS should be predominant as it is presented in the project of current document. All services datasets as flights, hotels or attractions are split between agents. This will increase the processing speed of a request. Also, that means each agent will be an expert on data he owns. But also here is a probability that any agent will miss some information for completion of his task. In that case the communication is vital.

The MATRES [5] is an assumption-bases multiagent system that bring to the user personalized travel services. A particularity of this application is the trust degree of the agent in other agents and in oneself. The first one will help the agent to choose who to send the request for help in case of inability to solve the proposed task. That will exclude unnecessary communication between agents. The second one, self-trust degree will help agent to choose which task type to perform.

The advantage of this system is that agents have three sources to find a result for the customer: his own knowledge database, community search (communication with other agents) and Internet search. Another advantage is the one that was not found in previous mentioned systems. It’s the agents’ assumptions that allow agent to reason with incomplete information by making guesses.

The disadvantage on the system is that it doesn’t have a stable recommendation quality. In the testing stage, were used three kinds of recommendation sources: cooperations among agents, decision assumption based on most popular option in the community of users, similar cases. After some simulations, was clear that a compromise should be made when choosing the recommendation strategy. For example, the *Similar cases* method produces better results for the hotel tasks only,

compared with *Most popular option in the community*. Using *Similar cases* is a good strategy to make assumptions for new users when necessary [5].

Another disadvantage is the trust degree on other agents that was too severe. Even if other agent was expert in some sort of service, the current agent refused to communicate with the first one because of low trust degree. Or another scenario is that an agent decreases the trust degree in the other one and in the future requests he will not consider the communication with him.

Our project had also implemented the trust degree between agents, but the main differences is the usage. In MATRES, an agent asks for help only from one most trusted agent, but in our system, the agent will send a request to all agents one at time in descending order of trust.

2.2. Web Services

A web service is a software system whose services are provided on the Internet by special programs and are identified by a URL string. This software system is located by other software systems. Web services provide interaction of software systems regardless of platform and are based on open standards and protocols.

The universality of the technologies is the basis for understanding web services. They run on standard technologies, independent of application providers and other network resources. They can be used in any operating system, application servers, programming languages, etc.

The main advantages are:

- Creating the necessary conditions for the interaction of software components regardless of the platform.
- Web services are based on open standard protocols. By implementing XML, web services can be easily formed and configured.
- The use of HTTP guarantees the interaction of systems through inter-network access.

A great disadvantage is the security level. All modern web services must implement encryption and require user authentication. Whether HTTPS is enough here or more secure protocols like XML Encryption, SAML, etc. are needed is decided during development.

2.2.1. RESTful services

REST is the abbreviation of Representational State Transfer. It is a structural design approach for crafting loosely attached applications using HTTP, often implemented in the growth of web services. REST web services do not impose any rules concerning how it needs to be applied in practice at a low level; it only holds the high-level design guiding principles and leaves it to the developer to think about the implementation [6].

In theory, REST is not tied to the network, but is almost always implemented as such and was inspired by HTTP. As a result, REST can be used wherever HTTP is possible.

The standard database verbs in a RESTful architecture are CRUD (C – create, R – read, U – update, D - delete). Those are most common actions used to manipulate the database records. To not make any confusion, should be enounced that explicitly CRUD operations in the REST context have an equivalent (e.g., create = post/put, read = get, update = put/post, delete = delete). HTTP verbs tell the server what to do with the data specified by the URL. The request can still contain additional information in its body that may be needed to perform the operation - for example, the data user want to save with the resource.

Each REST API request reports results by numerical codes - HTTP statuses.

For example, editing a record on the server may work successfully (code 200), may be blocked for security reasons (code 401 or 403), or even crash in the process due to a server error (code 500).

The REST API also allows customer to exchange more than just text information. User can use this tool to transfer files and data in special formats: XML or JSON.

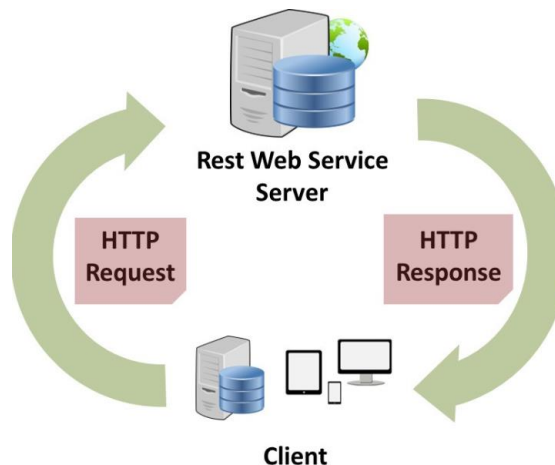


Figure 2.1 RESTful Client-Server Communication Cycle [image source [7]]

As it's shown in the Figure 2.1, it is necessary to separate the needs of the client interface from the needs of the server storing the data. This constraint increases the portability of client code to other platforms and simplifying the server part improves the scalability of the system. The very delimitation into "client" and "server" allows them to develop independently of each other.

Advantages of RESTful web services [8]:

- RESTful web services are platform independent.
- It can be written in any programming language and can be executed on any platform.
- It provides different data format like JSON, text, HTML, and XML.
- It is fast in comparison to SOAP because there is no strict specification like SOAP.
- These are reusable.
- They are language neutral.

2.2.2. Web API Services

An API or Application Programming Interface can be defined as a set of procedures that programmers use to retrieve concrete data or features for an application or service. A Web API on its turn, is an API over the web that communicates with other environments through HTTP calls (see Figure 2.2.).

An API includes functions, classes, methods, and structures that help one application to interact with another. The API contains some "bridges" that allow program A to access data from program B or some of its features. In this way, programmers can extend the functionality of their product and link it to the developments of others.

API can be considered as an interface, because in simple terms, an interface is a layer between application A and application B. It has processes that allow the two programs to exchange information and perform functions related to both sides, hiding the "inner workings" of the programs.

This approach allows for interaction between multiple utilities without having to think about how they are structured, what program logic drives them, or how the data being transmitted is handled. Interfaces simplify the work both for ordinary users and for programmers. The former does not have to think about what is behind the usual functions in their gadgets, and developers do not need to study the code of other programmers to connect someone else's product to their own.

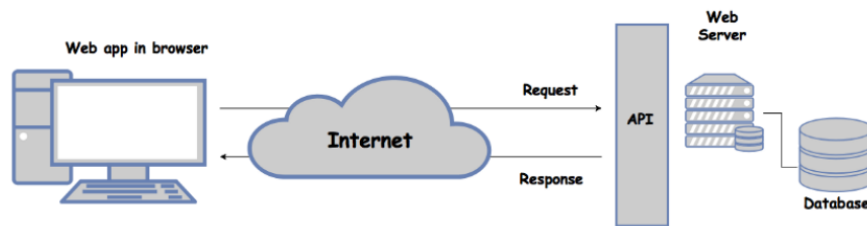


Figure 2.2 API basic architecture diagram [image source [10]]

This is called encapsulation. Hiding part of the functions for the sake of simplifying the work as a whole and minimizing the areas of the software where one of the developers could make a mistake.

Many companies offer APIs as a finished product. For example, American Express² offers the Account and Transaction API. It provides customer-authorized, account-specific data for all American Express proprietary card products including Personal, Small Business and Corporate Cards for parties certified under the Payment Service Directive 2 (PSD2) legislation. [9]

2.3. Technologies

2.3.1. MAS

The core of the current application was built using a multi-agent system. In simple terms, MAS is a system that is formed by multiple agents that are able to communicate and interact with each other. Their main reason is to solve a concrete problem, as it is in our case – finding a full vacation package. Each agent has his own knowledge, capabilities, and intelligence that they demonstrate by solving tasks and interacting with others.

Multi-agent systems have various real-world applications. For example, they can be used to set up robots to perform certain operations, investigate different environments with a high degree of danger, as well as exploring disaster scenarios, investigating transport systems.

Multi-agent systems in robotics, are going through specific and considerable issues. As example will be approached the RoboCup project [11].

The main challenge of this work is to develop a team of robots having an appearance or character resembling that of a real football team. The reason is not only to play using some strategies, but to beat a real football team, more explicitly, the world champion of 2050. Football being a complex game with multiple leagues, and teams (each with own strategy and characters) will give the opportunity to the RoboCup system to learn and study different approaches on solving the problem - winning the football world cup.

The simulation league looks like a standard computer game, but the key difference is that each player is an individual robot, driven by their own program. Each agent must decide on the next move. Because simulation frees researchers from the physical limitations' inherent limitations, these screen players can perform at a much more advanced level [12].

Another agent-based system in ADEPT. It is a project used in telecommunications domain for estimating an offer for installing a network by the network provider.

In the multi-agent system, each department is represented by an agent, and all the interactions between them take the form of negotiations. All negotiations are centered on a multi-attribute object, where attributes are, for instance, price, quality, duration of a service [13].

² American Express (AXP) is an American financial and travel services corporation with operations in more than 110 countries. To see <https://www.investopedia.com/articles/investing/032716/top-5-companies-owned-american-express-jblu-flws.asp#:~:text=American%20Express%20%28AXP%29%20is%20an%20American%20financial%20and,financial%20services%2C%20and%20travel-related%20services%20like%20traveler%27s%20checks>. Visited on 21.09.2022

2.3.1.1. ActressMAS Framework

In the presented project, as multiagent framework was used the ActressMAS. It is inspired by the actor model. The aim is to make it as easy to use as possible and was predestined for students learning in the domain of multi-agent systems [14].

As alternative framework can be counted MASON [15] or JADE [16], but they both are made in Java. The one that was created for .NET is Orleans [17] that also introduce the abstraction of virtual actors.

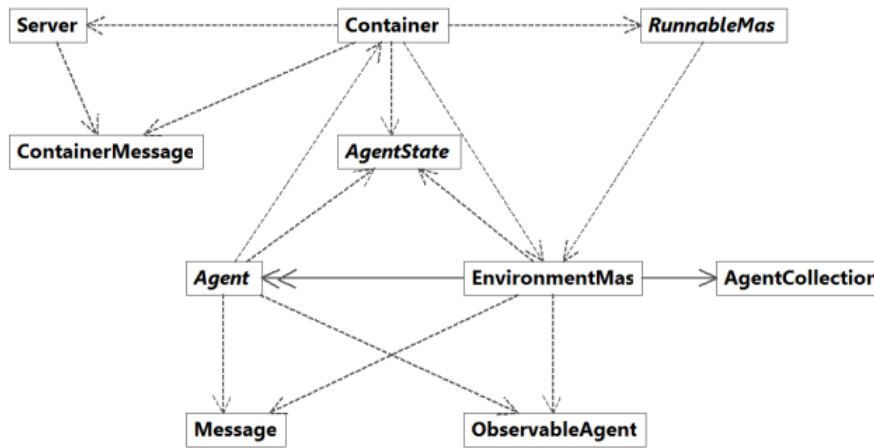


Figure 2.3. The general architecture of the ActressMAS framework. [image source [14]]

The reason of using the ActressMAS framework is the simplicity of use, light architecture (see Figure 2.3.) and compatibility with .NET. The communication between agents can be made by messaging and also through shared memory. The parallel execution allows agents to work on their task asynchronously. This brings high efficiency to the application and increase the execution speed. An interesting fact is that agents have a reactive behavior. That means they are triggered only when they got a message from others. Of course, agents can be customized to act even there are no incoming messages by tracking their state. ActressMAS proposes a lot of customized features that help to build an intelligent MAS environment. Some of these features were used as well in the presented project.

2.3.2. .NET

.NET is a framework from Microsoft that allows the use of the same namespaces, libraries, and APIs for different programming languages. Most commonly, these are four languages in the .NET family:

- C#.
- Visual Basic.
- Visual C++.
- F#.

If .NET didn't exist, users would have to install a runtime environment for programs in each language. That is, to run a Visual Basic application, you must download the runtime for Visual Basic. If the program is written in C#, you would have to download the runtime for it as well.

This is also important for programmers, because it allows them to develop one environment that is used for four languages at once. Otherwise, regular developers would have to wait until a new version of the libraries for their language is released. Less popular languages, like F#, would get updated much later than C#.

The best part of using .NET are the wide documentation libraries created by Microsoft. It's straight-forward for a beginner programmer to find required materials to learn basics, but also the professionals can be delighted by the newest features of .NET updates.

In the current project was used .NET 6 that was released in Nov 2021. Compared to previous version .NET 5, it brings more optimizations and refactoring.

2.3.3. ASP.NET Core

ASP.NET Core is a cross-platform framework for building a modern cloud of Internet-connected applications such as web, Internet of Things, and mobile server applications. ASP.NET Core applications can run on .NET Core or .NET Framework full platform.

ASP.NET Core has several architectural changes that result in a more compact and modular structure. It is no longer based on the System.Web.dll file. It is based on a set of detailed and well-structured NuGet³ packages. This allows you to optimize your application with the NuGet packages you need. The benefits of a smaller application footprint include tighter security, reduced maintenance, improved performance, and lower costs in a "pay for what you use" model.

This are the advantages of the ASP.NET Core:

- Common release history for Web UI and Web APIs
- Integration of modern client frameworks and development schemes
- Cloud-ready, environment-based configuration
- Native support for dependency deployment
- New lightweight and modular HTTP request
- Ability to host in IIS, or in your own application
- Delivered as complete NuGet packages
- New toolset that simplifies the development of modern web applications
- Build and run cross-platform ASP.NET applications on Windows, Linux, and Mac
- Publicly available and socially oriented framework

2.3.4. Fluent Migrations Framework

One of the best ways to achieve flexibility in the software development lifecycle is through automated build and deployment. More often than not, the bottleneck is database deployment. For many years, code deployment and database deployment were separate. The database was always deployed by database administrators, whereas code was usually deployed using automated and/or semi-automated systems.

Now the reliance on database administrators to deploy the deployment process is slower and there is more emphasis on coordination with multiple teams during deployment.

Fluent Migrator was developed for .NET users. It is a database migrator that uses fluent interface to work with database. It's easy to use this framework because the migration classes are some C# classes that inherit the "Migration" base class and just implement the "Up" method that will upgrade the database or "Down" for downgrade. The framework also maintains a "Version" read-only table (See Figure 2.4.) in database for tracking all migrations that were applied to database.

³ NuGet is the package manager for .NET. The NuGet client tools provide the ability to produce and consume packages. The NuGet Gallery is the central package repository used by all package authors and consumers. To see <https://www.nuget.org/>. Visited on 09.24.2022.

```

1 SELECT * FROM public."VersionInfo"
2

```

	Version bigint	AppliedOn timestamp without time zone	Description character varying (1024)
1	202206261343	2022-08-22 16:53:23	Migration202206261343Initial
2	202206261513	2022-08-22 16:53:23	Migration202206261513AddCustomerAndPropertyTables
3	202206281121	2022-08-22 16:53:23	Migration202206281121AddAttractionTables
4	202206281328	2022-08-22 16:53:23	Migration202206281328AddFlightTables
5	202206281648	2022-08-22 16:53:23	Migration202206281648AddAgentsTables
6	202206291217	2022-08-22 16:53:23	Migration202206291217AddFlightPreferencesTables
7	202206301318	2022-08-22 16:53:23	Migration202206301318AddPropertyPreferencesTables
8	202206301423	2022-08-22 16:53:23	Migration202206301423AddAttractionsPreferenceTable
9	202206301733	2022-08-22 16:53:23	Migration202206301733AddPreferencePackageTables
10	202206301856	2022-08-22 16:53:23	Migration202206301856AddAgentsRecommendationTables

Figure 2.4. Example of VersionInfo table created by Fluent Migrator

The biggest question when choosing Fluent Migrator would be “Why not choosing the most popular migration framework for .NET – Entity Framework?”. And the response will be simple. Entity Framework is an Object-Relational Mapping. That means you should have first to create each database entity as a class and then Entity Framework will abstract the code and then virtually will map to tables in the database. Fluent Migrator is commonly used with non-ORM frameworks. Besides the schema migrations, the developer can also migrate data using NuGet packages (e.g., CsvHelper for loading rows from a .csv file) and any data access framework as Entity Framework, Dapper or RepoDB.

2.3.5. OpenTripMap API

An API or application programming interface can be considered as a set of features and functionalities that allows an application to access some concrete data and interact with different operation system or software components.

OpenTripMap API allows users to retrieve data from OpenTripMap [18] website’s database through HTTP calls.

This are the features this API offers [19]:

- Place list - returns a list of places based on a location, type, rate, and other parameters
- Place Details - returns detailed information about a specific place, such as address, description, URL, image, and others
- Place Autosuggest - provides a places query by given (partial) search term and location context
- Geographic coordinates request - returns coordinates for the given placename (city, village, etc.)

The API is free for non-commercial use and its limitation are 5000 request/day and 10 requests/second.

2.3.6. POSTMAN

Postman is a service used for manual and automated testing of HTTP API. It can be used to execute any queries through a user-friendly web interface, create tests to check the API's performance automatically, and much more. Postman is not the only service of its kind, but it's the most popular.

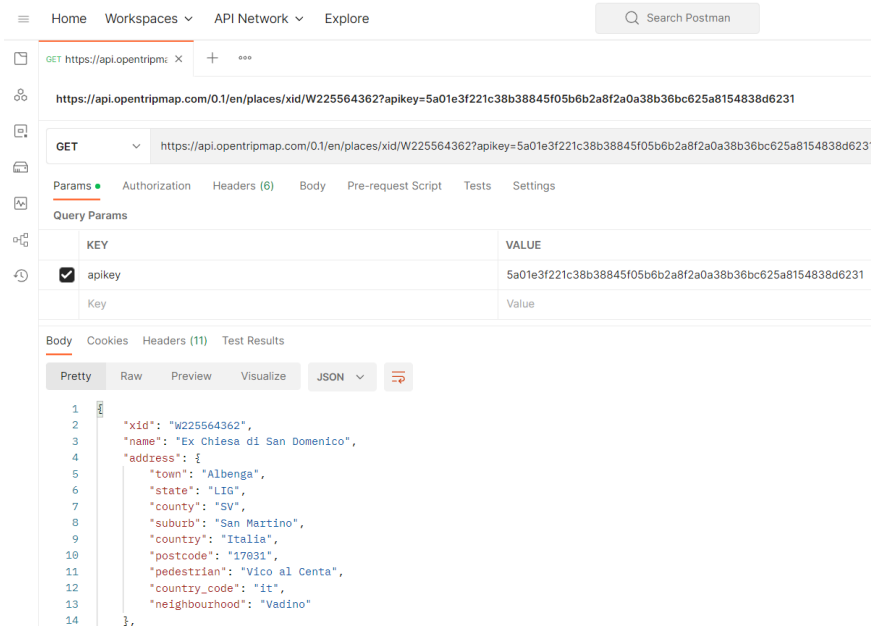


Figure 2.5. Postman http Get request to OpenTripMap API

Today, Postman is a super-popular tool. More than 8 million developers and testers use it. And here's why:

- It's free.
- Very easy to start using
- Postman is intuitive. In just a few minutes after downloading and installing it, user will be able to send his first request.
- Supports multiple APIs. Developer can use Postman to make all kinds of requests to any API (REST, SOAP, GraphQL).
- Postman can be tailored to user's specific needs using the Postman API.
- Easily integrate test suites into favorite CI/CD tool with Newman (the CLI collection runner - lets run Postman collections from the command line).
- Has a large community. Postman is very popular and, as a result, has a large community that will answer most questions.

2.3.7. AJAX

AJAX is an acronym that stands for Asynchronous JavaScript and XML. In fact, AJAX is not a new technology, as both JavaScript and XML have been around for quite some time, and AJAX is a synthesis of these technologies.

When using AJAX, there is no need to refresh the whole page every time, as only a specific part of it is refreshed. This is much more useful, because client don't have to wait too long, and is more economical, since not everyone has unlimited Internet access. In that case, the developer needs to make sure the user is aware of what is happening on the page. This can be done with the use of loading indicators, text messages indicating that the exchange of data with the server. Developers should also be aware that not all browsers support AJAX (older versions of browsers and text browsers). Plus, JavaScript can be disabled by the user.

The work principle is simple. On event trigger by user's manipulations (a button click or page load) XMLHttpRequest object is created by JavaScript. A payload will be sent to the web server and a response will be awaited. Being an asynchronous operation, the web browser will allow customer to keep navigate and use the page without any problems.

2.3.8. SQL and PostgreSQL

Structured Query Language or SQL is a non-procedural programming language designed primarily to describe data, retrieve them from a relational database and then to process. Therefore, SQL operates exclusively with databases and cannot be used alone to create a full-fledged application.

In this case, the tools of other languages that support embedding SQL commands will be needed. It is because of its specificity that SQL is considered an auxiliary tool for data processing. In practice, this language is only used in conjunction with other languages.

In general, application programming tools involve the creation of procedures. SQL does not have this capability. It is not possible to specify ways of solving tasks - only the meaning of each particular task is specified. In other words, in SQL databases, it is the results, not the procedures that lead to those results, that are important.

Advantages of SQL [20]:

- Faster Query Processing –
- Large amount of data is retrieved quickly and efficiently. Operations like Insertion, deletion, manipulation of data is also done in almost no time.
- No coding skills required
- For data retrieval, large number of lines of code is not required. All basic keywords such as SELECT, INSERT INTO, UPDATE, etc. are used and the syntactical rules are not complex in SQL, which makes it a user-friendly language.

You cannot make full use of databases without database management systems (DBMS). There are dozens of such systems and some of most popular are: Oracle, MySQL, Microsoft SQL Server, and PostgreSQL.

PostgreSQL is an open-source object-relational database management system. In PostgreSQL all tables are represented as objects which can be inherited and all operations on them are performed using object-oriented functions. However, the structure of stored files (and even records in them) may be very different. The main difference between PostgreSQL and other RDBMSs is the object-oriented functionality, including support for the ACID (Atomicity, Consistency, Isolation, Durability) concept.

PostgreSQL advantages are [21]:

- Easy to use.
- Has a user-defined data type.
- Open source.
- A lot of community support.
- Make use of Stored procedures.
- It supports ACID, i.e., Atomicity, Consistency, Isolation, Durability.

Chapter 3. Application design

Before start coding, a developer should clearly define the problem his future program needs to solve. Because without a good definition of the problem, can be spent a lot of effort and time on solving the wrong problem. At this stage a simple formulation of the essence of the problem without any hints to its possible solutions is carried out, and it should be formulated in a language understandable to the user, i.e., it should be described from the user's point of view.

Another step is setting program requirements. They are a detailed description of all the features of a program and the actions the program must perform. Such requirements are sometimes also called "Functional Specification" or simply "Specification". Requirements are developed to minimize changes to the system after the start of direct development. Such requirements must necessarily be formal, i.e., documented. Working out the requirements is very important, because it allows developer to define the functionality of the program before the start of programming.

On creating a development plan stage, developer should already be formally making a plan for software development, taking into account the problems at hand and the requirements that have been worked out. In other words, developer should make a plan of how he will proceed.

System architecture development or high-level design is very important, because without a good architecture, developer can solve the right problem, but come to the wrong solution. A good program architecture makes programming easier, while a bad architecture makes it harder.

After system architecture is done, developer will work on detailed design at a low level. In other words, classes and methods are designed here, different options and reasons for choosing the final approaches and implementation methods are considered, evaluated, and compared.

When developing small programs, programmers usually design the program themselves at this level. It looks like writing pseudocode or drawing schemes, that is why this stage is often considered as a part of direct coding.

3.1. Application architecture

When a developer would undertake to write a small, but real and growing project, he will firsthand how important it is for a program not only to work well, but also to be well organized. Don't believe that well-designed architecture is necessary only for large projects. Complexity tends to grow much faster than program size. And if developer of the application doesn't take care of it beforehand, there comes a point where he stops controlling it pretty quickly. The right architecture saves a lot of effort, time, and money. And often determines whether the project will survive. And even if it's just a matter of "building a stool," it's still very helpful to design it first.

There is no generally accepted term for "*software architecture*". However, when it comes to practice, it is clear to most developers which code is good and which is bad. Good architecture is, first of all, an advantageous architecture making development and maintenance of the program simpler and more efficient. A program with a good architecture is easier to extend and change as well as to test, debug and understand. So, in fact, it is possible to formulate a list of quite reasonable and universal criteria:

- System efficiency.
- System flexibility.
- Expandability of the system.

The requirement for the system architecture to be flexible and extensible (that is, capable of change and evolution) is so important that it has even been formulated as a separate principle - the Open-Closed Principle (the second of the five SOLID principles): Software entities (classes, modules, functions, etc.) should be open to expansion but closed to modification [22].

The Client-Server architecture involves the separation of service provisioning and requesting on different computers in the network, each of which performs its tasks independently of the others.

In this architecture several client computers (remote systems) send requests and receive services from a centralized service machine (server), which may also be called a host system.

The client machine provides the user with a so-called "user-friendly interface" to make it easier for the user to interact with the server.

In the current project, was applied two-tier architecture (Figure 3.1.). Two-tier it is called because of the need to distribute the three basic components between the two nodes (client and server). This type of architecture is used in client-server systems, where the server responds to client requests directly and complete, while using only its own resources. That is, the server does not call third-party network applications and does not access third-party resources to fulfill any part of the request.

This architecture seems to be the most logical for a client-server architecture. In it, however, two variants can be distinguished. When general data are stored on the server and their processing logic and business data are stored on the client machine, this architecture is called "fat client thin server". When not only the data but also the logic and business data are stored on the server, it is called "thin client fat server". This architecture is the prototype of cloud computing.

The applied approach was the "fat server". The client should maintain a lightweight structure and to not be aware of any business logics. Would be way easier for the system that encapsulates the business logic to communicate directly with server.

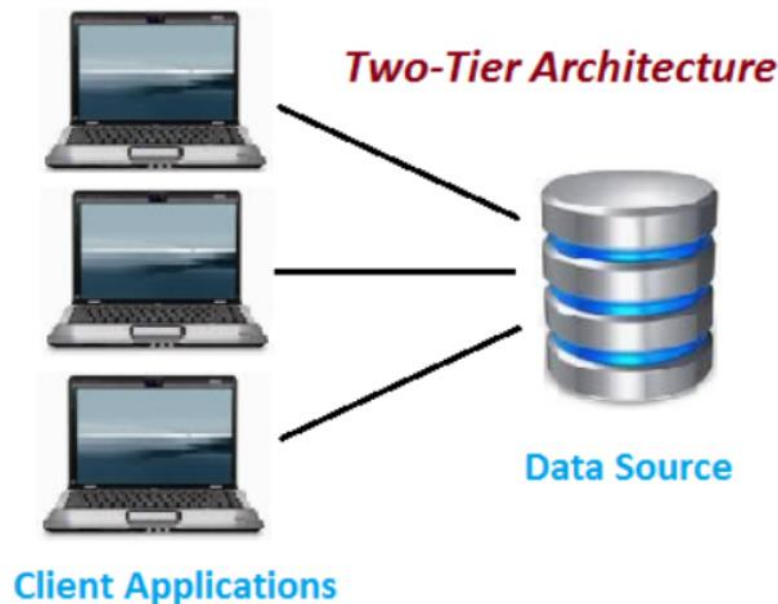


Figure 3.1. Two-Tier Architecture [image source [23]]

This type of architecture is easy to maintain and modify, and also should be considered that communication is way faster than others.

3.2. Application levels

As was mentioned, the application uses two-tier architecture. As presented in Figure 3.2., the client's request is sent from *Presentation* to *Data layer*. The last one encapsulates both, *Application* and *Server* layers. The *Application layer* stores what is called business logic. Here the request is processed. This layer has direct access to the database using *Server layer*. By "direct" is meant that application does not apply a separate layer for each of the *Data layer*.

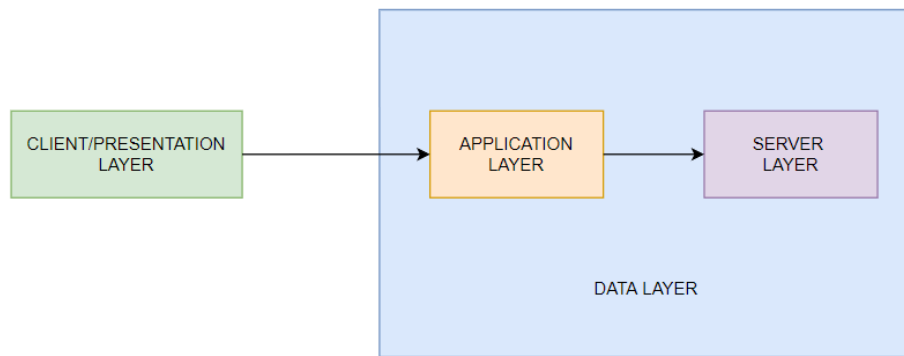


Figure 3.2. Application layers

3.2.1. Presentation tier

This is the first and uppermost layer that is present in the application. This layer represents the user interface, that is, the presentation of content to the end user through a graphical interface. This layer can be accessed through any type of client device, such as desktop, laptop, tablet, cell phone, etc. For content displayed to the user, the relevant web pages must be retrieved by a web browser or other presentation component running on the client device.

In more technical terms, presentation tier's front-end is built with HTML, cascading style sheets – CSS and JavaScript. It is deployed to a web application that is able to communicate with other layers through HTTP calls. This tier should not be aware about any business logic. It just sends a request to an API and waiting for a response. Where the API get the results from is not such important from this layer. If the data provider ever changes, the customer doesn't even have to notice.

The user is considered the trigger of events. Through some manipulations on the web browser, for example a button click, JavaScript will intercept the event and using AJAX will send a HTTP call to the backend of web application. From this point the data will be validated and sent to data layer through a HTTP request as well. The response will be interpreted and sent to the user's interface also by the web application.

To build the web application was used ASP.NET Core MVC Web Application.

The concept of MVC pattern implies the division of the application into three components:

- **Model:** describes the data used in the application as well as the logic that is directly related to the data, such as validation logic. Typically, model objects are stored in a database. In MVC, models are represented by two main types: view models, which are used by views to display and communicate data, and domain models, which describe the data management logic.
- **A view:** is responsible for the visual part or the user interface, often an HTML page, through which the user interacts with the application. A view may also contain logic related to the display of the data. A view must not contain any logic, however, about user request handling or data management.
- **Controller:** represents the central component of MVC, which provides the link between the user and the application, the view, and the data repository. This contains the logic for processing the user's request. The controller receives the user's input and processes it. And depending on the results of the processing, it sends a certain output to the user, e.g., in the form of a view filled with model data.

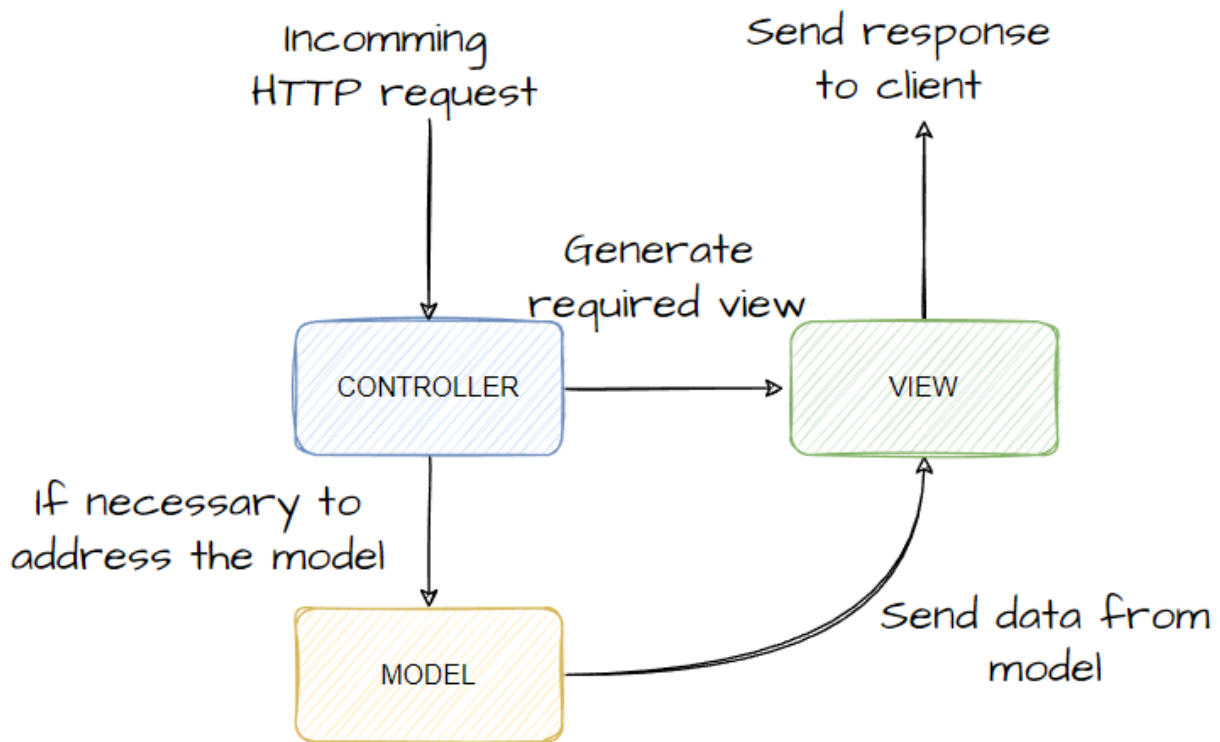


Figure 3.3. MVC request flow

In Figure 3.3., the model is an independent component - any changes to the controller or view have no effect on the model. The controller and view are relatively independent components. For example, you can access a particular controller from a view and generate views from a controller, but they can often be changed independently of each other.

This delineation of application components enables the concept of responsibility separation, where each component is responsible for its own strictly delineated scope. This makes it easier to build work on the individual components. And it makes it easier to develop, maintain and test the individual components. For example, should be cared about the visual part or the frontend, can be tested the view independently of the controller. Or can be focused on the backend and test the controller.

3.2.2. Data tier

This tier can be considered as a server that is responsible for request processing and transactions management. The server works on clients' jobs and manages the execution of their jobs. After each job is completed, the server sends the results to the client who sent the job. A “job” can be interpreted as a HTTP request.

The framework used for this tier to build HTTP services was ASP.NET Core Web API. For structuring the data tier, was used the “Onion Architecture”. It is the division of the application into layers. There is one independent layer which is at the center of the architecture. The second level depends on this level, the second level depends on the third level, and so on. So, it turns out that the second-dependent one overlaps around the first independent level. Around the second, the third, which may also depend on the first, is superimposed. Figuratively, this can be expressed as an onion, which also has a core, around which all the other layers, up to the husk, are layered.

The number of levels may vary, but in the center is always the Domain, that is, those model classes that are used in the application and whose objects are stored in the database. Is also shaped by business logic. The application layer’s services use this one for processing the user request data and then to receive a response for sending it to the next layer.

The first layer around the domain model consists of the interfaces that control the operation of the domain model. These are usually the repository interfaces through which developer interact with the database.

The external layer represents those components that change very frequently. Typically, the Application layer contains the lightweight services that bring data from database using repository interfaces and send them to API layer.

The last layer that has access to others is the API. It can be interpreted as API of the user interface. The controllers of API layer intercept the HTTP requests sent from web application and send them to the next layer for processing.

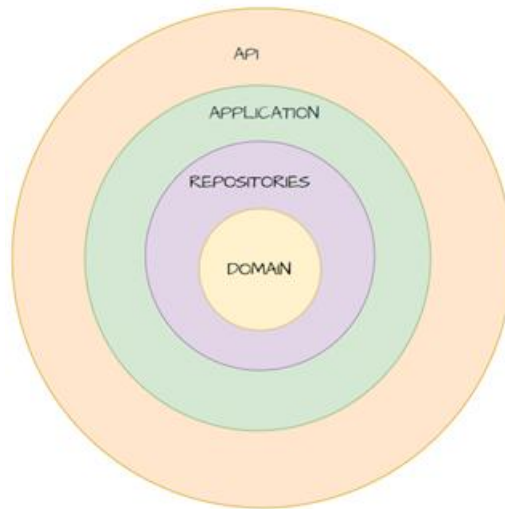


Figure 3.4. Data tier architecture

This architecture is ideal for applications with long lifecycles and complex business logic. Its use in such projects leads to excellent results because of the architecture's inherent emphasis on separating different aspects of the application. Onion Architecture pays special attention to describing system behavior in terms of contract programming and putting infrastructure code into external modules.

3.3 Application block diagram

As was presented previously, the application is formed by two layers. They are totally separated and can communicate with each other through HTTP calls.

In the Figure 3.5. is presented the simplified diagram of the application. The *User interface* is the graphical interface used by customer to send a request. There is no other type of users in this application. To access all services the app offers, customer should create an account, to login and that's all. The registration is required for tracking individually user's preferences and in that way to create a customer's profile.

The web application *MVC Controller* provides communication between the user and the system and uses the model and representation to implement the necessary reaction to user actions. As a rule, at the controller level, filtering of received data and authorization is performed - the user's rights to perform actions or receive information are checked.

The *Presentation level* does not contain any business logic. The processing of data is done on the *Business level*. *Web API Controller* is similar to ASP.NET MVC controller. It receives a HTTP request and sends back a response to the caller. Based on the incoming request URL and HTTP verb (Get, Post or others), Web API finds the route and decides which endpoint to execute. These controllers are using the *Application Services*. Here can be considered each service as a manager that uses required repositories (from *Repository*) or even other services from *Domain*. Using *Repository*, the application services are saving the request data (e.g., user preferences over vacation package).

The business logic from *Domain* has only one purpose – to process the user's request considered as a set of preferences. The whole business logic is based only on multiagent system. The purpose of the *MAS Environment* is to give a recommendation response based on user preferences.

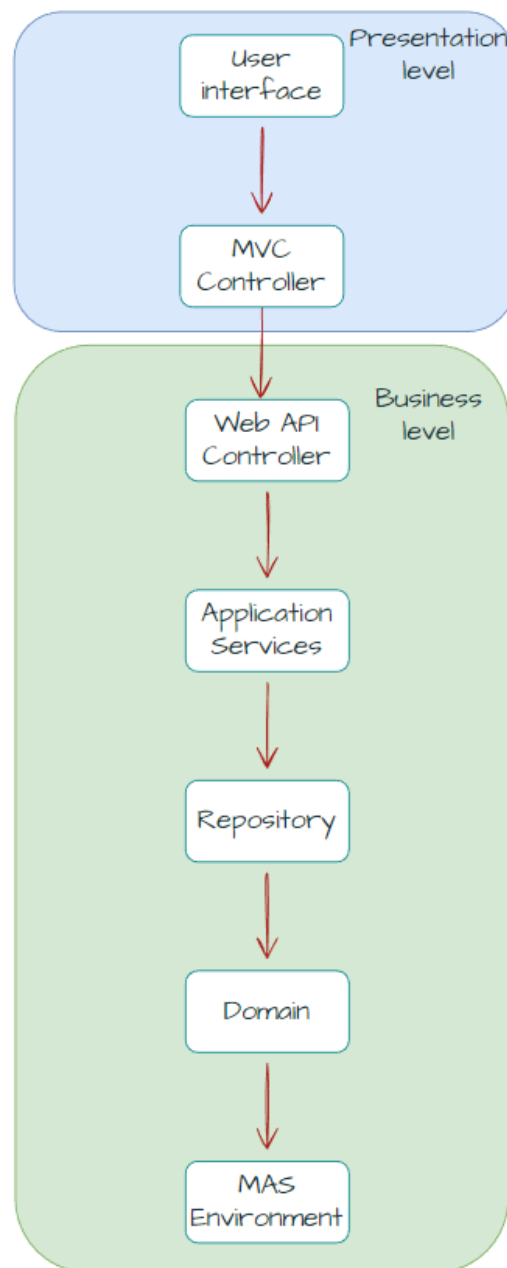


Figure 3.5. Application high level diagram

3.3.1. Web application block diagram

The web application has been designed to accommodate three actions: ask, receive, show. In other words, send a request to the server, receive the response, and show it to the user. In the application flow there is no data processing, the server takes care of everything. Respectively, the structure of the application is not very sophisticated. Figure 3.6 shows the flow behind a user request. The first condition or first step is the user authentication. It was only necessary because of the need to create the user profile. This will help in the training of custom agents that will be trained based on user feedback.

After logging in, the user will be redirected to the services homepage. This page has one purpose: to search for a complete holiday package as close as possible to the user's preferences. After entering the preferences, the user will click on the search button. This will trigger an event in JavaScript that was waiting for the button to be clicked. This function will use AJAX to trigger the MVC controller that is responsible for sending the user's preferences to the server.

From inside the MVC controller an HTTP request will be sent to the server. When a response is received, it will be returned to AJAX, which in turn will use JavaScript functionality to refresh the view and display the result.

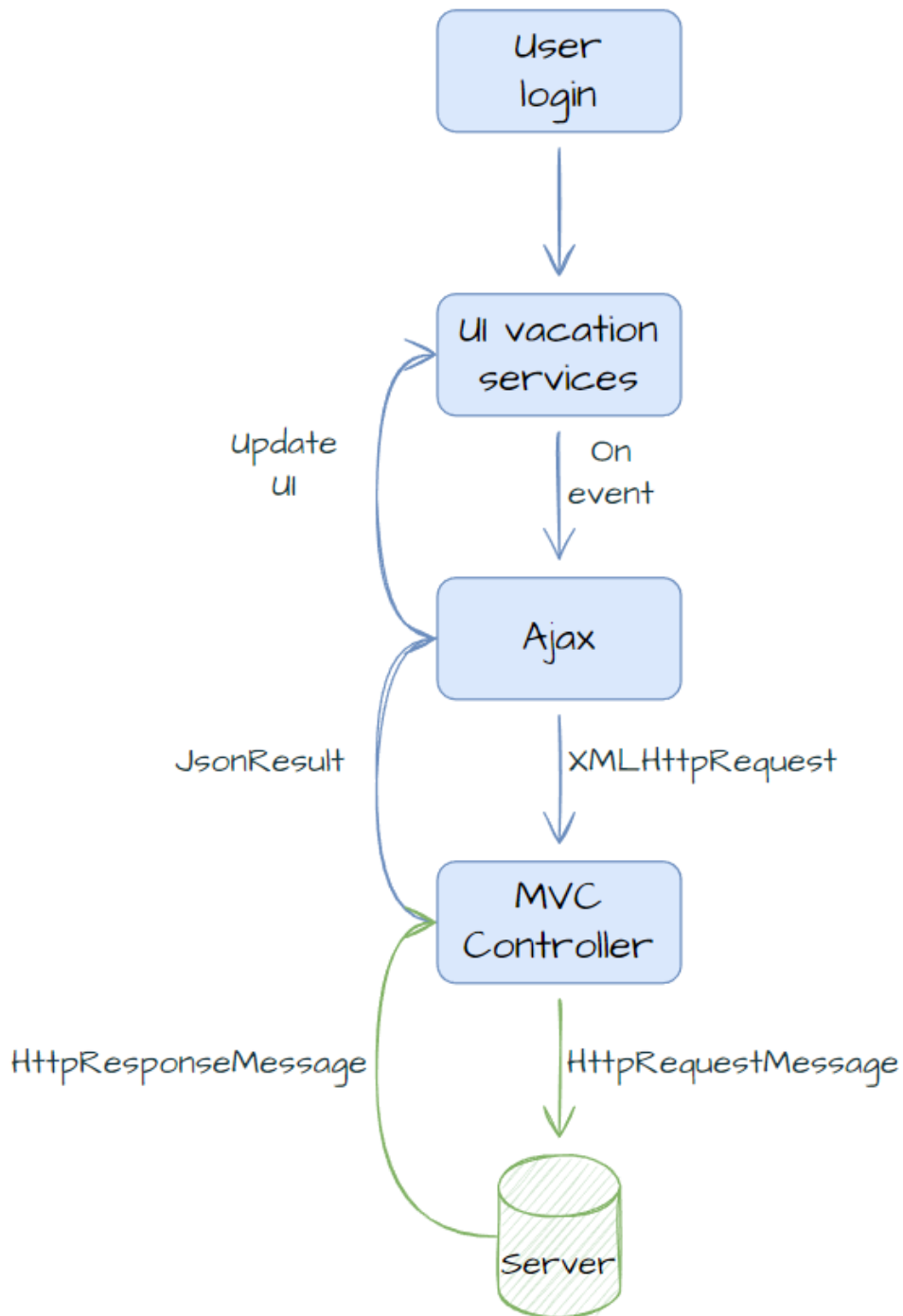


Figure 3.6. User sending a request for a full vacation package recommendation

3.3.2. Web API block diagram

The API application or rather the server does not only contain a part of the business logic but effectively every operation that needs to be calculated, operated takes place here. Therefore, the analysis of the user's preferences as well as the provision of an express response takes place here.

Design of MAS startup

The core of the API is considered the multi-agent system. Its purpose is to solve problems at an asynchronous pace. Breaking up tasks and solving them at the same time is the best strategy.

The biggest problem in designing a project to include a multi-agent system was: How will the MAS system be integrated into an API and how will they communicate?

A decision was made that the MAS environment should be created and initialized with the server "wake-up". The system instance can be invoked via a class that applies the Singleton⁴ pattern design. This was necessary to ensure the uniqueness of the MAS system instance, but also a better balancing of multi-threading access.

Access to the multi-agent system instance can theoretically be achieved from anywhere in the project, which is a disadvantage. Ideally, the system should be "*black box*". That is, the API service sends a request to the MAS and receives a response. This can be achieved by separating the MAS system from the server and creating yet another separate API. This would transform the application into a 3-tier architecture. This approach was not chosen because of the need to keep the architecture as simple as possible and to eliminate dependencies. But still, it was a good alternative.

After initialization of the multi-agent system instance, as shown in Figure 3.8., when the API is started, agent data is read from the database.

Also, from the database all information about flights, hotels or attractions is taken and shared equally between agents. A very important feature is the fact that the degree of expertness of an agent depends directly on his personal database. If the data the agent keeps changes, his rating will be meaningless.

Next, agents will be added to the MAS environment. "*Agent Coordinator*" is needed to coordinate the completion of all tasks. When this time comes, this agent will notify the "*Recommendation Service*" in an indirect way that the recommendation package has been completed. More information about that you can read in next chapters.

All agents except "Coordinator" will be marked as available. The last step in this flow is to start the MAS environment. Agents will now be able to receive requests and solve problems.

⁴ Singleton class allows to allocate and instantiate data single time.

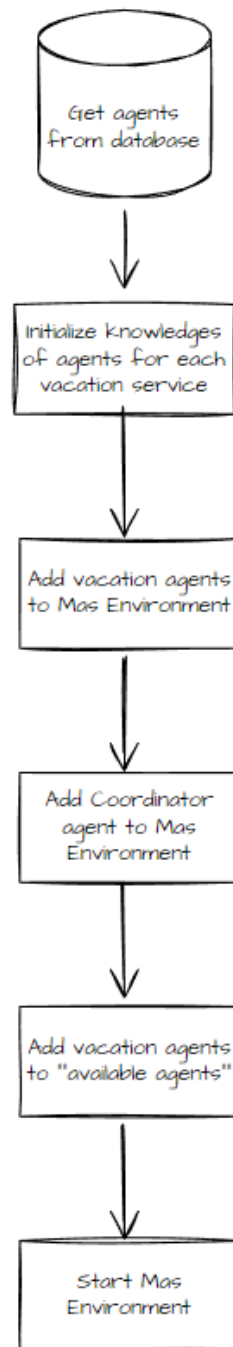


Figure 3.7. MAS environment configuration and start

Design of received vacation package request

Figure 3.7. shows the complete path that a request for the delivery of a holiday package from the user takes.

Everything starts from the API controller. It receives a request from the client interface, specifically the Web Application, the body of which contains the user's preferences. This data is stored in the database using repositories. Along with the preferences, a record of the client request itself is created in the database, and the preferences, recommendations and evaluations that will be performed on this holiday request will all be linked in the database to the so-called "*client request*".

Next, information about the degree of trust of an agent in other agents is taken from the database. This data will be sent to MAS Environment.

The process of sending, checking, and receiving the holiday package is managed by the *Recommendation Service*. First the MAS system is asked to provide the list of available agents.

Afterwards, the preference package is sent to be saved in the MAS memory, and the available agents are informed that a new request has just arrived.

From this point on, the multi-agent system will work to solve the proposed problem. In the meantime, the *Recommendation Service* will start a timer with an arbitrary value assumed to be 10 seconds (*10 seconds is also the timeout time of the HTTP request. If the response is slower, the Web Application will close the HTTP request and will not wait for a response*). Every few milliseconds, this service will query the MAS system trying to find out if the agents have already found a solution. If the answer is yes, the service will query the MAS to provide the recommendation package. If the agents take too long to solve the dilemma, the service will stop the timer and send a blank response to the Web Application. This scenario is not very probable, but still needed to be considered.

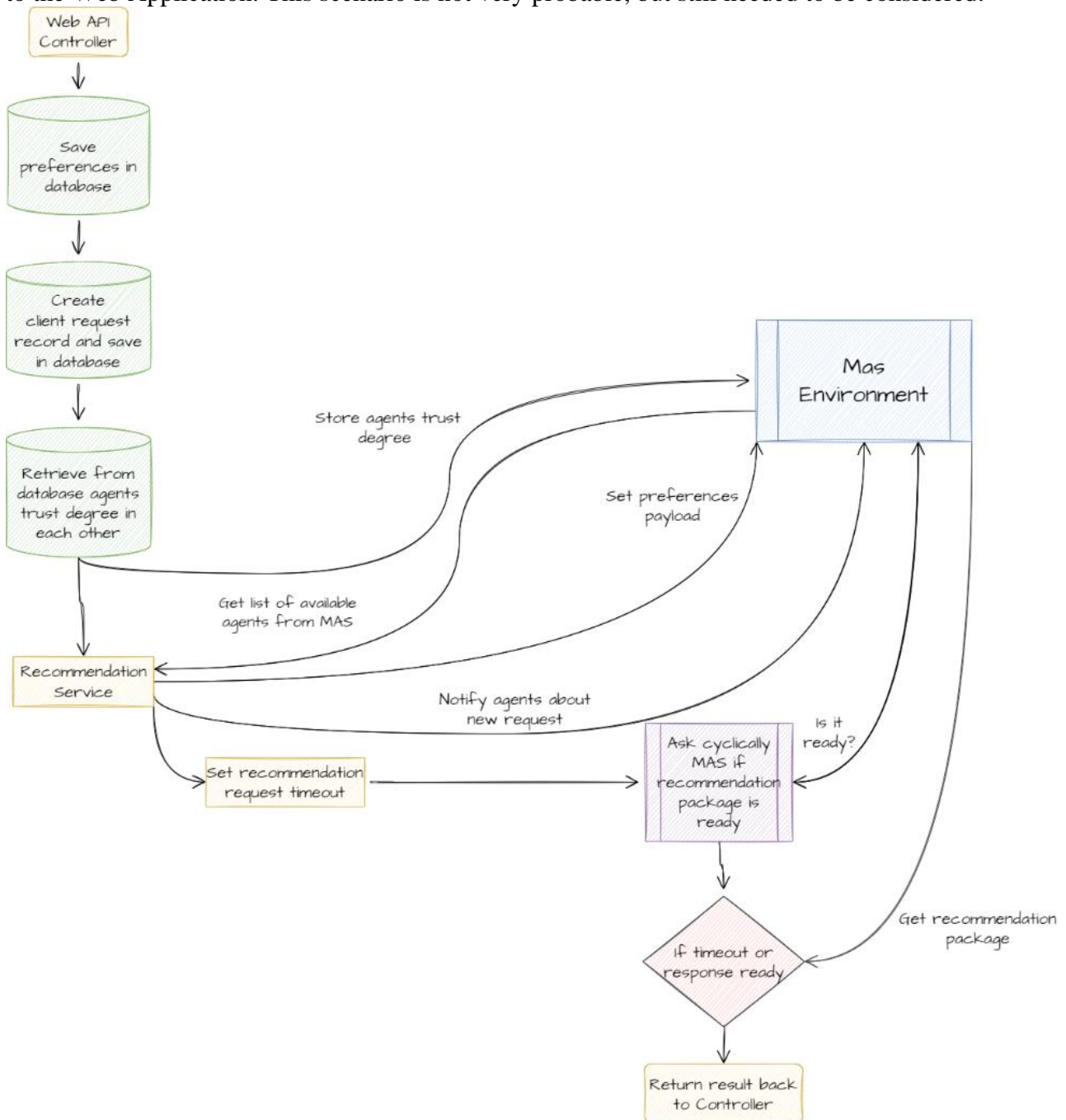


Figure 3.8. Web API receives a vacation package recommendation request

Design of MAS processing a request

As stated, MAS agents have a responsibility to solve a problem, in our case to find a solution for the service they are responsible for. The way agents start solving tasks is simple. The Agent Coordinator notifies the available agents about a new request.

Figure 3.9. shows the logical scheme that an agent follows when it receives a message.

First, the agent chooses the request type:

- a new request from the user
- a help request from an agent

In the first case, the user's preferences for the holiday package are read from the MAS memory. Then the agent subtracts its expert rate for the current user.

The tasks are stored in the MAS memory as a shared resource. Respectively when an agent wants to take a task from there, it must block access to the resource for other agents. The agent will choose a task according to its expert rank and available tasks.

Before finding the optimal service option, if the agent has to handle flights, it will fill in the missing preferences so that the algorithm gives the cheapest result (e.g., flight class to be *Economy*, flight type to be *Direct*).

In the case of *Property* or *Attractions* there is no need to supplement anything in the recommendation package. The agent immediately starts searching his database for an optimal offer. If one has been found, the agent stores the result in the shared memory and notifies the Coordinator that the service type has been completed (e.g., "Dear Coordinator, the result for Property is done."). If no results were found, the agent reads the trust level of other agents and sends a help message to each agent starting with the most trusted.

If the message the agent receives does not refer to a new request, it will most likely refer to a request from one of the agents for help. The message received will only specify the type of service that the other agent was unable to handle. The current agent will check if the task has not already been completed by other agents. If not, it will look for the most optimal recommendation in its database and store it in memory. After completion, the Coordinator will be notified about the completion of a particular task.

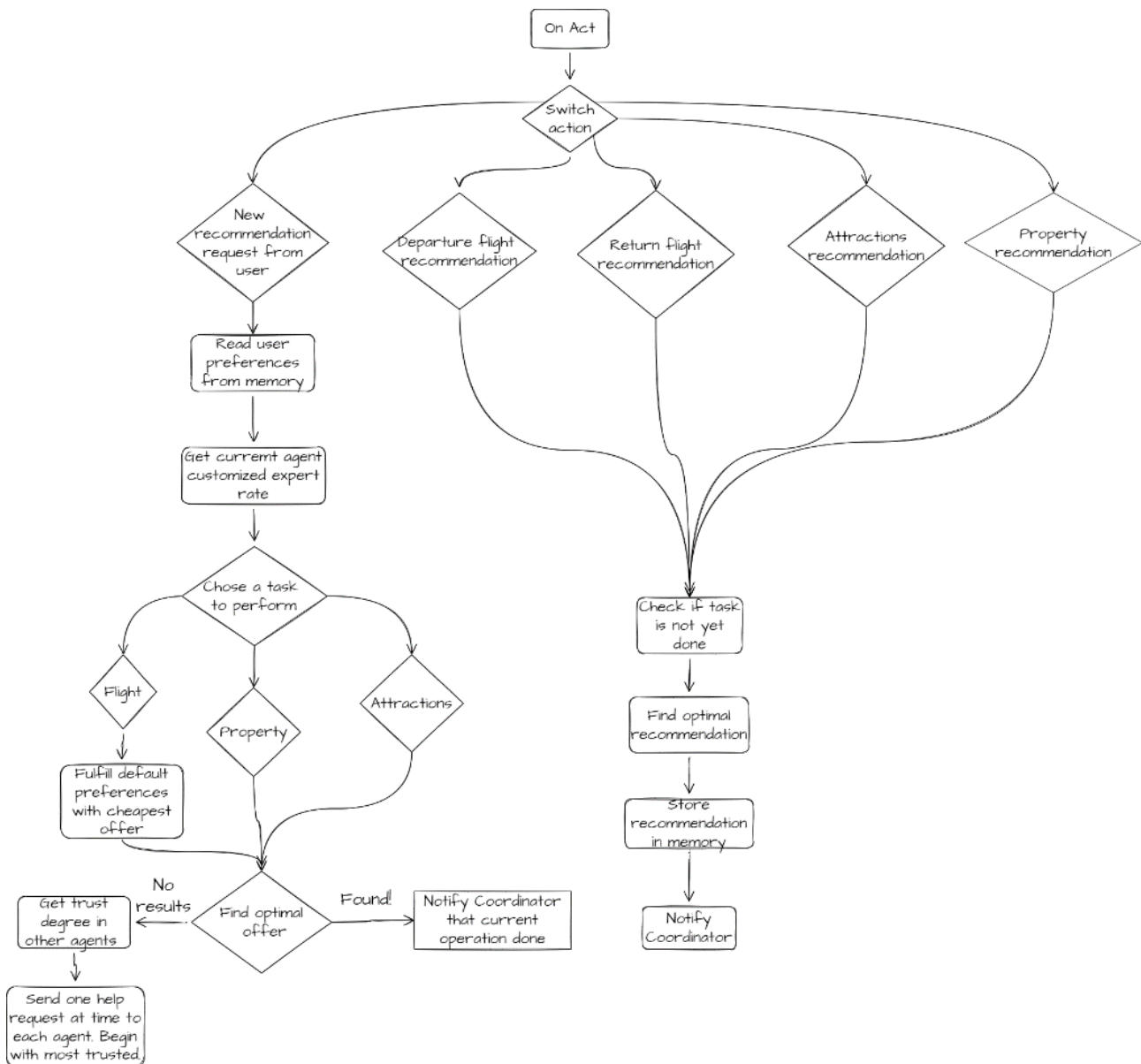


Figure 3.9. MAS agent receives a request

Save evaluation and update agents' rating

An important step that shapes the choice of agents to be experts in a certain service is user feedback. This is the final process that completes the flow of a holiday package request. Every time the user requests a complete holiday package, after receiving it, the user will have to give a rating to the respective service. Obviously, in real life this option would appear after the customer has accepted the option presented and is waiting for the customer to return from the holiday. This way he will be able to give a mark to the services offered.

Figure 3.10. shows how the Web API receives a request to submit a rating of the holiday services delivered by the agents. The request is sent from the GUI. The user fills in some forms and states his liking or disliking of some services.

The API controller receives the request, and the first step is to calculate the final rating of each service, based on the user's ratings. Basically, the final rating of a service is the arithmetic average of all the ratings of all the subservices of that service (e.g., for property will be considered the rating for Amenities, or for Property Type and others will be considered).

The next step is to calculate and update the expert rating of each agent. The ratings given to that agent in the last 30 days will be considered. The older the rating, the more its relevance will decrease and vice versa. By definition, the expert level of an agent is determined by the tasks it has completed and is directly dependent on the ratings it has received from all users.

Another step is the updating of an agent's level of trust in other agents. A recommendation package contains a record for the agent who was first assigned to the task and the agent who completed it. With this data, as well as the customer's evaluation, the algorithm will be able to calculate how many times the help from a particular agent was successful and how many were not.

The personal agent rating is calculated in a similar way to the self-expert rating. The only difference is that only records that refer to data processed by the agent exclusively for concrete client will be considered.



Figure 3.10. Save evaluation process

3.3.3. Database

A database is an organized structure designed to store and process interconnected information. Its use allows you to reduce the load on the server, increasing the speed of data and content pages, as well as increasing site security, which is important.

In the current project was used relational database in PostgreSQL. This particular database system was chosen because it is quite easy to use and it was desired to try a new technology, different from the known ones such as Microsoft SQL Server.

Creating the database design involves knowing exactly what the project aims to achieve. It is quite difficult to create a database that does not require changes to its structure over time.

In the course of the project the database was modified several times until a relatively optimal version was reached. Existing technologies had to be taken into consideration, but also constraints that could affect the structure of the database. For example, the biggest problem was: "Where will agents get their information for recommendations? Will there be an external API that will provide the necessary services, and will agents just send a request to the API and get a response?"

This is where resource constraints and limitations came into play. The simplest way to work with data would have been to find external APIs. For example, the Flight API [24], which provides data about flights and their prices in real time, or the Booking.com API [25] which provides data about apartments and hotels for booking. But APIs that work in real time in a travel domain cost money. Every API call is charged. For this reason, even large companies when choosing the technologies, they will work with, always calculate the price of using foreign services.

The project required the creation of a database table for each type of service: flight, stay properties and tourist attractions. The current structure of relational database can be seen in the *Annexes*.

The design of the reports for flights or properties to stay were made according to an improvised model, which could store the most important information about the respective services (**See in Annexes** *Property relations*, *Flight relations*).

The request of the client for full vacation package was designed in three divisions: preferences, recommendations, and evaluations.

The simplest part was modelling the first section – preferences (**See in Annexes** *Preferences relations*). Here it is quite clear what the user's preferences on already known holiday services might be. Plus, there are already sites that offer flight or hotel booking services, such as *Cheapflights* [26]. It was quite simple to analyze all the preferences a user might exhibit due to sites offering similar travel services. Except that these are based on only one service, whether it's just flying or just staying properties.

The development of the tables for the recommendation process (**See in Annexes** *Recommendation relations*) involved more advanced knowledge of the system itself. At this point the agents in the MAS system come into play. The recommendations are made by them, i.e., in addition to the data about the recommendations, each table containing the recommendation of a type of service also indicates two additional fields: the initial agent responsible for performing the task and the agent who completed the task. Thus, it will be easy to calculate the degree of trust of the agent, who was initially supposed to solve the task but failed, in the other agent, who helped the first one and finished the task.

For the creation of evaluation reports, it is necessary to have the design ready for the recommendation or preference relationships. Because the user will evaluate exactly the services that the agents have proposed as a recommendation.

The realization of the database relationships for the agents of the MAS must be approached separately (**See in Annexes** *Agent relations*). Only after studying the necessary documentation and forming a concrete vision of how the multi-agent system would work, it was possible to create the agent tables. It was necessary to create the agent table itself, but also those of trust in other agents. The expert rate of agents for a specific client also refers to the multi-agent system (**See in Annexes** *Customer relations*).

Chapter 4. The application development

The main embodiment of the final project is realized at coding stage. This is precisely the stage that everyone knows and probably thinks is the only stage in the software development process when developer direct writes code and debugs it. But, as it was observed, this is not the first and not the only stage of software development. The design stage of the project can also be considered a development stage. This is where schematics, pseudocode sketches and even some algorithm tests are made.

However, in addition to applying business logic, there is also scope in the development process to consider ways of optimizing and rectifying concepts developed at the design stage. The design and development phases complement each other. Sometimes the development part might find vulnerabilities in the design scheme or vice versa.

If it has not been done in the design phase, then on the development side there is also the research phase of external services such as APIs or technologies that would optimize the process. Thus, it can be concluded that application development does not only include code and no more.

4.1. Database

For database management PostgreSQL was chosen. It's a fairly easy to use tool, has a friendly interface, but most importantly it supports ACID concepts.

Database generation

For the design and realization of the database it was decided to apply an automatic method to generate the entities and the relationships between them. In order to achieve this, a separate application was created in .NET, which through migrations defines the database schema. *Fluent Migrations* was used as a framework for this. Obviously, the *Entity Framework* could also be used, which is a very popular and powerful tool, but it incorporates too much functionality that is not required at this stage. In addition, *Fluent Migrations* is perfect for frameworks that are not *Object-Relational Mapping*.

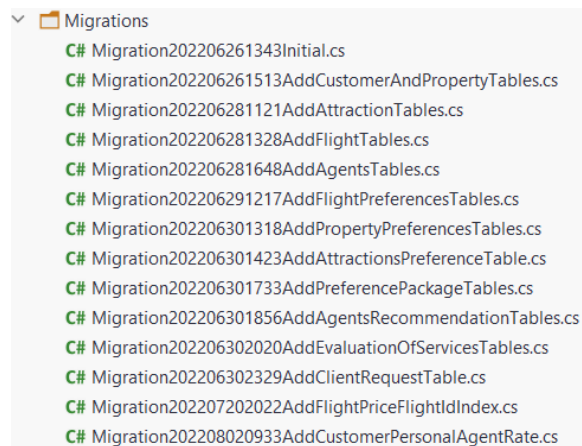


Figure 4.1. Database migrations

Figure 4.1. shows all the migrations that were used to create the database tables, the relationships between them, and the definition for the indexes. As a pattern is used the keyword Migration and then should be inserted the date and time the migration was created, plus a suggestive name. The full name of the migrations is very important, because this way *Fluent Migrator* will know the loading order of all migrations. If the migrations are loaded incorrectly, there is an inevitable risk of creating some database conflicts.

```

namespace DbMigrator.Migrations
{
    [Migration(version: 202206261343)]
    public class Migration202206261343Initial : AutoReversingMigration
    {
        private const string CityTable = "City";
        private const string CountryTable = "Country";
        public override void Up()
        {
            Create.Table(CountryTable) // ICreateTableWithColumnOrSchemaOrDescriptionSyntax
                .WithColumn(name: "Id").AsGuid().PrimaryKey()
                .WithColumn(name: "Name").AsString();

            Create.Table(CityTable) // ICreateTableWithColumnOrSchemaOrDescriptionSyntax
                .WithColumn(name: "Id").AsGuid().PrimaryKey()
                .WithColumn(name: "Name").AsString() // ICreateTableColumnOptionOrWithColumnSyntax
                .WithColumn(name: "CountryId").AsGuid().ForeignKey(CountryTable, primaryColumnName: "Id");
        }
    }
}

```

Figure 4.2. Example of initial migration to database

In the Figure 4.2. you can see an example of a migration. It is a simple C# class that inherits the *Migration* base class or as in our case *AutoReversingMigration*. Both belong to *Fluent Migrator*, the only difference being that *AutoReversingMigration* will know how to revert the data included in the *Up* method, effectively performing the reverse of the actions in this method. The Figure 4.2. example shows the creation of the "Country" and "City" tables. *City* will have a foreign key pointing to the *id* of the table "Country".

After setting up the database connection and configuring the necessary communication services, the migrations are ready to be uploaded.

On running the application, in the database *Fluent Migrations* will create a read-only table that will take the evidence of all the migrations that were already merged. In the Figure 4.3. you will see the order the migrations were uploaded.

	Version	AppliedOn	Description
1	202,206,261,343	2022-08-22 16:53:23.000	Migration202206261343Initial
2	202,206,261,513	2022-08-22 16:53:23.000	Migration202206261513AddCustomerAndPropertyTables
3	202,206,281,121	2022-08-22 16:53:23.000	Migration202206281121AddAttractionTables
4	202,206,281,328	2022-08-22 16:53:23.000	Migration202206281328AddFlightTables
5	202,206,281,648	2022-08-22 16:53:23.000	Migration202206281648AddAgentsTables
6	202,206,291,217	2022-08-22 16:53:23.000	Migration202206291217AddFlightPreferencesTables
7	202,206,301,318	2022-08-22 16:53:23.000	Migration202206301318AddPropertyPreferencesTables
8	202,206,301,423	2022-08-22 16:53:23.000	Migration202206301423AddAttractionsPreferenceTable
9	202,206,301,733	2022-08-22 16:53:23.000	Migration202206301733AddPreferencePackageTables
10	202,206,301,856	2022-08-22 16:53:23.000	Migration202206301856AddAgentsRecommendationTables
11	202,206,302,020	2022-08-22 16:53:23.000	Migration202206302020AddEvaluationOfServicesTables
12	202,206,302,329	2022-08-22 16:53:23.000	Migration202206302329AddClientRequestTable
13	202,207,202,022	2022-08-22 16:53:23.000	Migration202207202022AddFlightPriceFlightIdIndex
14	202,208,020,933	2022-08-22 16:53:23.000	Migration202208020933AddCustomerPersonalAgentRate
15	202,207,011,231	2022-08-22 16:53:42.000	Migration202207011231InsertCountriesCitiesAndAirports
16	202,207,051,250	2022-08-22 16:53:56.000	Migration202207051250InsertProperties
17	202,207,090,012	2022-08-22 16:53:56.000	Migration202207090012InsertFlightCompanies
18	202,207,090,043	2022-08-22 16:54:20.000	Migration202207090043InsertFlights
19	202,207,111,831	2022-08-22 16:54:21.000	Migration202207111831InsertCitiesAttractions
20	202,207,171,148	2022-08-22 16:54:21.000	Migration202207171148InsertStopsPreferenceTypes
21	202,207,211,942	2022-08-22 16:54:21.000	Migration202207211942InsertAgents

Figure 4.3. VersionInfo Table created automatically by Fluent Migrations

That's why on running the database migration application, the *Fluent Migrations* will check the migrations and will pick for upload only those that have not yet been applied. In the Figure 4.4. is presented the situation when was runt the migration application, but all migrations were already applied successfully and there is no work to do.

```

VacationPackage.DbMigrator App has STARTED

Migrations
-----
202206261343: Migration202206261343Initial
202206261513: Migration202206261513AddCustomerAndPropertyTables
202206281121: Migration202206281121AddAttractionTables
202206281328: Migration202206281328AddFlightTables
202206281648: Migration202206281648AddAgentsTables
202206291217: Migration202206291217AddFlightPreferencesTables
202206301318: Migration202206301318AddPropertyPreferencesTables
202206301423: Migration202206301423AddAttractionsPreferenceTable
202206301733: Migration202206301733AddPreferencePackageTables
202206301856: Migration202206301856AddAgentsRecommendationTables
202206302020: Migration202206302020AddEvaluationOfServicesTables
202206302329: Migration202206302329AddClientRequestTable
202207202022: Migration202207202022AddFlightPriceFlightIdIndex
[+] 202208020933: Migration202208020933AddCustomerPersonalAgentRate (current)
VacationPackage.DbMigrator App work done

```

Figure 4.4. Console log of database migration application

Database population

Assuming that is required to create an application that will provide users with vacation packages, one concern would be providing the multi-agent system with a dataset to work with. Preferably this data should be as large as possible, thus highlighting the advantages of a multi-agent system.

The same Fluent Migrations Framework was used to populate the tables. Only this time the migrations will be used for a different purpose - to create new records in the tables. But the first thing that was needed - finding the data or generating it.

For example, for properties to stay the database belonging to Airbnb [27] was chosen. From this database the names of some properties and their type were copied: hotel, house, apartment. All data were inserted into a .csv table. The other properties that are required to be indicated for this service in the database were randomly generated using *Kutools for Excel* [28]. It is an extension that increases the possibilities of Excel.

Creating flights and their schedules was more difficult. Here routes have to be created that are bi-directional, i.e., if the customer flies into a city, they have to have a flight company to fly back with and in addition diversification and multiple data is indispensable. It should also be taken into account that a route can fly several times a week and several times a day. *Kutools* was used to generate this data as well.

In order to have some data that is at least a little closer to reality, it was decided to find all passenger airports in Europe and create routes from these airports. Data from *Airportcodes.io* [29] was used as a resource for airport data. As in case of properties, data for the flights were stored in a .csv file.

The *OpenTripMap API* [18] was used to find tourist attractions in a concrete city. Obviously, if needed, agents could have called the API themselves and asked for the necessary attraction recommendations. But it was desired to keep the pattern of previous services, i.e., populating the databases with records. For this reason, a script was created to take the API data about the tourist attractions in the cities where our "multinational company" has airports and insert them into a .csv file.

After all the .csv data files for the three services: flight, properties, and attractions, were ready, the script for populating the database with values was implemented. For retrieving data from files and converting them to C# database models was very helpful the *CsvHelper* [30]. Thus, after reading from the .csv files, *Fluent Migrator* uploads the records to database using same approach of uploading migrations.

4.2. Data tier

The data level involves the implementation of business logic. As thought at the design stage, the basic implementation will be done on the server side.

ASP.NET Core Web API with platform on .NET 6 was used to create the server design. *.NET* was chosen as the application programming environment because it is modern and is in continuous

development. Another plus is the presence of the documentation. Microsoft likes to detail in depth the services its platform offers.

The design phase offered the possibility to create logical schematics that the implementation phase will develop in code. The first step of the implementation was the creation of the application layouts that would follow the "Onion architecture". After that it was necessary to configure the processes that will take place at startup (adding services, configuring the database connection, and loading the multi-agent system).

An interesting feature of this project was the implementation of the Dependency Injection design pattern. It was used in the services so that they do not depend on some hard-coded objects but can always be replaced with others. Plus, Dependency Injection fits perfectly with the architecture of the project, making it easy to test and validate functionality.

The design phase offered the possibility to create logical schematics that the implementation phase will develop in code. The first step of the implementation was the creation of the application layouts that would follow the Onion Architecture. After that, it was necessary to configure the processes that will take place at startup (adding services, configuring the database connection, and loading the multi-agent system).

An interesting feature of this project was the implementation of the Dependency Injection design pattern. It was used in the services so that they do not depend on some hard-coded objects but can always be replaced with others. Plus, Dependency Injection fits perfectly with the architecture of the project, making it easy to test and validate functionality.

Figure 4.5 shows an example of dependency injection from the project. Repositories are used to handle database operations. Thus, if is required to use a database resource, should be created the repository service that is strictly responsible for that process, define its interface and inject it into the required service. In this way, the services in the *Application* layer will have access to those belonging to the *Repositories* layer (see description in Chapter 3.2.2. Data tier) through *Dependency injection*.

```

private static void ConfigureServices(IServiceCollection services, IConfiguration configuration)
{
    // Add services to the container.
    services.AddScoped<IAgentService, AgentService>();
    services.AddScoped<IPreferencesPackageService, PreferencesPackageService>();
    services.AddScoped<IFlightService, FlightService>();
    services.AddScoped<IPropertyService, PropertyService>();
    services.AddScoped<IAttractionService, AttractionService>();
    services.AddScoped<IMasLoaderService, MasLoaderService>();
    services.AddScoped<IRecommendationService, RecommendationService>();
    services.AddScoped<IEvaluationService, EvaluationService>();
    services.AddScoped<IPreferencesPayloadInitializerServices, PreferencesPayloadInitializerServices>();
    services.AddScoped<ICustomerService, CustomerService>();

    services.AddScoped<IPreferencesPackageRequestRepository, PreferencesPackageRequestRepository>();
    services.AddScoped<IFlightRepository, FlightRepository>();
    services.AddScoped<IAgentRepository, AgentRepository>();
    services.AddScoped<IPropertyRepository, PropertyRepository>();
    services.AddScoped<IAttractionRepository, AttractionsRepository>();
    services.AddScoped<ICustomerRepository, CustomerRepository>();
}

```

a)

```

public class PropertyService : IPropertyService
{
    private readonly IPropertyRepository _propertyRepository;

    public PropertyService(IPropertyRepository propertyRepository)
    {
        _propertyRepository = propertyRepository;
    }

    public async Task<List<PropertyBusinessModel>> GetAllPropertiesAsync()
    {
        return await _propertyRepository.GetAllPropertiesAsync();
    }
}

```

b)

Figure 4.5. Dependency injection of the *Property Repository* a) in the *Property Service* b)

The core business logic implementation is focused on the multi-agent system. All requests for a complete holiday package arrive in MAS. The steps outlined at the design stage have been fulfilled exactly, but there were also some implementations that contributed to adjust the process of finding an optimal recommendation by the agents.

Similarity match with user needs

When agents receive a set of preferences on the holiday package, they should somehow check their knowledge base for the best match. But the idea of "best match" can be controversial. One thing is for sure, it should benefit the client as much as possible.

Ensuring a good result for the customer would mean knowing their general preferences. Somehow it would imply the need to create a profile of the customer, and to each call from the customer, the agents could give a response based on the preferences of the profile created. This implementation can be achieved using ML.NET [31] which involves the use of machine learning. This framework uses *matrix factorization* [32] for training the data and estimating a suitable recommendation.

The project presented in this paper tended to present a startup version of a project. Thus, ML.NET can be used in the future as an extension of the functionalities and increase the efficiency of the agents' results.

A simplified variant of finding a best match was the use of the method presented in the paper "*Improving recommendations through an assumption-based multiagent approach: An application in the tourism domain*", chapter 3.7.1. *LocalSearch*, equation 3 [5]. In this resource a very simple way of comparing the similarity between two entities is presented - the application of the *Euclidean Distance*. In simple terms it checks how many matches there are between user preferences and potential recommendation. The higher the number of matches compared to the number of unmatches, the more the recommendation tends to be accepted and sent to the user.

Update agents' ratings

Within the multi-agent system, agents choose a task that most closely matches their expert grade for that service. As explained in the design chapter, the rating of agents depends solely on customer feedback.

After each request of a vacation package, the user will have to fill in a form with his evaluations about the services provided. From the UI application the payload will be sent to the API or actually server. This is where the final rating of each type of service, i.e., flight, property, and tourist attraction, is first calculated. The formula is simple, the final rating given by the client to a specific service is the arithmetic average of all the subservices in that service. For example, for tourism attractions, each attraction will receive a rating of "Liked" or "Didn't like". The server will read the data of all ratings for the attractions and calculate the average of these values. This will be the final rating of the tourist attraction service.

So logically, the agent's rating for a given service will be conditioned by the last task he completed, i.e., exactly that final calculated rating of that service. The formula would be simple, take all the ratings the agent has received for that service and calculate the average.

But here come two problems with this formula. The first is loading too much data, and the second is the irrelevance of old data. People change their preferences over time. Why should the agent's rating depend on all the preferences the customer has ever shown? The best solution would be to consider the freshest feedback from the customer.

Not in great detail, but this principle has been discussed in "*Improving recommendations through an assumption-based multiagent approach: An application in the tourism domain*", chapter 3.5. *Agent specialization and trust* [5]". Here a formula for calculating the trust degree is presented.

$$R = q * \frac{e^{-r*t}}{e^{-r}} \quad (1)$$

This formula has been readapted for use in the current project. Thus, in equation (1) is presented the formula for calculating the relevance rate of an evaluation where **q** - is the final rating, **r** - is the relevance coefficient and **t** - is the number of days elapsed since the date of calculation of the final rating (i.e., since the date of the user evaluation).

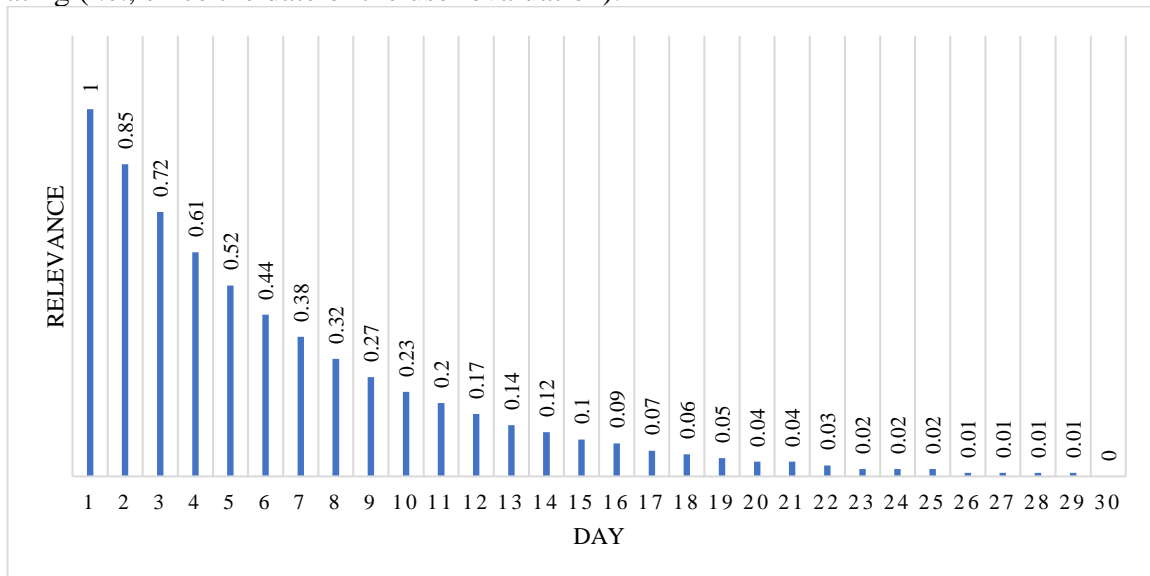


Figure 4.6. User evaluation relevance in 30 days cycle

The coefficient of relevance is a constant, in this project the value 0.16 was considered. This value was not chosen randomly, but several simulations were run with the consideration that the maximum lifetime of the evaluations would be 30 days. This means that the current day's rating should have maximum relevance and the rating given should not be altered, and the 30-day old rating should be completely irrelevant, i.e., null. In Figure 4.6. the degree of relevance of customer feedback is demonstrated. This formula was applied to calculate the agent's expert level for each vacation service (this includes the general and personal expert rate).

Optimization of API's response time

Since in the current project is used a REST architecture, comparable to the WebSocket architecture, must also be considered the execution time of the whole recommendation process. HTTP calls have a set time to wait for a response. If the server does not respond in time, the request will be cancelled, and an error message will appear.

Because of this, it was also demonstrated at the design stage that there is a timeout for waiting for the recommendation imposed on the MAS system. That timeout was just an assurance that the server would be able to provide a response, but it was also necessary to make a decision to optimize the server response time itself. This was achieved by applying the "*fire and forget*" principle. That is, triggering some functionalities without having to wait for a response from them. For example, at the end of a request to provide a recommendation, information about that recommendation is stored in the database. This is not vital for providing a response to the client. Thus, in order not to take time away from the response time to wait for the database save, this process is invoked without waiting for something to result. Such an approach is quite risky because this process is uncontrollable and may create some conflicts. For this reason, when implementing "*fire and forget*" a good assurance must be given that the data always reaches its destination.

4.3. Presentation tier

Front-end

The implementation of the user interface involved finding a GUI template that would fit the main purpose of the project - data entry functionality for a complete holiday package. Thus, the Colorlib's Travelix template [33] version was chosen...

After adjustments and redesigns, the search page for a holiday package looks like attached Figure 4.7. The recommendation pack is only five mandatory details that the user must indicate: departure and destination city, check in date and period of the holiday and how many adults will fly. The other details are absolutely optional. But the more preferences the user gives, the closer the recommendation will be to what customer wanted from the start.

Figure 4.7. Graphical interface of vacation preferences

A special feature of the holiday package search GUI is the loading of certain data from the database before sending the request to receive a recommendation. So, it appears that yes, the user makes requests to the server, probably without being aware of it, even before clicking on the Search button.

The list of starting cities is built from the moment the page loads. An HTTP request is made to the server asking for the cities data. But only when the user selects a specific city from which he wants to leave, an HTTP call is sent to the server to load the available cities as destination. These are found by the available flights from the start city.

Figure 4.8. Selecting Departure and destination City

The result of a request for a complete holiday package is shown in figure 4.9. Data are presented for each type of service, the data being as close as possible to the user's preferences.

Flight recommendation

Departure

Flight date: 07-10-2022

Departure time: 11:16

Stops: Direct

Class: Economy

Company: Azimuth

Airport: Mostar International Airport

City: Mostar

Country: Bosnia and Herzegovina

Property recommendation

Property name: cozy, secure, new queen mattress, great location**

Pets: no

Amenities:

1. Wi-Fi: no
2. Kitchen: yes
3. Washer: yes
4. Dryer: yes
5. AC: no

Top attractions

Name: Capela familiei Oganowici

Address: , MD2001, Chişinău, Moldova

Rate: 3

#religion #other_temples #interesting_places

Name: Nicolai Costenco

Address: , 2001, Chişinău, Moldova

Rate: 3

#historic #monuments_and_memorials #interesting_places #monuments

Figure 4.9. Recommendation result displayed on GUI

In Figure 4.10 is shown the form that the user should fill in when returning from holiday. This is the feedback that the user sends to the server to appreciate the work of the agents. This is where the calculation of the agents' ratings and their training as experts starts.

Departure flight

Did you enjoyed the flight class?

Yes No

Did you enjoyed the flight company?

Yes No

Was the departure flight date acceptable?

Yes No

Did the flight departure time matched your needs?

Yes No

Next

Figure 4.10. Evaluation of services formular

Another important interface is the login and registration page. This was necessary to create a user in the database, which each time would receive personalized results according to its evaluations. This is also directly related to the creation of agent ratings as "personalized agents", i.e., the same agents with only one rating for each type of service customized by user feedback.

Figure 4.11. GUI for login and registration

Back-end

For the back end of the GUI, schematic diagram logic from the design section was used to implement the code. One feature that was not conceived at the design stage was finding the most efficient way to retrieve the UI data entered by the client.

If Ajax had sent all the data entered by the client in a single call to the Controller, it would have been much more difficult to parse that payload. Thus, the decision was taken to send separate calls containing certain properties of the vacation services the user wanted (e.g., selected preferred flight companies, part of the day to fly most preferred and others).

```
<div id="btnSearch" class="button search_button">
  search
</div>
```

a)

```
$(document).ready(function(){
  $('#btnSearch').click(function(){
    var periodNumber = $("#periodNumber").val();
    var adultsNumber = $("#adultsNumber").val();
    var departureCity = $("#departure").val();
    var destinationCity = $("#destination").val();

    if (periodNumber !== "" && adultsNumber !== "" &&
        departureCity !== null && destinationCity !== null)
    {
      $.ajax({
        type: 'POST',
        url: '/Flight/StoreVacationPeriod/' + periodNumber,
        success: function (response) {
          // code on success
        },
        error: function (response) {
          // code on failure
        }
      });
    }
  });
});
```

b)

```
[HttpPost(template: "{action}/{days}")]
public void StoreVacationPeriod(short days)
{
  PreferencesPayloadSingleton.Instance.HolidaysPeriod = days;
}
```

c)

Figure 4.12. On HTML interface a) the client pushes the Search button. The jQuery handles the button click event b). Here using Ajax is sent a POST request to the MVC Controller c)

In Figure 4.12. a small part of the processes that take place when pressing the "Search" button is shown. In jQuery the values of the data boxes that the user has filled in are read and then sent to the Controller. In each Controller, the *Singleton* instance of the preferences package will add the information received from the request to its own container. After sending all the necessary information from AJAX, the last call from here will be the one that will signal that the preferences package has been completed with the data from the UI and can be sent to the server.

Chapter 5. Application testing

After the code is written, the programmer must debug his code to make sure there are no bugs in it.

Component testing

After the code is written and debugging is performed, it is necessary to test the implemented functionality. If the program consists of several components, each component is tested separately at first, since very large programs include huge functionality, which is often divided into separate components, which are developed separately. In smaller projects, this step may simply include testing of individual classes.

API testing

Component testing took place in various methods. As an example, for the testing of the server application it was necessary to create a *Console Application* that simulates the entire request flow of a complete holiday package, following these steps:

1. Create record of a new user in database directly (It's done manually, not from Console App)
2. Get the list of departure cities for vacation
3. Select a random city of departure
4. Get the list of arrival cities, considering previous departure city
5. Select a random city of arrival
6. Get flight companies that can bring customer from departure to destination
7. Select some of them to store as preferred ones randomly
8. Get flight companies that can bring customer back home
9. Select some random to store as preferred for return flight
10. Fulfill preferences package with required data. This means that the fields user id, departure date, holiday period and number of people who want to travel must be filled in. Otherwise, the other data are not mandatory, but can also be entered to test the correctness of the interpretation of the data by the MAS system, which should be able to take into account every preference (e.g., preferred flight companies).
11. Send preferences package as a payload of a HTTP POST request to server
12. Read recommendation response from API
13. Fulfill evaluation of the API response
14. Send it to API through HTTP request

These steps are sufficient to follow the workings of the multi-agent system. But interpreting the data is more difficult. It is quite hard to interpret a JSON payload when it has a large amount of data. Because of this, a simplified system for logging the most important data has been created in the API.

With each customer call, a file is created on local disk in *Log* folder, that keeps the user's registered preferences, the recommendations offered, the agents assigned to a recommendation service and those who actually made the recommendation. And if this will not be the first call, information about the agent rating updates for each service, a table with how many recommendations each agent has completed for each service separately (for the flight being two recommendations, one for the outbound and one for the return flight will be considered as two) and another table with the personal agent rating of each agent for each service will also be visible.

Personal agents services score				
		FlightScore	PropertyScore	AttractionsScore
	Agent Bob	1	1	0
	Agent Homer	1	0	1
	Agent John	2	1	1

Personal agent rates				
		FlightRating	PropertyRating	AttractionsRating
	Agent Bob	0.5	0.66	0
	Agent Homer	0.5	0	1
	Agent John	0.62	1	0.25

Figure 5.1. Example of a sequence from a log file

Also, for a better tracking of the interactions between agents, the messages the agents receive or send are recorded in a text document, as well as on the local disk. These messages are sent directly from the multi-agent system. An example is presented in the Figure 5.2.

```

Agent Bob got request
Agent Homer got request
Agent John got request
Agent Homer got task type Property
Agent John got task type Attractions
Agent Bob got task type Flight
Agent Homer send to Agent Bob property_recommendation_request
Agent John send to Coordinator attraction_recommendation_done
Agent Homer send to Agent John property_recommendation_request
Agent Bob send to Agent John departure_flight_recommendation_request
Agent Bob send to Agent Homer departure_flight_recommendation_request
Agent Bob send to Agent John return_flight_recommendation_request
Agent Bob send to Agent Homer return_flight_recommendation_request
Agent John send to Coordinator property_recommendation_done
Agent John send to Coordinator departure_flight_recommendation_done
Agent John send to Coordinator return_flight_recommendation_done

```

Figure 5.2. Example of logging agents' interaction

It is impossible to create integration tests for the entire preferences request flow of a complete holiday package. It is true that on each API project load, agents will have the same knowledge base, but it must be taken into consideration that the MAS algorithm, when it comes to no recommendation and has several choices that are equally good, takes one randomly. Thus, from an extremely large amount of data, cannot be concluded exactly which solution the agent will choose. In addition, when a new client appears in the system, it is logical that the rating of custom agents is zero, i.e., it is not known which agent will take a particular service. What remains is the analysis of the data in the log files and possibly, if necessary, the analysis in the database.

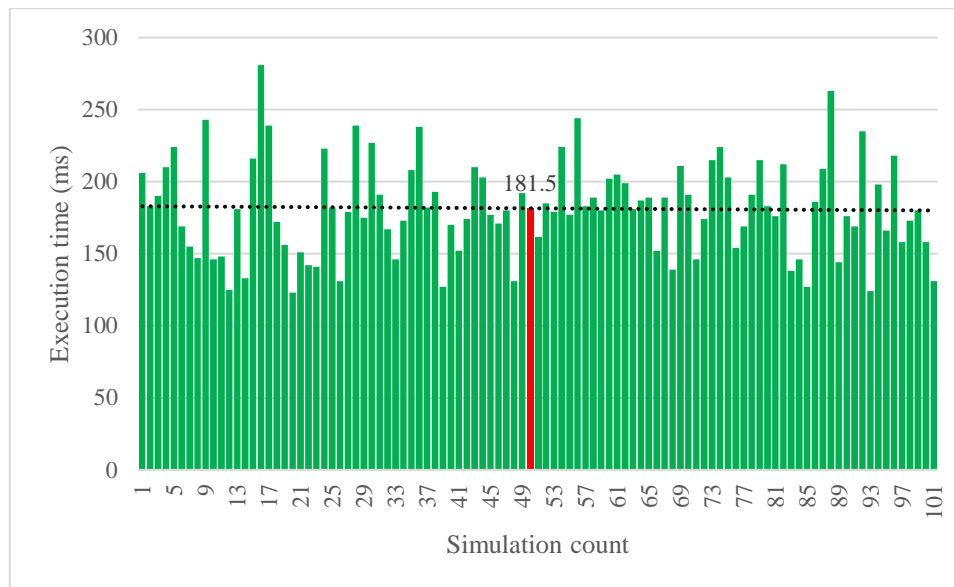


Figure 5.3. Time required to receive a full vacation package from API

Since a REST architecture is used, the request processing time for a recommendation has to be considered. In Figure 5.3. the resulting time for one hundred requests of a complete holiday package is shown.

The average waiting time for a result is approximately 181 milliseconds. Which is a pretty good result. If processing time adjustments were needed, then the place of focus would be the multi-agent system. The easiest example would be the application of search algorithms such as "*Binary Search*". Perhaps even implementing an agent-based algorithm to search more efficiently across a large collection of data.

Web application testing

Specific methods were used to test the web application. For the back end, was used POSTMAN which sent HTTP calls to the MVC Controller of the application. This way it was possible to check the correctness of the data transmission, the response code, and its body.

For the front end, it has been verified that all data entered by the user are correctly sent to the back end. Another way of testing and debugging was to use the development tools offered by browsers (e.g., Microsoft Edge or Google Chrome). Thanks to them a programmer can see the front-end errors of the web application, but also debug data from JavaScript.

However, there are also manual tests that do not require a setup or the use of other tools. For example, when logging in, if the user enters the wrong data, a warning message will be displayed.

Login

Email:
someWrongEmail@mail.com

Password:

Login

Wrong email or password

Don't have an account? [Sign Up!](#)

Figure 5.4. User enters wrong login or password and gets error

Or another example of testing would be trying to access the main page that offers vacation services to the user without being logged in. As shown in Figure 5.5, the front-end template used from *Codepen.io* [34] will display an error page.

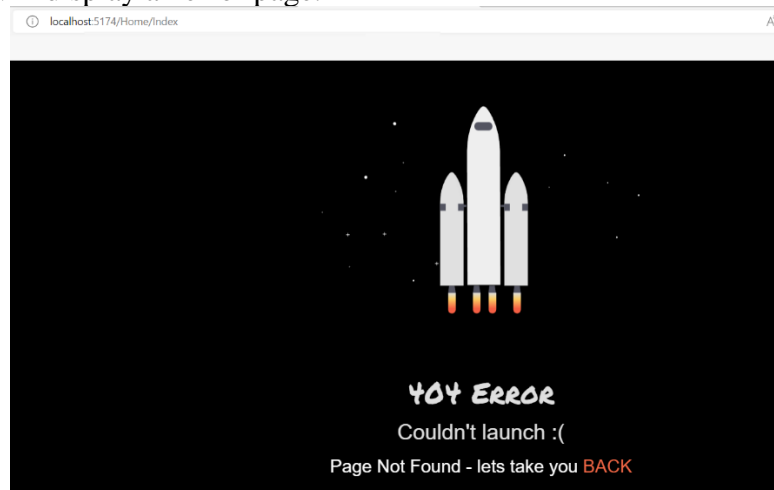


Figure 5.5. Access the vacation services page without being logged in

Component integration

When testing of all the components is complete, can be moved on to the integration of all the components into a single software package, this stage is exactly the process of integration, i.e., merging of all the components into a single system.

Testing of the entire system

At this stage, testing of the entire system is carried out, considering the integration of all components. At this stage, developer can identify problems with the interaction of components and fix them. Also at this stage, the main subject of testing is security, performance, resource leakage and other things that cannot be tested at lower levels of testing. The programmer must ensure that the data packet sent from the web application will arrive in a correct format to the API, and from there the data will be stored in a secure and correct way in the database.

Chapter 6. Demonstration and scenarios

For a better vision, some scenarios of use of this application will be exemplified. To start with, a new user account will be created. This way it will be easier to follow how the agent's ratings evolve and how the system behaves with new income. A new user is created in the dedicated page for this service as shown in Figure 6.1.

Join Us!

First name:
Demo_First_Name

Last name:
Demo_Last_Name

Login:
Demo_Login

Email:
Demo_Email@email.com

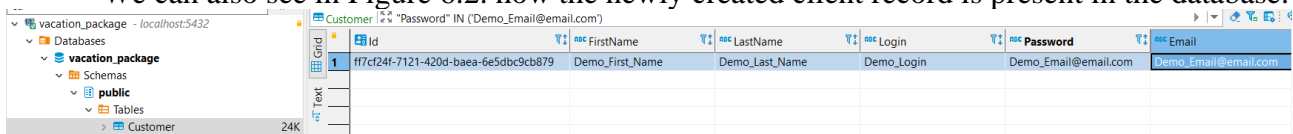
Set Password:

Sign Up

Already have an account? Login!

Figure 6.1. Register new user for demonstration

We can also see in Figure 6.2. how the newly created client record is present in the database.



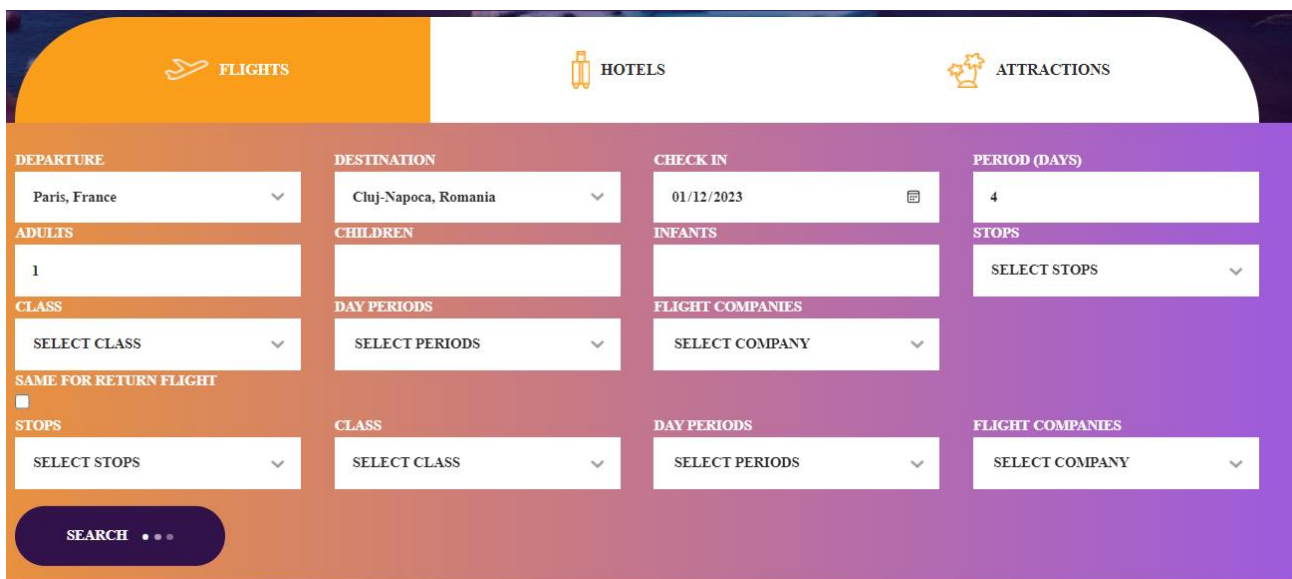
The screenshot shows a database management interface with a table named 'Customer'. The table has the following columns: Id, FirstName, LastName, Login, Password, and Email. The first row contains the following data: Id: ff7cf24f-7121-420d-baea-6e5dbc9cb879, FirstName: Demo_First_Name, LastName: Demo_Last_Name, Login: Demo_Login, Password: Demo_Email@email.com, Email: Demo_Email@email.com.

Id	FirstName	LastName	Login	Password	Email
ff7cf24f-7121-420d-baea-6e5dbc9cb879	Demo_First_Name	Demo_Last_Name	Demo_Login	Demo_Email@email.com	Demo_Email@email.com

Figure 6.2. The new registered user in the database

First scenario – minimal number of preferences in one request

In Figure 6.3. the customer enters a minimum set of data required to receive a complete holiday package.



The screenshot shows a vacation package search form with the following fields:

- DEPARTURE:** Paris, France
- DESTINATION:** Cluj-Napoca, Romania
- CHECK IN:** 01/12/2023
- PERIOD (DAYS):** 4
- ADULTS:** 1
- CHILDREN:**
- INFANTS:**
- STOPS:** SELECT STOPS
- CLASS:** SELECT CLASS
- DAY PERIODS:** SELECT PERIODS
- FLIGHT COMPANIES:** SELECT COMPANY
- SAME FOR RETURN FLIGHT:** ☐
- STOPS:** SELECT STOPS
- CLASS:** SELECT CLASS
- DAY PERIODS:** SELECT PERIODS
- FLIGHT COMPANIES:** SELECT COMPANY

A **SEARCH** button is located at the bottom left.

Figure 6.3. Data entered for searching a complete vacation package

And as can be seen in *Figure 6.4*, we got a result with some data that corresponds to the preferences previously exposed. More precisely, the departure and destination cities coincide, and the holiday period is exactly the right one. In the case of default values, i.e., values on which the user has not expressed a preference, the agent should take into account the variant that will be cheaper as a result. Within this project, no price of services (flight, accommodation, or attractions) was indicated. Thus, in the current scenario, agents provide the first result that corresponds to the user's requirements.

Flight recommendation

Departure

Flight date: 12-01-2023
Departure time: 02:07
Stops: Direct
Class: Economy
Company: Air France-KLM
Airport: Paris-Charles de Gaulle Airport
City: Paris
Country: France

Return

Flight date: 16-01-2023
Departure time: 22:49
Stops: Direct
Class: Economy
Company: Air Serbia
Airport: Cluj-Napoca International Airport
City: Cluj-Napoca
Country: Romania

Property recommendation

Property name: PrivateRoom - SunCity and Spacious - City Left
Pets: no
Amenities:

1. Wi-Fi: no
2. Kitchen: yes
3. Washer: yes
4. Dryer: yes
5. AC: yes
6. Heating: no
7. TV: no
8. Dish: yes

City: Cluj-Napoca
Place Type: PrivateRoom
Property Type: House
Bedrooms: 1
Bed: 1
Bathrooms: 0

Top attractions

Name: Babos Palace, Cluj-Napoca
Address: , 400036, Cluj-Napoca, România
Rate: 3h
#palaces #architecture #historic_architecture #interesting_places

Name: House of Religious Freedom
Address: , 400105, Cluj-Napoca, România
Rate: 3h
#historic_architecture #architecture #interesting_places #other_buildings_and_structures

Name: Palace of Justice
Address: , 400117, Cluj-Napoca, România
Rate: 3h
#historic_architecture #architecture #interesting_places #other_buildings_and_structures

Name: Institutul Teologic Protestant
Address: , 400116, Cluj-Napoca, România
Rate: 3
#religion #other_places #interesting_places

Figure 6.4. Result of preferences for the scenario with basic preferences

The next step will evaluate the results provided by the agents. This will directly affect their rating. For this, we will also give some negative votes. The steps for calculating the ratings were presented in Chapter 4, section “*Update agents’ ratings*”. The agents who handled the *flight* have 100 percent positive feedback. Which means their level of expert will increase. In the case of *property*, it will be considered that the agent didn't satisfy the customer at all. Thus, his property expert rating will drop (if it is not already at the lowest possible limit, i.e., 0). In the case of tourist attractions, the agent will receive a partial rating increase (reported to 0.75 value which is mean of *Attractions* ratings).

a) Departure flight

Did you enjoyed the flight class?
Yes No

Did you enjoyed the flight company?
Yes No

Was the departure flight date acceptable?
Yes No

Did the flight departure time matched your needs?
Yes No

b) Return flight

Did you enjoyed the flight class?
Yes No

Did you enjoyed the flight company?
Yes No

Was the return flight date acceptable?
Yes No

Did the flight return time matched your needs?
Yes No

c) Property

Did you enjoyed the property type?
Yes No

Did you enjoyed the place type?
Yes No

Are you ok with rooms and beds?
Yes No

Did you liked the amenities?
Yes No

d) Attractions

Did you enjoyed the Babos Palace, Cluj-Napoca?
Yes No

Did you enjoyed the House of Religious Freedom?
Yes No

Did you enjoyed the Palace of Justice?
Yes No

Did you enjoyed the Institutul Teologic Protestant?
Yes No

Figure 6.5. User's evaluation for basic scenario

In *Annex 6* is presented the file where all the data was logged in detail. At this stage can be ensured that the data has been correctly read from the user and transmitted to the server. Also, in the *Annex 7* is presented other log file that logged the communication between agents.

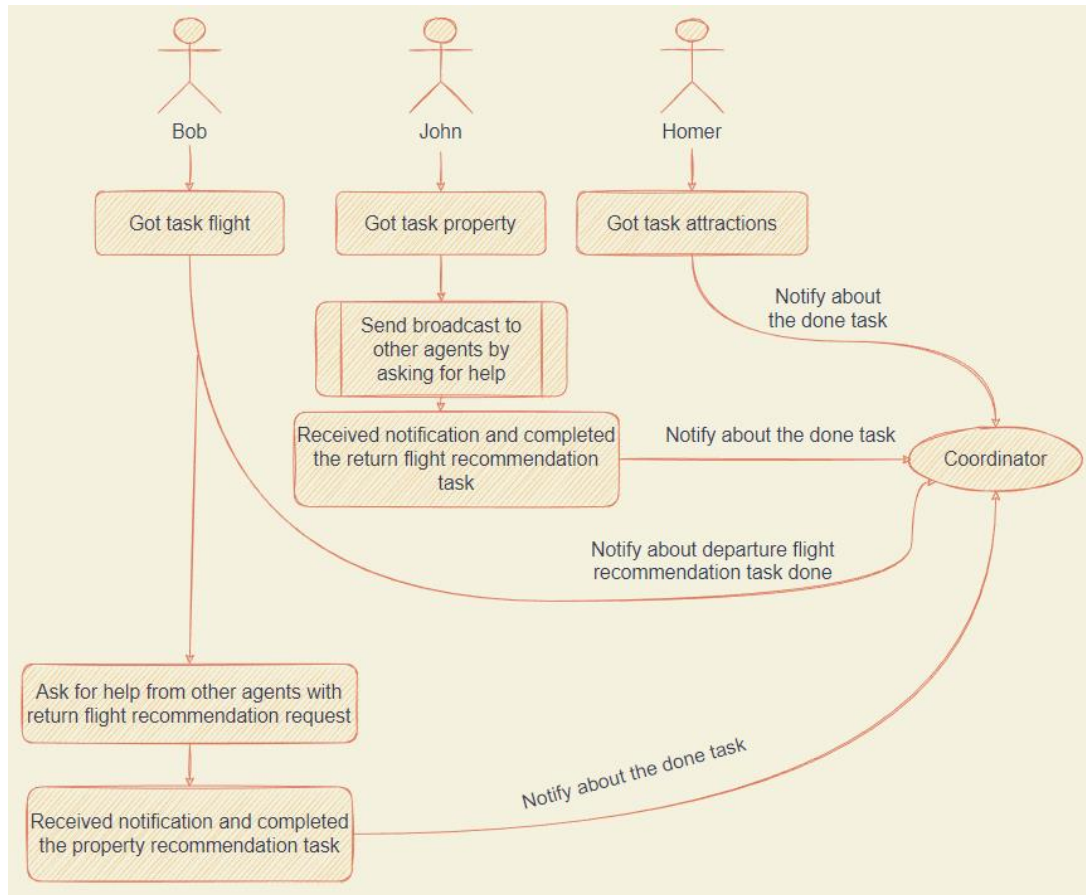


Figure 6.6. Demonstration of agents' communication

In *Figure 6.6.* is presented the communication schema between agents, that was interpreted based on *Annex 7*. Will be considered 3 travel agents who will work on the server side to provide recommendations to the client (Bob, John, and Homer). On user request, each agent gets a type of service to handle. If one of them has not been able to find any records that might be useful to the user, they ask for help from their colleagues. Also, when an agent receives a request for help, he will have to grant it in a short time, otherwise another agent will complete the task. Each interaction between agents also establishes the trust between them.

For example, in our case Bob receives a help request from John, manages to complete it in time and sends the response to the Coordinator. In this case, after the user has given feedback on the Property services, if the average value of the services is greater than 0.5 then John will increase his trust rating on Bob by one point. From *Figure 6.5. c)* we can see that the feedback for the property service is completely negative. That will decrease the trust degree of John in Bob's for this type of service.

Second scenario – maximal number of preferences in one request

Another case that could be taken into consideration is the one where the user fully exposes his preferences in one call. Thus, in addition to the preferences chosen from the first scenario, now the others will be added. In *Annex 8*, the most relevant data from the request are presented. In *Annex 9* is the full log that has been created after the request has been completed.

Table 6.1. shows how many of the parameters provided match the user's requirements. For example, departure flight companies do not match or property type. Cells marked with green mean perfect match. The yellow ones are partial, and the red ones are a mismatch. If to calculate how many

percent of the result matched, we would have 14 out of 21 properties matched, or 66.6%. This result cannot be considered to be a bad one. Agents operate with the data they have. The idea is to find a compromise between the data the client asks for and the data held by the agent. Even if it will not be exactly 100% what the client wants, the aim is to provide as close an offer as possible.

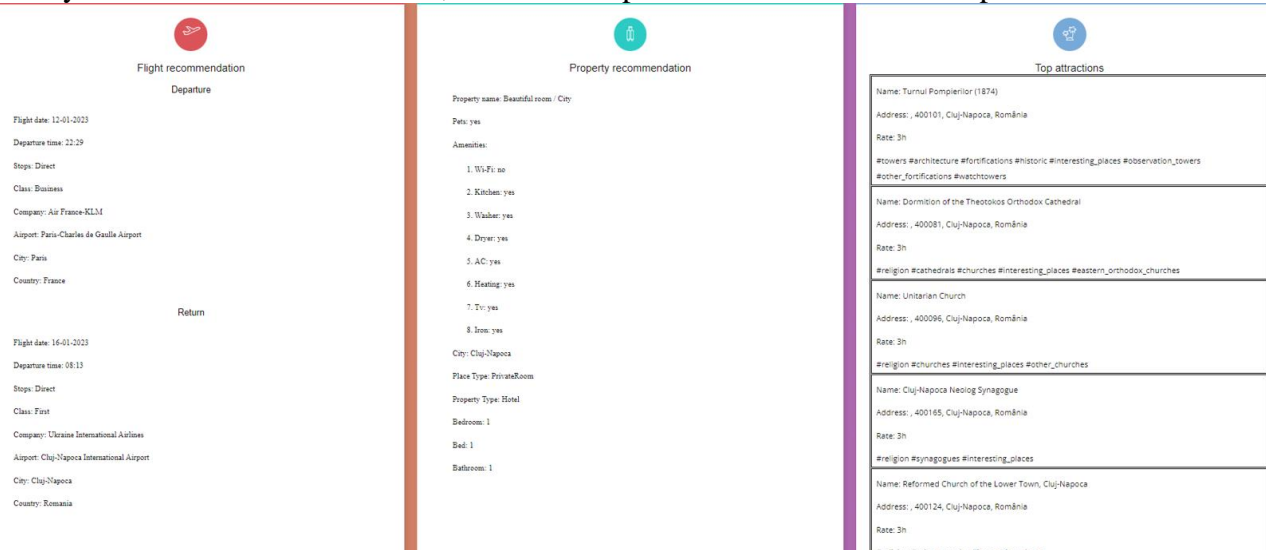


Figure 6.7. Recommendations for second scenario

The agent, which is considered "custom", has a set of data with which it operates, and which is defined solely by feedback from the customer. If at some point the customer gives a negative rating to the agent for the service provided, there is a chance that the agent's rating will drop to such an extent that his dataset will be considered insufficiently good (i.e., his expert grade will be lower than his colleagues) and the opportunity to perform that service will fall to others.

Generic Data	Departure Date	12-01-2023
	Holidays Period	4
	Departure City	Paris
	Destination City	Cluj-Napoca
Departure Flight	Companies	- Red Wings Airlines
	Day Periods	- Night
	Class	Business
Return Flight	Companies	- Ukraine International Airlines
	Day Periods	- Morning
	Class	First
Property	Property Type	- Guest House
	Place Type	- Private Room
	Room and Beds	Bedrooms: 2 Beds: 1 Bathrooms: 1
	Amenities	- Washer - Air Conditioning - Heating

Attractions	Type	<ul style="list-style-type: none"> - Historical - Natural

Table 6.1. Agents' recommendations matchings

Third scenario – random preferences in multiple requests

For a more thorough test a thousand requests were sent to the server. All data in the request was randomized (e.g., origin, destination of the flight, type of apartment rented, types of tourist attractions). The test source code is presented in *Annex 5: Application testing*.

In order to calculate the average degree of relevance of the results for the client, each property sent from the service test module was compared with the recommendations of the multi-agent system on the server. Similar to the comparison in the previous scenario. If, for example, the customer has chosen a certain airline, and the multi-agent system has found him a flight with exactly that airline, this match will have the value of 1. If there is no property match, the value will be 0.

All properties of all vacation services will be parsed, and at the end all values for match and unmatch will be summed and the arithmetic average will be done.

Figure 6.8. shows how closely the customer requirements matched the multi-agent system offerings for each service type. For example, for departure flights, in 1000 requests, agents matched 84% of flight properties (preferred flight's part of the day, company, class and others). The value *Mean of Means* represents the match rate of agents' recommendations for the user preferences for 1000 vacation packages (*each includes flight, property, and attractions*).

It must be mentioned in this context that the accuracy of the offers provided by the agents is based on the knowledge base they have. That is, if a client requests a vacation package with exact specifications that the agent does not have, the client will be provided with the result closest to his request.

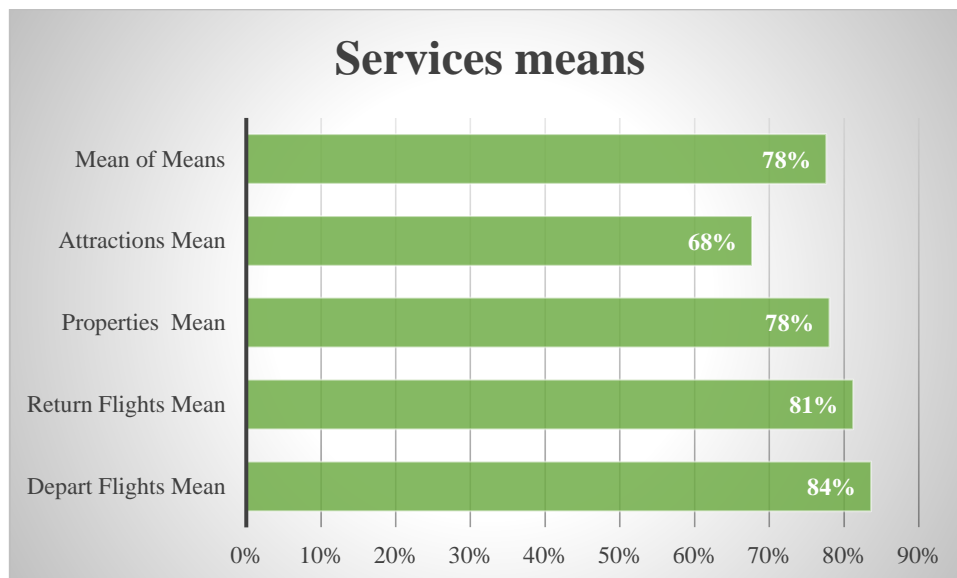


Figure 6.8. Means describing the match rate of agent recommendations against user preferences for each type of service per 1000 requests

Figure 6.9. shows how close was each multi-agent system offer to the user preferences in the 1000 requests. It can be seen that in two cases out of a thousand the multi-agent system provided an offer that matched below 35% of user preferences. Statistically, 990 results had a match greater than or equal to 50%.

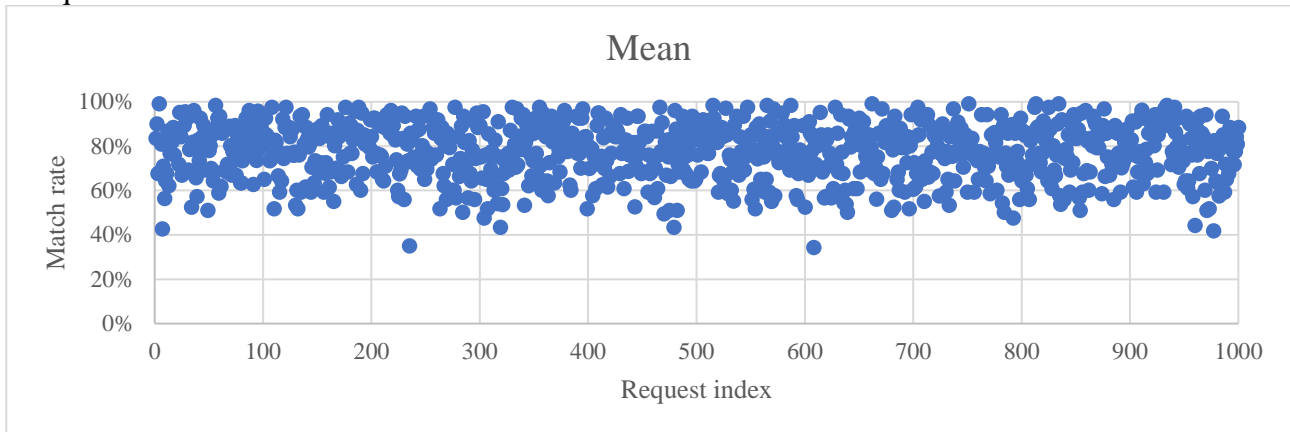


Figure 6.9. Mean values describing how accurate the results of vacation package were compared to the user's preferences in 1000 requests.

Chapter 7. Conclusions and future directions of development

This paper presents a web application based on the use of a multi-agent system to recommend a complete holiday package to the user. This system will be composed of 3 agents, also called "*tour agents*", each for one type of service: flight, property to stay and tourist attractions, and there is one more agent type "*coordinator*", who manages the completion of tasks and the announcement of the request completion. When the system receives a request, the coordinator will notify all agents that they have work to do. The travel agents will access the "gift basket" one by one and will choose one "gift" at a time, i.e., they will choose a service to complete. After searching for a suitable recommendation in its own database, the agent finding a solution will inform the coordinator, or if not found, will ask for help from other agents.

So, can be concluded that the multi-agent system aims not only at sharing tasks and executing them at the same time, but also at providing the opportunity for agents to talk to each other to solve a problem.

Over time, agents consolidate their level of expert in a particular service. Thus, each will have their own knowledge, and cooperation between agents will result in giving the client what they requested. Expert agents become experts exclusively because of the feedback that users provide. Because of this, when choosing a task, the agent will choose it based on its rating created exclusively by user evaluations.

Another feature of this application is that the user does not have to enter much data to receive a recommendation. Just with a minimum required, agents can find the necessary data, such as: departure and visit city, check-in date and holiday period plus the total number of adults.

When providing a response, agents do not search the user's request history to make an assumption of their preferences. Thus, as an improvement, a machine learning implementation could be added to study the user profile and make a much better offer based on his preference history.

Another way to increase the performance of the system is to add more powerful algorithms to search for potential recommendations. If it is assumed that a company's database stores millions of results, then processing all the data to find the best match is totally inefficient. This requires too many resources and time. One way would be Linear search, Binary search, Interpolation search, Hash table.

In principle, a search algorithm is not necessary if the multi-agent system would use an API to provide data about each holiday service. That is, there would be no need to store in its own database records about flights, or properties, or tourist attractions around the world. Agents could actually make an HTTP request to the API and receive the necessary responses. It would be a good practice, except that in this scenario it diminishes the usefulness of the multi-agent system. Because it should be the one that studies the user profile and based on the information it has to give a response.

One improvement that would certainly change the way the server communicates with the web application would be to move from *RESTful* to *Web Socket* architecture. The biggest advantage of this being the creation of a channel between client and server and maintaining continuous real-time communication.

Bibliography

- [1] Felix Richter, International travel levels tipped to soar again in 2022. Jun 30, 2022. Available online: <https://www.weforum.org/agenda/2022/06/international-travel-2022-covid19-tourism/#:~:text=The%20UNWTO's%20latest%20World%20Tourism,compared%20to%20the%20previous%20year>. (accessed on 10 Jun 2022)
- [2] David Camacho, José M. Molina, Daniel Borrajo, Ricardo Aler, “MAPWEB: cooperation between planning agents and Web agents”, 2001
- [3] F. Lorenzi, D. S. Santos, D. de Oliveira, and A. L. C. Bazzan. Task allocation in case-based recommender systems: a swarm intelligence approach. In H. Lin, editor, Architectural Design of Multi-Agent Systems. Information Science Reference, 2007
- [4] Fabiana Lorenzi, Fabio Arreguy Camargo Correa, Ana L. C. Bazzan, Mara Abel, Francesco Ricci, “A Multiagent Recommender System with Task-Based Agent Specialization”, 2008
- [5] Fabiana Lorenzi, Ana L. C. Bazzana, Mara Abel, Francesco Ricci, “Improving recommendations through an assumption-based multiagent approach: An application in the tourism domain”, 2011
- [6] W3schools® of Technology, RESTful Web Services. 2009 – 2023. Available online: <https://www.w3schools.in/restful-web-services/intro> (accessed on 5 July 2022)
- [7] Ahmet Özlü, Mastering REST Architecture — REST Architecture Details. Jul 29, 2018. Available online: <https://ahmetozlu.medium.com/mastering-rest-architecture-rest-architecture-details-e47ec659f6bc> (accessed on 5 July 2022)
- [8] JavaTpoint, Introduction to RESTful Web Services. 2011-2021. Available online: <https://www.javatpoint.com/restful-web-services-spring-boot#:~:text=Advantages%20of%20RESTful%20web%20services%20RESTful%20web%20services,data%20format%20like%20JSON%2C%20text%2C%20HTML%2C%20and%20XML>. (accessed on 9 July 2022)
- [9] American Express Company, Open Banking. 2022. Available online: <https://developer.americanexpress.com/open-banking> (accessed on 16 September 2022)
- [10] Gabry Martinez, How to create your own little Restful Web API without getting lost in the process — Part 2. Apr 19, 2018. Available online: <https://gabrymartinez.medium.com/how-to-create-your-own-little-restful-web-api-and-not-get-lost-in-the-process-part-2-473400256ce0> (accessed on 15 August 2022)
- [11] RoboCup, RoboCup Federation official website. 2022. Available online: <https://www.robocup.org/> (accessed on 21 August 2022)
- [12] Frans C.A. Groen, Matthijs T.J. Spaan, Jelle R. Kok, and Gregor Pavlin, “Real world multi-agent systems: information sharing, coordination and planning”, 2007
- [13] Mihaela Oprea, “Applications of multi-agent systems”, 2004
- [14] Florin Leon, “ActressMAS, a .NET Multi-Agent Framework Inspired by the Actor Model”, 2022
- [15] Luke, S.; Balan, G.C.; Panait, L.; Cioffi-Revilla, C.; Paus, S. MASON: A Java Multi-Agent Simulation Library. In Proceedings of the Agent 2003 Conference on Challenges in Social Simulation, 2003.
- [16] Bellifemine, F.L.; Caire, G.; Greenwood, D. Developing Multi-Agent Systems with JADE, 2007
- [17] Bernstein, P.; Bykov, S.; Geller, A.; Kliot, G.; Thelin, J. Orleans: Distributed Virtual Actors for Programmability and Scalability; 2014.
- [18] OpenTripMap – Map Service for sightseeing and travel planning. 2019. Available online: <https://opentripmap.com/en/#0/> (accessed on 12 September 2022)
- [19] OpenTripMap, OpenTripMap API. 2019. Available online: <https://opentripmap.io/product> (accessed on 12 September 2022)
- [20] simranjenny84, Advantages and Disadvantages of SQL. 07 July 2021. Available online: <https://www.geeksforgeeks.org/advantages-and-disadvantages-of-sql/> (accessed on 16 September 2022)
- [21] Priya Pedamkar, What is PostgreSQL? 2022. Available online: <https://www.educba.com/what-is-postgresql/> (accessed on 17 September 2022)
- [22] Robert C. Martin, “Design Principles and Design Patterns”, 2000. Available online: http://staff.cs.utu.fi/~jounsmmed/doos_06/material/DesignPrinciplesAndPatterns.pdf (accessed on 24 September 2022)
- [23] Nirav S., How to design 2 – tier architecture for Web application with High Availability. 30 May 2020. Available online: <https://www.eternalsoftsolutions.com/blog/how-to-design-aws-architecture-for-web-application-with-high-availability/> (accessed on 20 September 2022)
- [24] FlightAPI, Flight Pricing Data API Provider For Airlines – Flight API. 2019. Available online: <https://www.flightapi.io/> (accessed on 25 September 2022)
- [25] Booking.com, Booking.com Developers API. 1996–2023. Available online: <https://developers.booking.com/api/index.html> (accessed on 25 September 2022)
- [26] Cheapflights, Cheap Flights, Airline Tickets & Airfares – Find Deals on Flights at Cheapflights.com. 2022. Available online: <https://www.cheapflights.com/book-flights-online/> (accessed on 2 November 2022)

-
- [27] Inside Airbnb, Inside Airbnb: Get the Data. 2022. Available online: <http://insideairbnb.com/get-the-data/> (accessed on 2 November 2022)
- [28] ExtendOffice, Kutools – Combine More Than 300 Advanced Functions and Tools for Microsoft Excel. 2009 – 2023. Available online: <https://www.extendoffice.com/product/kutools-for-excel.html> (accessed on 2 November 2022)
- [29] Airportcodes.io, Airport Codes Europe (EU) European Airports | Airportcodes.io. Available online: <https://airportcodes.io/en/continent/europe/> (accessed on 3 November 2022)
- [30] Josh Close, A .NET library for reading and writing CSV files. Extremely fast, flexible, and easy to use | CsvHelper. 2009-2022. Available online: <https://joshclose.github.io/CsvHelper/> (accessed on 3 November 2022)
- [31] Microsoft, ML.NET. 2022. Available online: <https://dotnet.microsoft.com/en-us/apps/machinelearning-ai/ml-dotnet> (accessed on 3 November 2022)
- [32] Microsoft, MatrixFactorizationTrainer Class. 2022. Available online: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.matrixfactorizationtrainer?view=ml-dotnet-preview> (accessed on 6 November 2022)
- [33] Aigars Silkalns, Travelix. 21 May 2022. Available online: <https://colorlib.com/wp/template/travelix/> (accessed on 6 November 2022)
- [34] Namrata Podder 404 Page – svg animation. Available online: <https://codepen.io/namratapdr/pen/yLOgREo> (accessed on 6 November 2022)

Annexes

Annex 1: Database relations

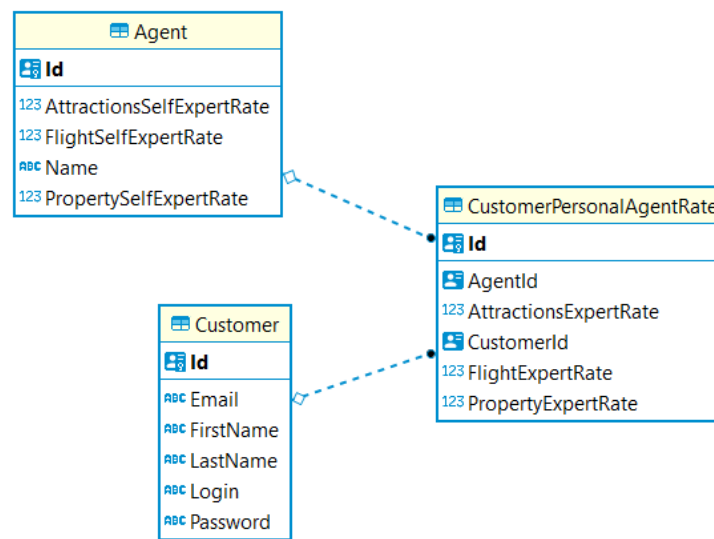


Figure A.1. Customer relations

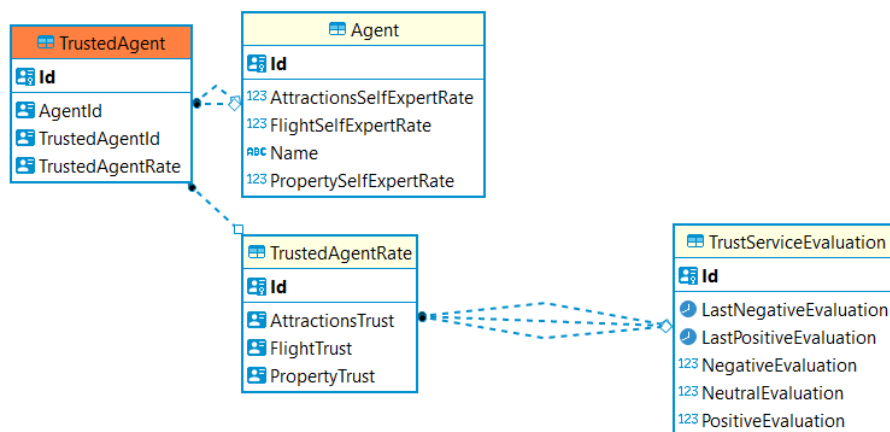


Figure A.2. Agent relations

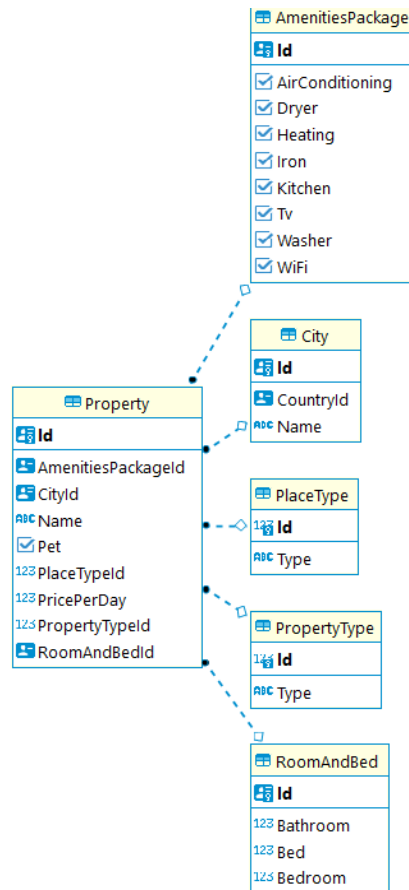
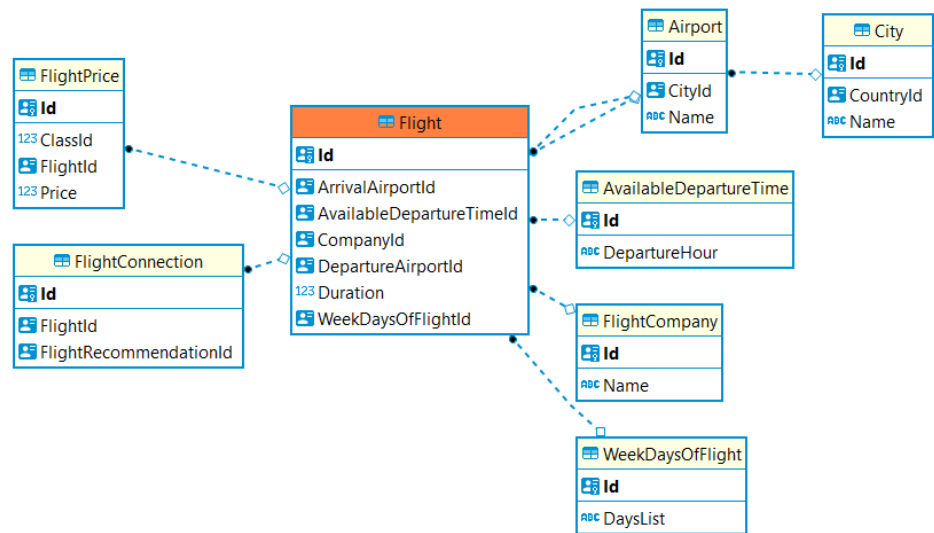


Figure A.3. Property relations

OpenTripMapAttraction
Xid
Country
CountryCode
Country
Geometry
Height
Html
Image
Kinds
Lat
LatMax
LatMin
Lon
LonMax
LonMin
Name
Neighbourhood
Osm
Otm
Pedestrian
Postcode
Rate
Source
State
Suburb
Text
Title
Town
Width
Wikidata
Wikipedia

Figure A.4. Attractions table



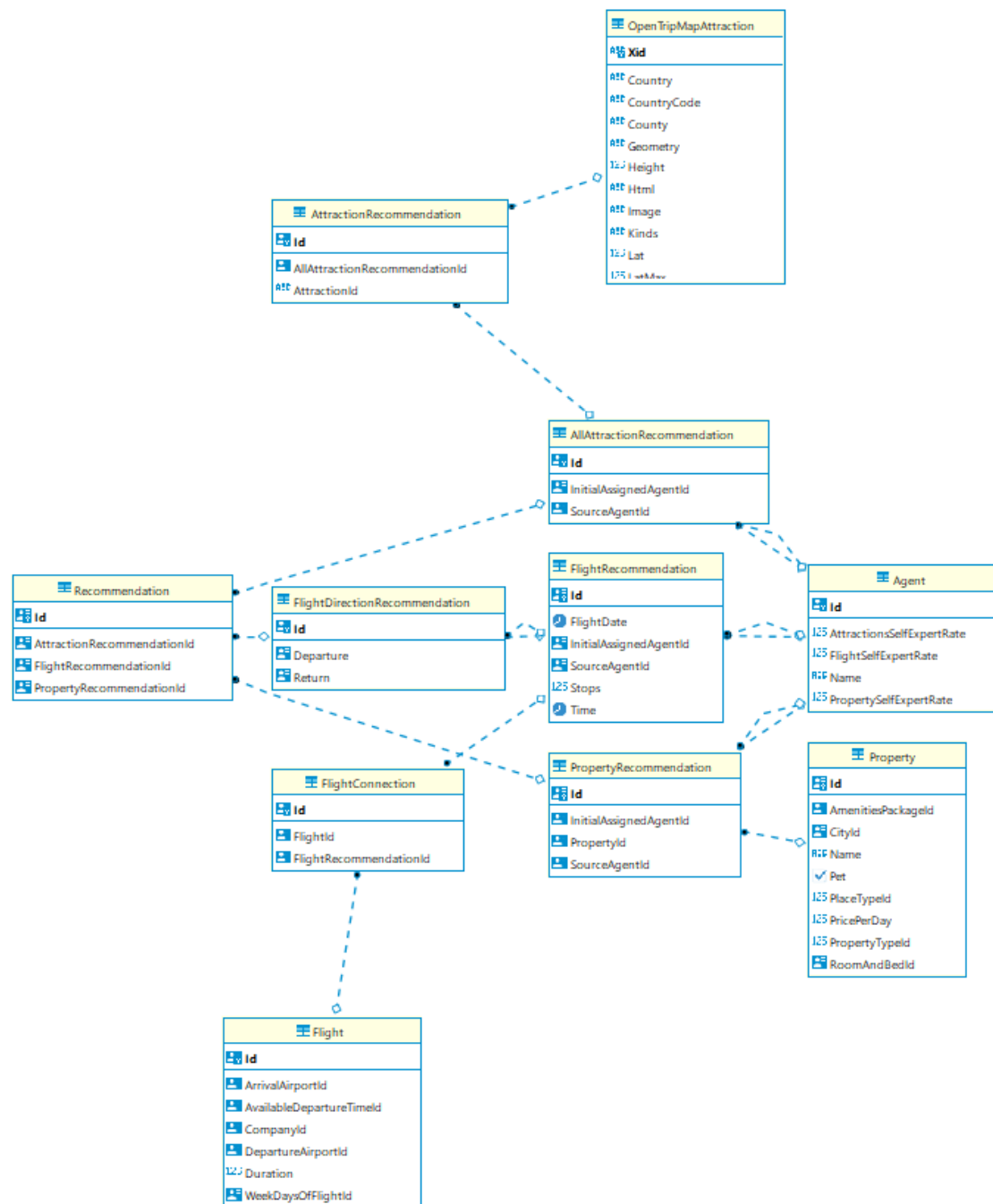


Figure A.7. Recommendation relations

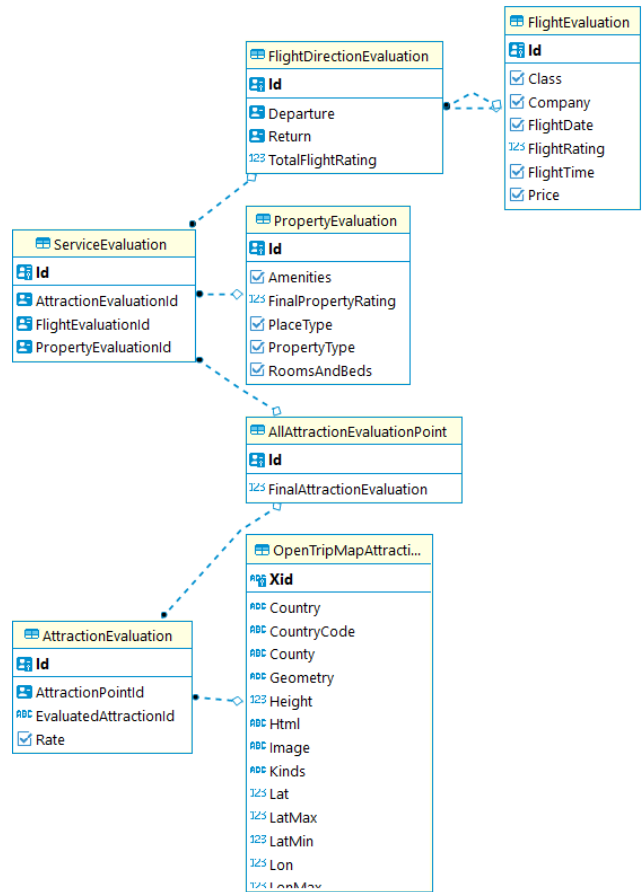


Figure A.8. Evaluation relations

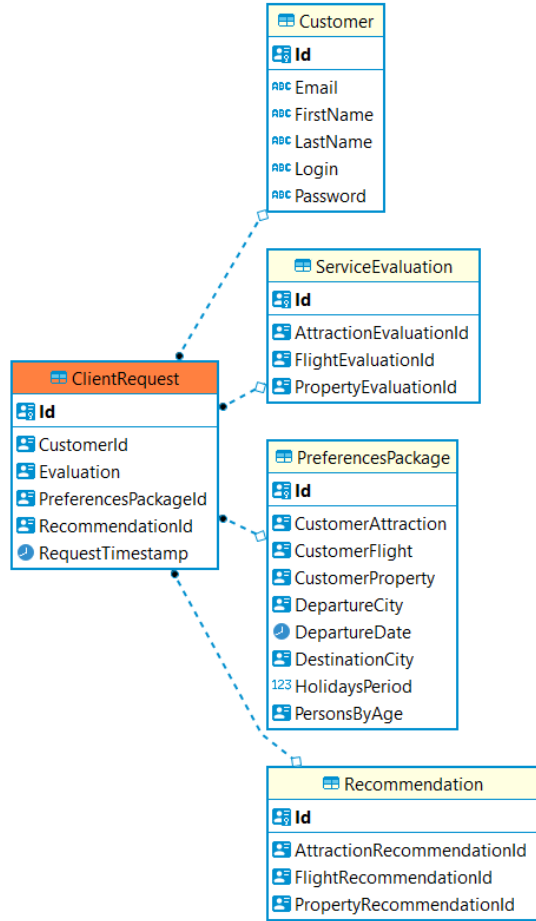


Figure A.9. Client's request relations

Annex 2: Web API

CustomerController.cs

```

using Microsoft.AspNetCore.Mvc;
using VacationPackageWebApi.Domain.AgentsEnvironment.Services;

namespace VacationPackageWebApi.API.Controllers;

[Route("[controller]/[action]")]
[ApiController]
public class FlightController : Controller
{
    private readonly IFlightService _flightService;

    public FlightController(IFlightService flightService)
    {
        _flightService = flightService;
    }

    [HttpGet]
    public List<string> GetFlightDepartureCities()
    {
        return _flightService.GetFlightDepartureCities();
    }

    [HttpGet("{flightDepartureCity}")]
    public List<string> GetFlightArrivalCities(string flightDepartureCity)
    {
        return _flightService.GetFlightArrivalCities(flightDepartureCity);
    }

    [HttpGet("{departureAndDestinationCity}")]
    public List<string> GetFlightCompaniesForDepartureDestinationCity(string departureAndDestinationCity)
    {
        return _flightService.GetFlightCompaniesForDepartureDestinationCity(departureAndDestinationCity);
    }
}

```

VacationPackageController.cs

```

using Microsoft.AspNetCore.Mvc;
using VacationPackageWebApi.Domain.AgentsEnvironment.Services;
using VacationPackageWebApi.Domain.CustomerServicesEvaluation;
using VacationPackageWebApi.Domain.Helpers;
using VacationPackageWebApi.Domain.PreferencesPackageRequest;
using VacationPackageWebApi.Infrastructure.Repositories.Models.RequestOfClient;

namespace VacationPackageWebApi.API.Controllers;

[Route("[controller]/[action]")]
[ApiController]
public class VacationPackageController : Controller
{
    private readonly IPreferencesPackageService _preferencesPackageService;
    private readonly IPreferencesPayloadInitializerServices _preferencesPayloadInitializerServices;

    public VacationPackageController(IPreferencesPackageService preferencesPackageService,
        IPreferencesPayloadInitializerServices preferencesPayloadInitializerServices)
    {
        _preferencesPackageService = preferencesPackageService;
        _preferencesPayloadInitializerServices = preferencesPayloadInitializerServices;
    }

    [HttpPost]
    public async Task<IActionResult> RequestVacationRecommendation([FromBody] PreferencesRequest preferencesPayload)
    {

```

```
var clientRequest = new ClientRequest
{
    Id = Guid.NewGuid(),
    RequestTimestamp = DateTime.Now
};

_preferencesPayloadInitializerServices.FulfillCustomizedExpertAgentsRates(ref preferencesPayload);

var recommendationPackage =
    await _preferencesPackageService.RequestVacationPackage(preferencesPayload, clientRequest.Id,
        clientRequest.RequestTimestamp);
UserReportHelper.WriteUserPreferencesRequest(preferencesPayload, clientRequest.RequestTimestamp);

if (recommendationPackage is
{
    AttractionsRecommendationResponse: { }, FlightRecommendationResponse: { },
    PropertyPreferencesResponse: { }
})
    Task.Factory.StartNew(
        _preferencesPackageService.SaveRecommendationResponse(recommendationPackage, clientRequest.Id));

recommendationPackage!.ClientRequestId = clientRequest.Id;

return new JsonResult(recommendationPackage);
}

[HttpPost]
public async Task<ActionResult> SaveEvaluations([FromBody] ServiceEvaluationDto serviceEvaluation)
{
    await _preferencesPackageService.SaveEvaluation(serviceEvaluation);
    UserReportHelper.ClearCurrentProcessingAgentSelfExpertLogData();
    return new JsonResult("Success");
}
}
```

CustomerController.cs

```
using Microsoft.AspNetCore.Mvc;
using VacationPackageWebApi.Domain.AgentsEnvironment.Services;
using VacationPackageWebApi.Domain.Customer;

namespace VacationPackageWebApi.API.Controllers;

[Route("[controller]/[action]")]
public class CustomerController : Controller
{
    private readonly ICustomerService _customerService;

    public CustomerController(ICustomerService customerService)
    {
        _customerService = customerService;
    }

    [HttpGet("{userData}")]
    public async Task<CustomerDto?> GetCustomerModel(string userData)
    {
        return await _customerService.GetCustomerModel(userData);
    }

    [HttpPost]
    public async Task<ActionResult> CreateNew([FromBody] CustomerDto customer)
    {
        await _customerService.CreateNewCustomer(customer);
        return new JsonResult("Success");
    }
}
```

Bootstrap.cs

```

using System.Text.Json;
using VacationPackageWebApi.Application.Services;
using VacationPackageWebApi.Domain.AgentsEnvironment.Contracts;
using VacationPackageWebApi.Domain.AgentsEnvironment.Services;
using VacationPackageWebApi.Domain.Attractions.Contracts;
using VacationPackageWebApi.Domain.Customer.Contracts;
using VacationPackageWebApi.Domain.Flight.Contracts;
using VacationPackageWebApi.Domain.PreferencesPackageRequest.Contracts;
using VacationPackageWebApi.Domain.Property.Contracts;
using VacationPackageWebApi.Domain.Services;
using VacationPackageWebApi.Infrastructure.Repositories;
using VacationPackageWebApi.Infrastructure.Repositories.Repositories;

namespace VacationPackageWebApi.API.Infrastructure;

public static class Bootstrap
{
    public static string AppSettingsPath { get; set; } = "appsettings.json";

    public static string EnvFilename { get; set; } = ".env";

    public static WebApplicationBuilder CreateAppBuilder(string[] args)
    {
        return WebApplication.CreateBuilder(args);
    }

    public static WebApplication BuildApp(WebApplicationBuilder builder)
    {
        var configuration = GetConfiguration();

        builder.WebHost
            .UseConfiguration(configuration)
            .UseUrls(configuration.GetValue<string>("Hostings:Urls"));

        ConfigureServices(builder.Services, configuration);

        var app = builder.Build();

        ConfigureApp(app);

        return app;
    }

    private static void ConfigureServices(IServiceCollection services, IConfiguration configuration)
    {
        // Add services to the container.
        services.AddScoped<IAgentService, AgentService>();
        services.AddScoped<IPreferencesPackageService, PreferencesPackageService>();
        services.AddScoped<IFlightService, FlightService>();
        services.AddScoped<IPropertyService, PropertyService>();
        services.AddScoped<IAttractionService, AttractionService>();
        services.AddScoped<IMasLoaderService, MasLoaderService>();
        services.AddScoped<IRecommendationService, RecommendationService>();
        services.AddScoped<IEvaluationService, EvaluationService>();
        services.AddScoped<IPreferencesPayloadInitializerServices, PreferencesPayloadInitializerServices>();
        services.AddScoped<ICustomerService, CustomerService>();

        services.AddScoped<IPreferencesPackageRequestRepository, PreferencesPackageRequestRepository>();
        services.AddScoped<IFlightRepository, FlightRepository>();
        services.AddScoped<IAgentRepository, AgentRepository>();
        services.AddScoped<IPropertyRepository, PropertyRepository>();
        services.AddScoped<IAttractionRepository, AttractionsRepository>();
        services.AddScoped<ICustomerRepository, CustomerRepository>();

        services.AddControllers();

        services.AddEndpointsApiExplorer();
    }
}

```

```
services.AddRouting(options => options.LowercaseUrls = true);

AddDb(services, configuration);

LoadMassEnvironment(services);
}

private static void LoadMassEnvironment(IServiceCollection services)
{
    var masLoaderService = services.BuildServiceProvider().CreateScope().ServiceProvider
        .GetRequiredService<IMasLoaderService>();
    masLoaderService.LoadMasEnvironmentAsync();
}

private static void AddDb(IServiceCollection services, IConfiguration configuration)
{
    var vacationPackageDatabaseOptions =
        configuration.GetOptions<VacationPackageDatabaseOptions>(VacationPackageDatabaseOptions.ConfigKey);
    services.AddEntityFramework(vacationPackageDatabaseOptions);
}

/// Configures the HTTP request pipeline.
private static void ConfigureApp(WebApplication app)
{
    app.HandleRouteNotFound();

    app.UseHttpsRedirection();

    app.UseAuthorization();

    app.MapControllers();
}

private static IConfiguration GetConfiguration()
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile(AppSettingsPath, false, true)
        .AddEnvironmentVariables();

    return builder.Build();
}

private static void HandleRouteNotFound(this WebApplication app)
{
    app.UseStatusCodePages(new StatusCodePagesOptions
    {
        HandleAsync = async ctx =>
        {
            if (ctx.HttpContext.Response.StatusCode == 404)
            {
                var result = JsonSerializer.Serialize(new
                {
                    message = "Route not found"
                });

                ctx.HttpContext.Response.ContentType = "application/json";

                await ctx.HttpContext.Response.WriteAsync(result);
            }
        }
    });
}
```

ConfigurationExtensions.cs

```
namespace VacationPackageWebApi.API.Infrastructure;
```

```

public static class ConfigurationExtensions
{
    public static T GetOptions<T>(this IConfiguration configuration, string configKey)
        where T : class, new()
    {
        var options = new T();
        configuration.GetSection(configKey).Bind(options);

        return options;
    }
}

```

Program.cs

```

using VacationPackageWebApi.API.Infrastructure;

var builder = Bootstrap.CreateAppBuilder(args);

var app = Bootstrap.BuildApp(builder);

app.Run();

```

AttractionService.cs

```

using VacationPackageWebApi.Domain.AgentsEnvironment.Services;
using VacationPackageWebApi.Domain.Attractions;
using VacationPackageWebApi.Domain.Attractions.Contracts;

namespace VacationPackageWebApi.Application.Services;

public class AttractionService : IAttractionService
{
    private readonly IAttractionRepository _attractionRepository;

    public AttractionService(IAttractionRepository attractionRepository)
    {
        _attractionRepository = attractionRepository;
    }

    public async Task<List<AttractionBusinessModel>> GetAllAttractionsAsync()
    {
        return await _attractionRepository.GetAllAttractionsAsync();
    }
}

```

CusomerService.cs

```

using VacationPackageWebApi.Domain.AgentsEnvironment.Services;
using VacationPackageWebApi.Domain.Customer;
using VacationPackageWebApi.Domain.Customer.Contracts;

namespace VacationPackageWebApi.Application.Services;

public class CustomerService : ICustomerService
{
    private readonly ICustomerRepository _customerRepository;

    public CustomerService(ICustomerRepository customerRepository)
    {
        _customerRepository = customerRepository;
    }

    public async Task<CustomerDto?> GetCustomerModel(string userData)
    {
        var customerData = userData.Split(" ");
        return await _customerRepository.GetCustomerModel(new CustomerLoginRequest
        {

```

```
        email = customerData[0],
        password = customerData[1]
    });
}

public async Task CreateNewCustomer(CustomerDto customer)
{
    await _customerRepository.CreateNewCustomer(customer);
}
}
```

FlightService.cs

```
using VacationPackageWebApi.Domain.AgentsEnvironment.Services;
using VacationPackageWebApi.Domain.Flight;
using VacationPackageWebApi.Domain.Flight.Contracts;

namespace VacationPackageWebApi.Application.Services;

public class FlightService : IFlightService
{
    private readonly IFlightRepository _flightRepository;

    public FlightService(IFlightRepository flightRepository)
    {
        _flightRepository = flightRepository;
    }

    public async Task<List<FlightBusinessModel>> GetAllFlightsAsync()
    {
        return await _flightRepository.GetAllFlights();
    }

    public List<string> GetFlightDepartureCities()
    {
        var flightsDepartureDestinationCities = _flightRepository.GetFlightsByUniqueDepartureArrivalAirport();

        return flightsDepartureDestinationCities.Select(flightDepartureDestination =>
            flightDepartureDestination.DepartureCityName + ", " + flightDepartureDestination.DepartureCountryName)
            .Distinct().ToList();
    }

    public List<string> GetFlightArrivalCities(string flightDepartureCity)
    {
        var flightArrivalCities = _flightRepository.GetFlightArrivalCities(flightDepartureCity);

        /* Destinations black list
        * Athens
        * Kazan
        * Nikolayev
        * Sabetta
        This is a temporary implementation. */
        foreach (var city in flightArrivalCities.ToList())
        {
            switch (city.DestinationCityName)
            {
                case "Athens":
                case "Kazan":
                case "Nikolayev":
                case "Sabetta":
                    flightArrivalCities.Remove(city);
                    break;
            }
        }

        return flightArrivalCities.Select(flightArrivalDestination => flightArrivalDestination.DestinationCityName +
            ", " + flightArrivalDestination
                .DestinationCountryName).Distinct().ToList();
    }

    public List<string> GetFlightCompaniesForDepartureDestinationCity(string departureAndDestinationCity)
```



```

{
    var departureDestination = departureAndDestinationCity.Split(" ");
    var departureCity = departureDestination[0];
    var destinationCity = departureDestination[1];

    return _flightRepository.GetFlightCompaniesForCities(departureCity, destinationCity);
}
}

```

PreferencesPackageService.cs

```

using VacationPackageWebApi.Domain.AgentsEnvironment.Services;
using VacationPackageWebApi.Domain.CustomerServicesEvaluation;
using VacationPackageWebApi.Domain.Helpers;
using VacationPackageWebApi.Domain.Mas.BusinessLogic;
using VacationPackageWebApi.Domain.PreferencesPackageRequest;
using VacationPackageWebApi.Domain.PreferencesPackageRequest.Contracts;
using VacationPackageWebApi.Domain.PreferencesPackageResponse;

namespace VacationPackageWebApi.Application.Services;

public class PreferencesPackageService : IPreferencesPackageService
{
    private readonly IAgentService _agentService;
    private readonly IEvaluationService _evaluationService;
    private readonly IPreferencesPackageRequestRepository _preferencesPackageRepository;
    private readonly IRecommendationService _recommendationService;

    public PreferencesPackageService(IAgentService agentService,
        IPreferencesPackageRequestRepository preferencesPackageRepository, IRecommendationService recommendationService,
        IEvaluationService evaluationService)
    {
        _evaluationService = evaluationService;
        _preferencesPackageRepository = preferencesPackageRepository;
        _recommendationService = recommendationService;
        _agentService = agentService;
    }

    public async Task<PreferencesResponse?> RequestVacationPackage(PreferencesRequest preferencesPayload,
        Guid clientRequestId, DateTime requestTimestamp)
    {
        var preferencesPackageId = await _preferencesPackageRepository.SavePreferences(preferencesPayload);
        await _preferencesPackageRepository.CreateClientRequest(preferencesPayload.CustomerId, preferencesPackageId,
            clientRequestId, requestTimestamp);
        await _agentService.InitializeCustomerPersonalAgentsRate(preferencesPayload.CustomerId);
        CommonRecommendationLogic.StoreAgentsTrustRate(_agentService.GetAllAgentsTrustInOtherAgent());
        return await _recommendationService.GetFullRecommendationsPackage(preferencesPayload);
    }

    public Action SaveRecommendationResponse(PreferencesResponse preferencesResponse, Guid clientRequestId)
    {
        _preferencesPackageRepository.WritePreferencesResponse(preferencesResponse);
        return _preferencesPackageRepository.SaveRecommendation(preferencesResponse, clientRequestId);
    }

    public async Task<Task> SaveEvaluation(ServiceEvaluationDto evaluationOfServices)
    {
        _evaluationService.CalculateEvaluationRatings(ref evaluationOfServices);
        _preferencesPackageRepository.SaveEvaluation(evaluationOfServices);
        UserReportHelper.WriteUserEvaluation(evaluationOfServices);
        await _preferencesPackageRepository.UpdateAgentsSelfExpertRate();
        await _preferencesPackageRepository.UpdateAgentTrustServiceEvaluation(evaluationOfServices);
        await _preferencesPackageRepository.UpdateCustomerPersonalAgentRate(evaluationOfServices);

        return Task.CompletedTask;
    }
}

```

PropertyService.cs

```
using VacationPackageWebApi.Domain.AgentsEnvironment.Services;
using VacationPackageWebApi.Domain.Property;
using VacationPackageWebApi.Domain.Property.Contracts;

namespace VacationPackageWebApi.Application.Services;

public class PropertyService : IPropertyService
{
    private readonly IPropertyRepository _propertyRepository;

    public PropertyService(IPropertyRepository propertyRepository)
    {
        _propertyRepository = propertyRepository;
    }

    public async Task<List<PropertyBusinessModel>> GetAllPropertiesAsync()
    {
        return await _propertyRepository.GetAllPropertiesAsync();
    }
}
```

AgentService.cs

```
using VacationPackageWebApi.Domain.AgentsEnvironment.AgentModels;
using VacationPackageWebApi.Domain.AgentsEnvironment.Contracts;
using VacationPackageWebApi.Domain.AgentsEnvironment.Services;

namespace VacationPackageWebApi.Application.Services;

public class AgentService : IAgentService
{
    private readonly IAgentRepository _agentRepository;

    public AgentService(IAgentRepository agentRepository)
    {
        _agentRepository = agentRepository;
    }

    public async Task<List<TourismAgent>> GetAllAgentsAsync()
    {
        return await _agentRepository.GetAllAgentsAsync();
    }

    public Task InitializeCustomerPersonalAgentsRate(Guid customerId)
    {
        return _agentRepository.InitializeCustomerPersonalAgentsRate(customerId);
    }

    public List<CustomerPersonalAgentRateBusinessModel> GetCustomerPersonalAgentsServicesRates(Guid customerId)
    {
        return _agentRepository.GetCustomerPersonalAgentsServicesRates(customerId);
    }

    public Dictionary<Guid, List<TrustAgentRateBusinessModel>> GetAllAgentsTrustInOtherAgent()
    {
        return _agentRepository.GetAllAgentsTrustInOtherAgent();
    }

    public List<TrustAgentRateBusinessModel> GetTrustInOtherAgentsOfAgentWithId(Guid agentId)
    {
        return _agentRepository.GetTrustInOtherAgentsOfAgentWithId(agentId);
    }
}
```

IAgentRepository.cs

```
using VacationPackageWebApi.Domain.AgentsEnvironment.AgentModels;
```

```
namespace VacationPackageWebApi.Domain.AgentsEnvironment.Contracts;
```

```
public interface IAgentRepository
{
    public Task<List<TourismAgent>> GetAllAgentsAsync();
    public List<CustomerPersonalAgentRateBusinessModel> GetCustomerPersonalAgentsServicesRates(Guid customerId);
    public Task InitializeCustomerPersonalAgentsRate(Guid customerId);
    public List<TrustAgentRateBusinessModel> GetTrustInOtherAgentsOfAgentWithId(Guid agentId);
    public Dictionary<Guid, List<TrustAgentRateBusinessModel>> GetAllAgentsTrustInOtherAgent();
}
```

```
IAgentService.cs
```

```
using VacationPackageWebApi.Domain.AgentsEnvironment.AgentModels;
```

```
namespace VacationPackageWebApi.Domain.AgentsEnvironment.Services;
```

```
public interface IAgentService
{
    public Task<List<TourismAgent>> GetAllAgentsAsync();

    public Task InitializeCustomerPersonalAgentsRate(Guid customerId);

    public List<CustomerPersonalAgentRateBusinessModel> GetCustomerPersonalAgentsServicesRates(
        Guid customerId);

    public List<TrustAgentRateBusinessModel> GetTrustInOtherAgentsOfAgentWithId(Guid agentId);
    public Dictionary<Guid, List<TrustAgentRateBusinessModel>> GetAllAgentsTrustInOtherAgent();
}
```

```
IAttractionService.cs
```

```
using VacationPackageWebApi.Domain.Attractions;
```

```
namespace VacationPackageWebApi.Domain.AgentsEnvironment.Services;
```

```
public interface IAttractionService
{
    public Task<List<AttractionBusinessModel>> GetAllAttractionsAsync();
}
```

```
ICustomerService.cs
```

```
using VacationPackageWebApi.Domain.Customer;
```

```
namespace VacationPackageWebApi.Domain.AgentsEnvironment.Services;
```

```
public interface ICustomerService
{
    public Task<CustomerDto?> GetCustomerModel(string userData);
    public Task CreateNewCustomer(CustomerDto customer);
}
```

```
IEvaluationService.cs
```

```
using VacationPackageWebApi.Domain.CustomerServicesEvaluation;
```

```
namespace VacationPackageWebApi.Domain.AgentsEnvironment.Services;
```

```
public interface IEvaluationService
{
    public void CalculateEvaluationRatings(ref ServiceEvaluationDto evaluationOfServices);
}
```

```
IFlightService.cs
```

```
using VacationPackageWebApi.Domain.Flight;
```

```
namespace VacationPackageWebApi.Domain.AgentsEnvironment.Services;
```

```
public interface IFlightService
{
    public Task<List<FlightBusinessModel>> GetAllFlightsAsync();
    public List<string> GetFlightDepartureCities();
    List<string> GetFlightArrivalCities(string flightDepartureCity);
    List<string> GetFlightCompaniesForDepartureDestinationCity(string departureAndDestinationCity);
}
```

IMasLoaderService.cs

```
namespace VacationPackageWebApi.Domain.AgentsEnvironment.Services;
```

```
public interface IMasLoaderService
{
    public Task LoadMasEnvironmentAsync();
}
```

IPreferencesPackageService.cs

```
using VacationPackageWebApi.Domain.CustomerServicesEvaluation;
using VacationPackageWebApi.Domain.PreferencesPackageRequest;
using VacationPackageWebApi.Domain.PreferencesPackageResponse;

namespace VacationPackageWebApi.Domain.AgentsEnvironment.Services;

public interface IPreferencesPackageService
{
    public Task<PreferencesResponse?> RequestVacationPackage(PreferencesRequest preferencesPayload,
        Guid clientRequestId, DateTime requestTimestamp);

    public Action SaveRecommendationResponse(PreferencesResponse preferencesResponse, Guid clientRequestId);
    public Task<Task> SaveEvaluation(ServiceEvaluationDto evaluationOfServices);
}
```

IPreferencesPayloadInitializerServices.cs

```
using VacationPackageWebApi.Domain.PreferencesPackageRequest;

namespace VacationPackageWebApi.Domain.AgentsEnvironment.Services;

public interface IPreferencesPayloadInitializerServices
{
    public void FulfillCustomizedExpertAgentsRates(ref PreferencesRequest preferencesPayload);
}
```

IPropertyService.cs

```
using VacationPackageWebApi.Domain.Property;

namespace VacationPackageWebApi.Domain.AgentsEnvironment.Services;

public interface IPropertyService
{
    public Task<List<PropertyBusinessModel>> GetAllPropertiesAsync();
}
```

IRecommendationService.cs

```
using VacationPackageWebApi.Domain.PreferencesPackageRequest;
using VacationPackageWebApi.Domain.PreferencesPackageResponse;

namespace VacationPackageWebApi.Domain.AgentsEnvironment.Services;
```

```

public interface IRecommendationService
{
    public Task<PreferencesResponse?> GetFullRecommendationsPackage(PreferencesRequest preferencesPayload);
}

```

IAttractionRepository.cs

```
namespace VacationPackageWebApi.Domain.Attractions.Contracts;
```

```

public interface IAttractionRepository
{
    public Task<List<AttractionBusinessModel>> GetAllAttractionsAsync();
}

```

ICustomerRepository.cs

```
namespace VacationPackageWebApi.Domain.Customer.Contracts;
```

```

public interface ICustomerRepository
{
    public Task<CustomerDto?> GetCustomerModel(CustomerLoginRequest userData);
    public Task CreateNewCustomer(CustomerDto customer);
}

```

ClassTypeEnum.cs

```
namespace VacationPackageWebApi.Domain.Enums;
```

```

public enum ClassTypeId
{
    Economy = 1,
    Business,
    First,
    Default
}

```

CoordinatorTasksToTrackEnum.cs

```
namespace VacationPackageWebApi.Domain.Enums;
```

```

public enum CoordinatorTasksDone
{
    DepartureFlight,
    ReturnFlight,
    Property,
    Attraction
}

```

IFlightRepository.cs

```
using VacationPackageWebApi.Domain.Flight.UIModels;
```

```
namespace VacationPackageWebApi.Domain.Flight.Contracts;
```

```

public interface IFlightRepository
{
    public Task<List<FlightBusinessModel>> GetAllFlights();
    public List<FlightDepartureCities> GetFlightsByUniqueDepartureArrivalAirport();
    public List<FlightDestinationCities> GetFlightArrivalCities(string flightDepartureCity);
    List<string> GetFlightCompaniesForCities(string departureCity, string destinationCity);
}

```

DateTimeUtils.cs

```
namespace VacationPackageWebApi.Domain.Helpers;

public static class DateTimeUtils
{
    public static string GetDayOfWeekFromDate(this DateTime date)
    {
        return date.DayOfWeek.ToString();
    }
}
```

StringUtils.cs

```
namespace VacationPackageWebApi.Domain.Helpers;

public static class StringUtils
{
    public static List<TimeOnly> ConvertStringTimeListToTimeOnly(this string timeData)
    {
        var myStr = timeData.Split(' ');
        return myStr.Select(TimeOnly.Parse).ToList();
    }

    private static DayOfWeek? ConvertStringToDayOfWeek(this string dayOfWeek)
    {
        return dayOfWeek switch
        {
            "Monday" => DayOfWeek.Monday,
            "Tuesday" => DayOfWeek.Tuesday,
            "Wednesday" => DayOfWeek.Wednesday,
            "Thursday" => DayOfWeek.Thursday,
            "Friday" => DayOfWeek.Friday,
            "Saturday" => DayOfWeek.Saturday,
            "Sunday" => DayOfWeek.Sunday,
            _ => null
        };
    }

    public static List<DayOfWeek> ConvertStringDaysOfWeekListToEnumList(this string daysOfFlightList)
    {
        var myStr = daysOfFlightList.Split(' ');

        return myStr.Select(str => (DayOfWeek) ConvertStringToDayOfWeek(str)!).ToList();
    }
}
```

UserReportHelper.cs

```
using VacationPackageWebApi.Domain.CustomerServicesEvaluation;
using VacationPackageWebApi.Domain.Enums;
using VacationPackageWebApi.Domain.Helpers.Models;
using VacationPackageWebApi.Domain.PreferencesPackageRequest;
using VacationPackageWebApi.Domain.PreferencesPackageResponse;

namespace VacationPackageWebApi.Domain.Helpers;

public static class UserReportHelper
{
    private const string _pathToLogFile = @"C:\Users\emihailov\Desktop\VacationPackageWebApi\Log\";
    private static string _fileName;

    private static string _currentProcessingAgentSelfExpertRate = string.Empty;
    private static string _currentProcessingAgentSelfExpertService = string.Empty;
    private static bool _personalAgentServiceScoreHeaderInitialized;

    public static void WriteUserPreferencesRequest(PreferencesRequest preferencesPayload, DateTime requestTimestamp)
    {
        _fileName = $"log{requestTimestamp.Date:yy-MM-dd}-{requestTimestamp.ToString("HH-mm-ss")}.txt";
        using var fileStreamWriter = File.AppendText(_pathToLogFile + _fileName);
    }
}
```

```

fileStreamWriter.WriteLine("_____");
fileStreamWriter.WriteLine("Preferences of the user");
fileStreamWriter.WriteLine($"Customer Id = {preferencesPayload.CustomerId}");
fileStreamWriter.WriteLine($"Departure Date = {preferencesPayload.DepartureDate:dd-MM-yyyy}");
fileStreamWriter.WriteLine($"Holidays Period = {preferencesPayload.HolidaysPeriod}");
fileStreamWriter.WriteLine($"Request Timestamp = {requestTimestamp}");
fileStreamWriter.WriteLine($"Departure City = {preferencesPayload.DepartureCityNavigation.Name}");
fileStreamWriter.WriteLine($"Destination City = {preferencesPayload.DestinationCityNavigation.Name}");

fileStreamWriter.WriteLine("Persons by age:");
fileStreamWriter.WriteLine($"Adults: {preferencesPayload.PersonsByAgeNavigation.Adult}");
fileStreamWriter.WriteLine($"Children: {preferencesPayload.PersonsByAgeNavigation.Children}");
fileStreamWriter.WriteLine($"Infant: {preferencesPayload.PersonsByAgeNavigation.Infant}");

fileStreamWriter.WriteLine("\nFlight Preferences");
if (preferencesPayload.CustomerFlightNavigation != null)
{
    fileStreamWriter.WriteLine("\tDeparture Flight Preferences");
    if (preferencesPayload.CustomerFlightNavigation!.DepartureNavigation != null)
    {
        fileStreamWriter.WriteLine("\tFlight Companies");
        if (preferencesPayload.CustomerFlightNavigation!.DepartureNavigation!.FlightCompaniesNavigationList != null)
        {
            foreach (var departureFlightCompanyPreference in preferencesPayload.CustomerFlightNavigation
                .DepartureNavigation.FlightCompaniesNavigationList)
            {
                fileStreamWriter.WriteLine($"Company: {departureFlightCompanyPreference.Company.Name}");
            }
        }
        fileStreamWriter.WriteLine("\tDeparture Day Periods");
        if (preferencesPayload.CustomerFlightNavigation!.DepartureNavigation!.DeparturePeriodPreference != null)
        {
            if (preferencesPayload.CustomerFlightNavigation!.DepartureNavigation!.DeparturePeriodPreference
                .EarlyMorning)
            {
                fileStreamWriter.WriteLine("\t\tEarly Morning");
            }

            if (preferencesPayload.CustomerFlightNavigation!.DepartureNavigation!.DeparturePeriodPreference
                .Morning)
            {
                fileStreamWriter.WriteLine("\t\tMorning");
            }

            if (preferencesPayload.CustomerFlightNavigation!.DepartureNavigation!.DeparturePeriodPreference
                .Afternoon)
            {
                fileStreamWriter.WriteLine("\t\tAfternoon");
            }

            if (preferencesPayload.CustomerFlightNavigation!.DepartureNavigation!.DeparturePeriodPreference
                .Night)
            {
                fileStreamWriter.WriteLine("\t\tNight");
            }
        }

        fileStreamWriter.WriteLine(
            $"Class: {(ClassTypeId)preferencesPayload.CustomerFlightNavigation!.DepartureNavigation!.Class!.Class!.ToString()}");

        fileStreamWriter.WriteLine("\tStops");
        if (preferencesPayload.CustomerFlightNavigation!.DepartureNavigation!.StopsNavigation != null)
        {
            switch (preferencesPayload.CustomerFlightNavigation!.DepartureNavigation!.StopsNavigation.Type)
            {
                case (short) StopsTypePreferenceId.Direct:
                    fileStreamWriter.WriteLine("\t\tDirect");
                    break;
                case (short) StopsTypePreferenceId.OneStop:
                    fileStreamWriter.WriteLine("\t\tOne Stop");
                    break;
                case (short) StopsTypePreferenceId.TwoOrMoreStops:
                    fileStreamWriter.WriteLine("\t\tTwo or More Stops");
                    break;
            }
        }

        fileStreamWriter.WriteLine("\nReturn Flight Preferences");
        if (preferencesPayload.CustomerFlightNavigation!.ReturnNavigation != null)
    }
}

```



```
{
    fileStreamWriter.WriteLine("\t\tFlight Companies:");
    if (preferencesPayload.CustomerFlightNavigation!.ReturnNavigation!.FlightCompaniesNavigationList !=
        null)
        foreach (var returnFlightCompanyPreference in preferencesPayload.CustomerFlightNavigation
            .ReturnNavigation
            .FlightCompaniesNavigationList)
            fileStreamWriter.WriteLine($"{\t\t{returnFlightCompanyPreference.Company.Name}}");
    fileStreamWriter.WriteLine("\tDeparture Day Periods:");
    if (preferencesPayload.CustomerFlightNavigation!.ReturnNavigation!.DeparturePeriodPreference != null)
    {
        if (preferencesPayload.CustomerFlightNavigation!.ReturnNavigation!.DeparturePeriodPreference
            .EarlyMorning)
            fileStreamWriter.WriteLine("\t\tEarly Morning");

        if (preferencesPayload.CustomerFlightNavigation!.ReturnNavigation!.DeparturePeriodPreference
            .Morning)
            fileStreamWriter.WriteLine("\t\tMorning");

        if (preferencesPayload.CustomerFlightNavigation!.ReturnNavigation!.DeparturePeriodPreference
            .Afternoon)
            fileStreamWriter.WriteLine("\t\tAfternoon");

        if (preferencesPayload.CustomerFlightNavigation!.ReturnNavigation!.DeparturePeriodPreference.Night)
            fileStreamWriter.WriteLine("\t\tNight");
    }

    fileStreamWriter.WriteLine(
        $"{\tClass: {(ClassTypeId)
preferencesPayload.CustomerFlightNavigation!.ReturnNavigation!.Class!.Class}.ToString()}}");

    fileStreamWriter.WriteLine("\tStops:");
    if (preferencesPayload.CustomerFlightNavigation!.ReturnNavigation!.StopsNavigation != null)
        switch (preferencesPayload.CustomerFlightNavigation!.ReturnNavigation!.StopsNavigation.Type)
        {
            case (short) StopsTypePreferenceId.Direct:
                fileStreamWriter.WriteLine("\t\tDirect");
                break;
            case (short) StopsTypePreferenceId.OneStop:
                fileStreamWriter.WriteLine("\t\tOne Stop");
                break;
            case (short) StopsTypePreferenceId.TwoOrMoreStops:
                fileStreamWriter.WriteLine("\t\tTwo or More Stops");
                break;
        }
    }
}

fileStreamWriter.WriteLine("\nProperty Preferences");
if (preferencesPayload.CustomerPropertyNavigation is {Pets: true})
    fileStreamWriter.WriteLine("\tPets: yes");

fileStreamWriter.WriteLine("\tProperty Type:");
if (preferencesPayload.CustomerPropertyNavigation!.PropertyTypeNavigation != null)
{
    if (preferencesPayload.CustomerPropertyNavigation!.PropertyTypeNavigation.House)
        fileStreamWriter.WriteLine("\t\tHouse");

    if (preferencesPayload.CustomerPropertyNavigation!.PropertyTypeNavigation.Apartment)
        fileStreamWriter.WriteLine("\t\tApartment");

    if (preferencesPayload.CustomerPropertyNavigation!.PropertyTypeNavigation.GuestHouse)
        fileStreamWriter.WriteLine("\t\tGuestHouse");

    if (preferencesPayload.CustomerPropertyNavigation!.PropertyTypeNavigation.Hotel)
        fileStreamWriter.WriteLine("\t\tHotel");
}

fileStreamWriter.WriteLine("\tPlace Type:");
if (preferencesPayload.CustomerPropertyNavigation!.PlaceTypeNavigation != null)
```

```

{
    if (preferencesPayload.CustomerPropertyNavigation!.PlaceTypeNavigation.EntirePlace)
        fileStreamWriter.WriteLine("\t\t EntirePlace");

    if (preferencesPayload.CustomerPropertyNavigation!.PlaceTypeNavigation.PrivateRoom)
        fileStreamWriter.WriteLine("\t\t PrivateRoom");

    if (preferencesPayload.CustomerPropertyNavigation!.PlaceTypeNavigation.SharedRoom)
        fileStreamWriter.WriteLine("\t\t SharedRoom");
}

fileStreamWriter.WriteLine("\tRoom and Beds:");
if (preferencesPayload.CustomerPropertyNavigation!.RoomsAndBedsNavigation != null)
{
    fileStreamWriter.WriteLine(
        $" \t\tBedrooms: {preferencesPayload.CustomerPropertyNavigation!.RoomsAndBedsNavigation.Bedrooms}");
    fileStreamWriter.WriteLine(
        $" \t\tBeds: {preferencesPayload.CustomerPropertyNavigation!.RoomsAndBedsNavigation.Beds}");
    fileStreamWriter.WriteLine(
        $" \t\tBathrooms: {preferencesPayload.CustomerPropertyNavigation!.RoomsAndBedsNavigation.Bathrooms}");
}

fileStreamWriter.WriteLine("\tAmenities:");
if (preferencesPayload.CustomerPropertyNavigation!.AmenitiesNavigation != null)
{
    if (preferencesPayload.CustomerPropertyNavigation!.AmenitiesNavigation.WiFi)
        fileStreamWriter.WriteLine("\t\t WiFi");

    if (preferencesPayload.CustomerPropertyNavigation!.AmenitiesNavigation.Kitchen)
        fileStreamWriter.WriteLine("\t\t Kitchen");

    if (preferencesPayload.CustomerPropertyNavigation!.AmenitiesNavigation.Washer)
        fileStreamWriter.WriteLine("\t\t Washer");

    if (preferencesPayload.CustomerPropertyNavigation!.AmenitiesNavigation.Dryer)
        fileStreamWriter.WriteLine("\t\t Dryer");

    if (preferencesPayload.CustomerPropertyNavigation!.AmenitiesNavigation.AirConditioning)
        fileStreamWriter.WriteLine("\t\t Air Conditioning");

    if (preferencesPayload.CustomerPropertyNavigation!.AmenitiesNavigation.Heating)
        fileStreamWriter.WriteLine("\t\t Heating");

    if (preferencesPayload.CustomerPropertyNavigation!.AmenitiesNavigation.Tv)
        fileStreamWriter.WriteLine("\t\t TV");

    if (preferencesPayload.CustomerPropertyNavigation!.AmenitiesNavigation.Iron)
        fileStreamWriter.WriteLine("\t\t Iron");
}

fileStreamWriter.WriteLine("\n \tAttractions Preferences:");
if (preferencesPayload.CustomerAttractionNavigation != null)
{
    if (preferencesPayload.CustomerAttractionNavigation.Architecture)
        fileStreamWriter.WriteLine("\t\t Architecture");

    if (preferencesPayload.CustomerAttractionNavigation.Cultural)
        fileStreamWriter.WriteLine("\t\t Cultural");

    if (preferencesPayload.CustomerAttractionNavigation.Historical)
        fileStreamWriter.WriteLine("\t\t Historical");

    if (preferencesPayload.CustomerAttractionNavigation.Natural)
        fileStreamWriter.WriteLine("\t\t Natural");

    if (preferencesPayload.CustomerAttractionNavigation.Other)
        fileStreamWriter.WriteLine("\t\t Other");

    if (preferencesPayload.CustomerAttractionNavigation.Religion)

```

```
        fileStreamWriter.WriteLine("\t\t Religion");

        if (preferencesPayload.CustomerAttractionNavigation.IndustrialFacilities)
            fileStreamWriter.WriteLine("\t\t Industrial Facilities");
    }

    fileStreamWriter.WriteLine("_____");
}

public static void WritePreferencesResponse(PreferencesResponse preferencesResponse,
    string departureFlightSourceAgentName, string returnFlightSourceAgentName, string propertySourceAgentName,
    string attractionsSourceAgentName)
{
    using var fileStreamWriter = File.AppendText(_pathToLogFile + _fileName);

    fileStreamWriter.WriteLine("_____");
    fileStreamWriter.WriteLine("Flight recommendations ");
    fileStreamWriter.WriteLine("\n\tDeparture flight: ");
    fileStreamWriter.WriteLine(
        $"{preferencesResponse.FlightRecommendationResponse!.FlightDirectionRecommendation!.DepartureFlightRecommendation!.Flight
        Date:dd-MM-yyyy}");
    fileStreamWriter.WriteLine(
        $"{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.DepartureFlightRecommendation.Departur
        eTime:HH:mm}");

    fileStreamWriter.WriteLine("\tClass:");
    switch (preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation
        .DepartureFlightRecommendation.FlightClass)
    {
        case (short) ClassTypeId.Business:
            fileStreamWriter.WriteLine("\t\tBusiness");
            break;
        case (short) ClassTypeId.Economy:
            fileStreamWriter.WriteLine("\t\tEconomy");
            break;
        case (short) ClassTypeId.First:
            fileStreamWriter.WriteLine("\t\tFirst");
            break;
    }

    fileStreamWriter.WriteLine("\tMore flight information");
    fileStreamWriter.WriteLine(
        $"{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.DepartureFlightRecommendation.FlightCo
        nnection.First().Flight.DepartureAirport.City.Country.Name}");
    fileStreamWriter.WriteLine(
        $"{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.DepartureFlightRecommendation.FlightCo
        nnection.First().Flight.DepartureAirport.City.Name}");
    fileStreamWriter.WriteLine(
        $"{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.DepartureFlightRecommendation.FlightCo
        nnection.First().Flight.DepartureAirport.Name}");
    fileStreamWriter.WriteLine(
        $"{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.DepartureFlightRecommendation.FlightCo
        nnection.First().Flight.Company.Name}");
    fileStreamWriter.WriteLine(
        $"{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.DepartureFlightRecommendation.FlightCo
        nnection.First().Flight.WeekDaysOfFlight.DaysList}");
    fileStreamWriter.WriteLine(
        $"{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.DepartureFlightRecommendation.FlightCo
        nnection.First().Flight.AvailableDepartureTime.DepartureHour}");

    fileStreamWriter.WriteLine(
        $"{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.DepartureFlightRecommendation.FlightCo
        nnection.First().Flight.AvailableDepartureTime.DepartureHour}");
}
```

```

{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.DepartureFlightRecommendation.InitialAssignedAgentName}");
    fileStreamWriter.WriteLine($"\\tSource agent name: {departureFlightSourceAgentName}");

    fileStreamWriter.WriteLine("\\n \\tReturn flight: ");
    fileStreamWriter.WriteLine(
        $"\\tFlight Date:
{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.ReturnFlightRecommendation!.FlightDate:
dd-MM-yyyy}");
    fileStreamWriter.WriteLine(
        $"\\tDeparture Time:
{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.ReturnFlightRecommendation.DepartureTime:HH:mm}");

    fileStreamWriter.WriteLine("\\tClass ");
    switch (preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation
        .ReturnFlightRecommendation.FlightClass)
    {
        case (short) ClassTypeId.Business:
            fileStreamWriter.WriteLine("\\t\\tBusiness");
            break;
        case (short) ClassTypeId.Economy:
            fileStreamWriter.WriteLine("\\t\\tEconomy");
            break;
        case (short) ClassTypeId.First:
            fileStreamWriter.WriteLine("\\t\\tFirst");
            break;
    }

    fileStreamWriter.WriteLine("\\n \\tMore flight information");
    fileStreamWriter.WriteLine(
        $"\\t \\tDeparture Country:
{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.ReturnFlightRecommendation.FlightConnection.First().Flight.DepartureAirport.City.Country.Name}");
    fileStreamWriter.WriteLine(
        $"\\t \\tDeparture City:
{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.ReturnFlightRecommendation.FlightConnection.First().Flight.DepartureAirport.City.Name}");
    fileStreamWriter.WriteLine(
        $"\\t \\tDeparture Airport:
{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.ReturnFlightRecommendation.FlightConnection.First().Flight.DepartureAirport.Name}");
    fileStreamWriter.WriteLine(
        $"\\t \\tFlight Company:
{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.ReturnFlightRecommendation.FlightConnection.First().Flight.Company.Name}");
    fileStreamWriter.WriteLine(
        $"\\t \\tFlight is each:
{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.ReturnFlightRecommendation.FlightConnection.First().Flight.WeekDaysOfFlight.DaysList}");
    fileStreamWriter.WriteLine(
        $"\\t \\tAvailable departure time:
{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.ReturnFlightRecommendation.FlightConnection.First().Flight.AvailableDepartureTime.DepartureHour}");

    fileStreamWriter.WriteLine(
        $"\\n \\tInitial assigned agent name:
{preferencesResponse.FlightRecommendationResponse.FlightDirectionRecommendation.ReturnFlightRecommendation.InitialAssignedAgentName}");
    fileStreamWriter.WriteLine($"\\tSource agent name: {returnFlightSourceAgentName}");

    fileStreamWriter.WriteLine("\\nProperty recommendations");
    fileStreamWriter.WriteLine(
        $"\\t City {preferencesResponse.PropertyPreferencesResponse!.PropertyRecommendationBModel.Property.City.Name}");
    if (preferencesResponse.PropertyPreferencesResponse.PropertyRecommendationBModel.Property.Pet)
        fileStreamWriter.WriteLine("\\t Pet: yes");
    fileStreamWriter.WriteLine("\\tAmenities:");
    if (preferencesResponse.PropertyPreferencesResponse.PropertyRecommendationBModel.Property.AmenitiesPackage.WiFi)

```

```
        fileStreamWriter.WriteLine("\t \t Wifi");

    if (preferencesResponse.PropertyPreferencesResponse.PropertyRecommendationBModel.Property.AmenitiesPackage
        .Kitchen)
        fileStreamWriter.WriteLine("\t \t Kitchen");

    if (preferencesResponse.PropertyPreferencesResponse.PropertyRecommendationBModel.Property.AmenitiesPackage
        .Washer)
        fileStreamWriter.WriteLine("\t \t Washer");

    if (preferencesResponse.PropertyPreferencesResponse.PropertyRecommendationBModel.Property.AmenitiesPackage
        .Dryer)
        fileStreamWriter.WriteLine("\t \t Dryer");

    if (preferencesResponse.PropertyPreferencesResponse.PropertyRecommendationBModel.Property.AmenitiesPackage
        .AirConditioning)
        fileStreamWriter.WriteLine("\t \t Air Conditioning");

    if (preferencesResponse.PropertyPreferencesResponse.PropertyRecommendationBModel.Property.AmenitiesPackage
        .Heating)
        fileStreamWriter.WriteLine("\t \t Heating");

    if (preferencesResponse.PropertyPreferencesResponse.PropertyRecommendationBModel.Property.AmenitiesPackage.Tv)
        fileStreamWriter.WriteLine("\t \t TV");

    if (preferencesResponse.PropertyPreferencesResponse.PropertyRecommendationBModel.Property.AmenitiesPackage.Iron)
        fileStreamWriter.WriteLine("\t \t Iron");

    fileStreamWriter.WriteLine(
        $"{Property type:
    {preferencesResponse.PropertyPreferencesResponse.PropertyRecommendationBModel.Property.PropertyType.Type}"}");
    fileStreamWriter.WriteLine(
        $"{Place type:
    {preferencesResponse.PropertyPreferencesResponse.PropertyRecommendationBModel.Property.PlaceType.Type}"}");
    fileStreamWriter.WriteLine(
        $"{Bedrooms:
    {preferencesResponse.PropertyPreferencesResponse.PropertyRecommendationBModel.Property.RoomAndBed.Bedroom}"}");
    fileStreamWriter.WriteLine(
        $"{Beds:
    {preferencesResponse.PropertyPreferencesResponse.PropertyRecommendationBModel.Property.RoomAndBed.Bed}"}");
    fileStreamWriter.WriteLine(
        $"{Bathrooms:
    {preferencesResponse.PropertyPreferencesResponse.PropertyRecommendationBModel.Property.RoomAndBed.Bathroom}"}");

    fileStreamWriter.WriteLine(
        $"{Initial assigned agent name:
    {preferencesResponse.PropertyPreferencesResponse.PropertyRecommendationBModel.InitialAssignedAgentName}"}");
    fileStreamWriter.WriteLine($"{Source agent name: {propertySourceAgentName}");

    fileStreamWriter.WriteLine("Attractions recommendations ");
    foreach (var attraction in preferencesResponse.AttractionsRecommendationResponse!.AttractionRecommendationList)
    {
        fileStreamWriter.WriteLine($"{City: {attraction.Attraction.Town}");
        fileStreamWriter.WriteLine($"{Name: {attraction.Attraction.Name}");
        fileStreamWriter.WriteLine($"{Kind pattern: {attraction.Attraction.Kinds}");
        fileStreamWriter.WriteLine();
    }

    fileStreamWriter.WriteLine(
        $"{Initial assigned agent name:
    {preferencesResponse.AttractionsRecommendationResponse.InitialAssignedAgentName}");
    fileStreamWriter.WriteLine($"{Source agent name: {attractionsSourceAgentName}");

    fileStreamWriter.WriteLine("_____");
}

public static void WriteUserEvaluation(ServiceEvaluationDto serviceEvaluation)
{
    using var fileStreamWriter = File.AppendText(_pathToLogFile + _fileName);
    fileStreamWriter.WriteLine("_____");
```

```

fileStreamWriter.WriteLine("User's services evaluations");
fileStreamWriter.WriteLine("\nFlight evaluation");
fileStreamWriter.WriteLine("Departure flight evaluation");
fileStreamWriter.WriteLine(
    $"{Class: {(serviceEvaluation.FlightEvaluation.DepartureNavigation.Class ? "yes" : "no")}}");
fileStreamWriter.WriteLine(
    $"{Price: {(serviceEvaluation.FlightEvaluation.DepartureNavigation.Price ? "yes" : "no")}}");
fileStreamWriter.WriteLine(
    $"{Proposed Flight Date: {(serviceEvaluation.FlightEvaluation.DepartureNavigation.FlightDate ? "yes" : "no")}}");
fileStreamWriter.WriteLine(
    $"{Flight Time: {(serviceEvaluation.FlightEvaluation.DepartureNavigation.FlightTime ? "yes" : "no")}}");
fileStreamWriter.WriteLine(
    $"{Flight Company: {(serviceEvaluation.FlightEvaluation.DepartureNavigation.Company ? "yes" : "no")}}");
fileStreamWriter.WriteLine(
    $"{Flight Rating: {Math.Round((double) serviceEvaluation.FlightEvaluation.DepartureNavigation.FlightRating!, 2)}}");

fileStreamWriter.WriteLine("\n Return flight evaluation");
fileStreamWriter.WriteLine(
    $"{Class: {(serviceEvaluation.FlightEvaluation.ReturnNavigation.Class ? "yes" : "no")}}");
fileStreamWriter.WriteLine(
    $"{Price: {(serviceEvaluation.FlightEvaluation.ReturnNavigation.Price ? "yes" : "no")}}");
fileStreamWriter.WriteLine(
    $"{Proposed Flight Date: {(serviceEvaluation.FlightEvaluation.ReturnNavigation.FlightDate ? "yes" : "no")}}");
fileStreamWriter.WriteLine(
    $"{Flight Time: {(serviceEvaluation.FlightEvaluation.ReturnNavigation.FlightTime ? "yes" : "no")}}");
fileStreamWriter.WriteLine(
    $"{Flight Company: {(serviceEvaluation.FlightEvaluation.ReturnNavigation.Company ? "yes" : "no")}}");
fileStreamWriter.WriteLine(
    $"{Flight Rating: {Math.Round((double) serviceEvaluation.FlightEvaluation.ReturnNavigation.FlightRating!, 2)}}");

fileStreamWriter.WriteLine(
    $"{\n \t Entire Flight Rating: {Math.Round((double) serviceEvaluation.FlightEvaluation.FinalFlightRating!, 2)}}");

fileStreamWriter.WriteLine("\nProperty evaluation");
fileStreamWriter.WriteLine(
    $"{Property type: {(serviceEvaluation.PropertyEvaluation.PropertyType ? "yes" : "no")}}");
fileStreamWriter.WriteLine(
    $"{Place type: {(serviceEvaluation.PropertyEvaluation.PlaceType ? "yes" : "no")}}");
fileStreamWriter.WriteLine(
    $"{Rooms and Beds: {(serviceEvaluation.PropertyEvaluation.RoomsAndBeds ? "yes" : "no")}}");
fileStreamWriter.WriteLine(
    $"{Amenities: {(serviceEvaluation.PropertyEvaluation.Amenities ? "yes" : "no")}}");
fileStreamWriter.WriteLine(
    $"{Rating: {Math.Round((double) serviceEvaluation.PropertyEvaluation.FinalPropertyRating!, 2)}}");

fileStreamWriter.WriteLine("\nAttractions evaluation");
foreach (var attractionEvaluation in serviceEvaluation.AttractionEvaluation.AttractionEvaluations)
{
    fileStreamWriter.WriteLine($"{\t Evaluated attraction: {attractionEvaluation.AttractionName}");
    fileStreamWriter.WriteLine($"{\t Liked: {(attractionEvaluation.Rate ? "yes" : "no")}}");
}

fileStreamWriter.WriteLine(
    $"{\n \t Final rating: {Math.Round((double) serviceEvaluation.AttractionEvaluation.FinalAttractionEvaluation!, 2)}}");

fileStreamWriter.WriteLine("_____");
}

public static void WriteCustomerPersonalAgentRate(List<PersonalAgentRateLogModel> personalAgentRateLogList,
    bool isFirstCall = true)
{
    using var fileStreamWriter = File.AppendText(_pathToLogFile + _fileName);
    fileStreamWriter.WriteLine("_____");

    fileStreamWriter.WriteLine("Personal agent rates");

    fileStreamWriter.WriteLine($"{\t \t \t \t FlightRating \t PropertyRating \t AttractionsRating");

```



```

        foreach (var customerPersonalAgentRate in personalAgentRateLogList)
        {
            fileStreamWriter.WriteLine(
                $"{customerPersonalAgentRate.AgentName} \t \t {Math.Round(customerPersonalAgentRate.FlightExpertRate, 2)} \t \t {Math.Round(customerPersonalAgentRate.PropertyExpertRate, 2)} \t \t \t {Math.Round(customerPersonalAgentRate.AttractionsExpertRate, 2)}");

            fileStreamWriter.WriteLine("_____");
        }

public static void WriteCustomerPersonalAgentServiceRecommendationsCounter(
    PersonalAgentServiceScoreLogModel personalAgentServiceScoreLogModel)
{
    using var fileStreamWriter = File.AppendText(_pathToLogFile + _fileName);
    if (_personalAgentServiceScoreHeaderInitialized == false)
    {
        _personalAgentServiceScoreHeaderInitialized = true;
        fileStreamWriter.WriteLine("_____");
        fileStreamWriter.WriteLine("\nPersonal agents services score");
        fileStreamWriter.WriteLine("\t \t \t \t FlightScore \t PropertyScore \t AttractionsScore");
    }

    fileStreamWriter.WriteLine(
        $"{personalAgentServiceScoreLogModel.AgentName} \t \t {personalAgentServiceScoreLogModel.FlightRecommendationsDoneForCurrentUser} \t \t {personalAgentServiceScoreLogModel.PropertyRecommendationsDoneForCurrentUser} \t \t \t {personalAgentServiceScoreLogModel.AttractionsRecommendationsDoneForCurrentUser}");
}

public static void WriteAgentsUpdatedSelfExpertRate(
    PersonalAgentSelfExpertRateLogModel personalAgentSelfExpertRateLogModel)
{
    using var fileStreamWriter = File.AppendText(_pathToLogFile + _fileName);
    if (_currentProcessingAgentSelfExpertRate == string.Empty)
        fileStreamWriter.WriteLine("\nAgents updating self expert logic");

    if (_currentProcessingAgentSelfExpertRate != personalAgentSelfExpertRateLogModel.AgentName)
    {
        _currentProcessingAgentSelfExpertRate = personalAgentSelfExpertRateLogModel.AgentName;
        _currentProcessingAgentSelfExpertService = personalAgentSelfExpertRateLogModel.ServiceType;

        fileStreamWriter.WriteLine($"Name: {personalAgentSelfExpertRateLogModel.AgentName}");
        fileStreamWriter.WriteLine($"Service type: {personalAgentSelfExpertRateLogModel.ServiceType}");
    }

    if (_currentProcessingAgentSelfExpertService != personalAgentSelfExpertRateLogModel.ServiceType)
    {
        _currentProcessingAgentSelfExpertService = personalAgentSelfExpertRateLogModel.ServiceType;
        fileStreamWriter.WriteLine($"Service type: {personalAgentSelfExpertRateLogModel.ServiceType}");
    }

    fileStreamWriter.WriteLine($"Date of request: {personalAgentSelfExpertRateLogModel.DateOfRequest}");
    fileStreamWriter.WriteLine(
        $"Difference between today and request date: {personalAgentSelfExpertRateLogModel.DaysDifferenceFromToday}");
    fileStreamWriter.WriteLine(
        $"Original value: {personalAgentSelfExpertRateLogModel.ExpertServiceRatings.OriginalValue}");
    fileStreamWriter.WriteLine(
        $"Actual value: {personalAgentSelfExpertRateLogModel.ExpertServiceRatings.ActualValue}");
}

public static void ClearCurrentProcessingAgentSelfExpertLogData()
{
    _currentProcessingAgentSelfExpertRate = string.Empty;
    _currentProcessingAgentSelfExpertService = string.Empty;
    _personalAgentServiceScoreHeaderInitialized = false;
}
}

```

TimeoutFunctionHandler.cs

```

using VacationPackageWebApi.Domain.Mas.Singleton;

namespace VacationPackageWebApi.Domain.Helpers;

public static class TimeoutFunctionHandler
{
    public static async Task CheckRecommendationReadyUntilSuccessOrTimeout(CancellationTokentoken cancellationToken)
    {
        try
        {
            await Task.Run(async () =>
            {
                while (!cancellationToken.IsCancellationRequested)
                {
                    await Task.Delay(50, cancellationToken);
                    if (MasEnvironmentSingleton.Instance.Memory["PreferencesResponseStatus"] == "done")
                    {
                        MasEnvironmentSingleton.Instance.Memory["PreferencesResponseStatus"] = "undone";
                        return;
                    }
                }
            }, cancellationToken);
        }
        catch (TaskCanceledException)
        {
            if (!cancellationToken.IsCancellationRequested) throw;
        }
    }
}

```

AgentServiceRating.cs

```

namespace VacationPackageWebApi.Domain.Mas.AgentsExpertBusinessModel;

public record AgentServiceRating
{
    public Guid AgentId { get; set; }
    public float ServiceRating { get; set; }
    public DateTime ServiceEvaluationDate { get; set; }
}

```

AttractionsRecommendationLogic.cs

```

using VacationPackageWebApi.Domain.AgentsEnvironment.AgentModels;
using VacationPackageWebApi.Domain.Attractions;
using VacationPackageWebApi.Domain.Mas.Singleton;
using VacationPackageWebApi.Domain.PreferencesPackageRequest;
using VacationPackageWebApi.Domain.PreferencesPackageResponse;
using VacationPackageWebApi.Domain.PreferencesPackageResponse.AttractionsPreferencesResponse;

namespace VacationPackageWebApi.Domain.Mas.BusinessLogic;

public static class AttractionsRecommendationLogic
{
    private static readonly Random Random = new();

    public static bool FindOptimalAttractionAndStoreInMemory(Guid sourceAgentId,
        string initialAssignedAgentName, object recommendationPopulationLock, TourismAgent tourismAgent,
        PreferencesRequest preferencesRequest)
    {
        var cityAttractions = FindAttractionsByCity(tourismAgent, preferencesRequest.DestinationCityNavigation.Name);

        if (cityAttractions.Any())
        {
            var topAttractions =
                FindOptimalTopAttractionsBasedOnUserPreferences(preferencesRequest, cityAttractions);

            var attractionsRecommendationList = ConvertToAttractionRecommendationBModelList(topAttractions);
        }
    }
}

```



```
        StoreInMemoryAttractionsRecommendations(recommendationPopulationLock, sourceAgentId,
            initialAssignedAgentName, attractionsRecommendationList);
        return true;
    }

    return false;
}

private static void StoreInMemoryAttractionsRecommendations(object recommendationPopulationLock, Guid sourceAgentId,
    string initialAssignedAgentName, List<AttractionRecommendationBModel> attractionsRecommendationList)
{
    var attractionsRecommendationResponse =
        PopulateAttractionsRecommendationResponse(attractionsRecommendationList);
    attractionsRecommendationResponse.SourceAgentId = sourceAgentId;
    attractionsRecommendationResponse.InitialAssignedAgentName = initialAssignedAgentName;

    lock (recommendationPopulationLock)
    {
        (MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] as PreferencesResponse)!
            .AttractionsRecommendationResponse ??= attractionsRecommendationResponse;
    }
}

private static AttractionsRecommendationResponse PopulateAttractionsRecommendationResponse(
    List<AttractionRecommendationBModel> attractionsRecommendationList)
{
    return new AttractionsRecommendationResponse
    {
        AttractionRecommendationList = attractionsRecommendationList
    };
}

private static List<AttractionRecommendationBModel> ConvertToAttractionRecommendationBModelList(
    List<AttractionBusinessModel> topAttractions)
{
    return topAttractions.Select(attraction => new AttractionRecommendationBModel { Attraction = attraction })
        .ToList();
}

private static List<AttractionBusinessModel> FindOptimalTopAttractionsBasedOnUserPreferences(
    PreferencesRequest preferencesRequest, HashSet<AttractionBusinessModel> cityAttractions)
{
    var topAttractions = new List<AttractionBusinessModel>();

    if (preferencesRequest.CustomerAttractionNavigation == null)
    {
        if (cityAttractions.Count > 5)
            topAttractions = cityAttractions.OrderBy(_ => Random.Next()).Take(5).ToList();
        else
            return cityAttractions.ToList();

        return topAttractions;
    }

    var reserveAttractions = new List<AttractionBusinessModel>();

    foreach (var attraction in cityAttractions)
    {
        if (preferencesRequest.CustomerAttractionNavigation.Architecture &&
            attraction.Kinds.Contains("architecture"))
            topAttractions.Add(attraction);

        else if (preferencesRequest.CustomerAttractionNavigation.Cultural && attraction.Kinds.Contains("cultural"))
            topAttractions.Add(attraction);

        else if (preferencesRequest.CustomerAttractionNavigation.Historical &&
            attraction.Kinds.Contains("historic"))
            topAttractions.Add(attraction);
    }
}
```

```

else if (preferencesRequest.CustomerAttractionNavigation.Natural && attraction.Kinds.Contains("natural"))
    topAttractions.Add(attraction);

else if (preferencesRequest.CustomerAttractionNavigation.Other && attraction.Kinds.Contains("other"))
    topAttractions.Add(attraction);

else if (preferencesRequest.CustomerAttractionNavigation.Religion && attraction.Kinds.Contains("religion"))
    topAttractions.Add(attraction);

else if (preferencesRequest.CustomerAttractionNavigation.IndustrialFacilities &&
    attraction.Kinds.Contains("industrial"))
    topAttractions.Add(attraction);
else
    reserveAttractions.Add(attraction);

if (topAttractions.Count >= 5) return topAttractions;
if (cityAttractions.Last().Xid == attraction.Xid)
    if (reserveAttractions.Count != 0)
    {
        var topAttractionsNeedMore = 5 - topAttractions.Count;
        topAttractions.AddRange(reserveAttractions.Count >= topAttractionsNeedMore
            ? reserveAttractions.Take(topAttractionsNeedMore)
            : reserveAttractions);
    }
}

return topAttractions;
}

private static HashSet<AttractionBusinessModel> FindAttractionsByCity(AgentLocalDb agentLocalDb, string city)
{
    return agentLocalDb.AttractionsList.Where(a => a.Town == city).ToHashSet();
}

```

CommonRecommendationLogic.cs

```

using VacationPackageWebApi.Domain.AgentsEnvironment.AgentModels;
using VacationPackageWebApi.Domain.Enums;
using VacationPackageWebApi.Domain.Mas.Singleton;
using VacationPackageWebApi.Domain.PreferencesPackageRequest;
using VacationPackageWebApi.Domain.PreferencesPackageResponse;

namespace VacationPackageWebApi.Domain.Mas.BusinessLogic;

public static class CommonRecommendationLogic
{
    public const int Match = 1;
    public const int NoMatch = 0;
    private static readonly Random Random = new();

    public static TaskType AccessPreferencesAndChoseTask(object taskDistributionLock, TourismAgent agent)
    {
        lock (taskDistributionLock)
        {
            var availableTasks = GetListOfAvailableTasks()?.ToList();

            if (availableTasks == null) return TaskType.Default;
            if (!availableTasks.Any()) return TaskType.Default;

            if (agent.ConfInd[TaskType.Flight] == 0 && agent.ConfInd[TaskType.Property] == 0 &&
                agent.ConfInd[TaskType.Attractions] == 0)
            {
                var randomIndexTaskType = Random.Next(availableTasks.Count);
                RemoveTaskFromAvailableTasks(availableTasks[randomIndexTaskType]);
                RemoveAgentFromAvailableAgentsList(agent.Name);
                return availableTasks[randomIndexTaskType];
            }
        }
    }
}

```

```
foreach (var (key, _) in agent.ConfInd.OrderByDescending(t => t.Value))
{
    if (!availableTasks.Contains(key)) continue;
    RemoveTaskFromAvailableTasks(key);
    RemoveAgentFromAvailableAgentsList(agent.Name);
    agent.Status = false;
    return key;
}

return TaskType.Default;
}

public static void StoreAgentsTrustRate(Dictionary<Guid, List<TrustAgentRateBusinessModel>> agentsTrustRates)
{
    MasEnvironmentSingleton.Instance.Memory["AgentsTrustRates"] = agentsTrustRates;
}

public static List<TrustAgentRateBusinessModel>? GetAgentTrustRateOfAgentWithId(Guid id)
{
    return (MasEnvironmentSingleton.Instance.Memory["AgentsTrustRates"] as
        Dictionary<Guid, List<TrustAgentRateBusinessModel>?>)[id];
}

public static Dictionary<TaskType, float> GetCurrentAgentCustomizedExpertRate(Guid currentAgentId,
    Dictionary<Guid, Dictionary<TaskType, float>> customizedExpertAgentRates)
{
    if (customizedExpertAgentRates.Count != 0)
        return customizedExpertAgentRates.SingleOrDefault(r => r.Key == currentAgentId).Value;

    var defaultCustomizedExpertRate = new Dictionary<TaskType, float>();
    for (var taskType = TaskType.Flight; taskType < TaskType.Default; taskType++)
        defaultCustomizedExpertRate.Add(taskType, 0.0f);

    return defaultCustomizedExpertRate;
}

private static List<TaskType>? GetListOfAvailableTasks()
{
    return MasEnvironmentSingleton.Instance.Memory["AvailableTasks"];
}

private static void RemoveTaskFromAvailableTasks(TaskType task)
{
    (MasEnvironmentSingleton.Instance.Memory["AvailableTasks"] as List<TaskType>!).Remove(task);
}

public static Task InsertAgentNameToAvailableAgents(string agentName)
{
    (MasEnvironmentSingleton.Instance.Memory["AvailableAgents"] as List<string>!).Add(agentName);
    return Task.CompletedTask;
}

public static void SetPreferencesResponseStatusDone()
{
    MasEnvironmentSingleton.Instance.Memory["PreferencesResponseStatus"] = "done";
}

public static bool IsDepartureFlightRecommendationDone()
{
    var preferencesResponse = MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] as PreferencesResponse;
    return preferencesResponse?.FlightRecommendationResponse?.FlightDirectionRecommendation
        ?.DepartureFlightRecommendation != null;
}

public static bool IsReturnFlightRecommendationDone()
{
    var preferencesResponse = MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] as PreferencesResponse;
    return preferencesResponse?.FlightRecommendationResponse?.FlightDirectionRecommendation
```

```

        ?.ReturnFlightRecommendation != null;
    }

    public static bool IsPropertyRecommendationDone()
    {
        var preferencesResponse = MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] as PreferencesResponse;
        return preferencesResponse?.PropertyPreferencesResponse?.PropertyRecommendationBModel != null;
    }

    public static bool IsAttractionsRecommendationDone()
    {
        var preferencesResponse = MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] as PreferencesResponse;
        return preferencesResponse?.AttractionsRecommendationResponse != null;
    }

    public static PreferencesRequest GetPreferencesPayload()
    {
        return MasEnvironmentSingleton.Instance.Memory["PreferencesPayload"];
    }

    public static Task SetPreferencesPayload(PreferencesRequest preferences)
    {
        MasEnvironmentSingleton.Instance.Memory["PreferencesPayload"] = preferences;
        return Task.CompletedTask;
    }

    public static void RemoveAgentFromAvailableAgentsList(string name)
    {
        (MasEnvironmentSingleton.Instance.Memory["AvailableAgents"] as List<string>!).Remove(name);
    }

    public static Task<List<string>> GetListOfAvailableAgentsAsync()
    {
        return Task.FromResult((MasEnvironmentSingleton.Instance.Memory["AvailableAgents"] as List<string>!));
    }

    public static Task<List<string>> GetListOfAllAgentsExceptCurrentAndCoordinator(TourismAgent tourismAgent)
    {
        var availableAgents = MasEnvironmentSingleton.Instance.AllAgents();
        if (availableAgents.Count > 2)
        {
            availableAgents.Remove(tourismAgent.Name);
            availableAgents.Remove("Coordinator");
        }

        return Task.FromResult(availableAgents.ToList());
    }
}

```

FlightRecommendationLogic.cs

```

using VacationPackageWebApi.Domain.AgentsEnvironment.AgentModels;
using VacationPackageWebApi.Domain.Enums;
using VacationPackageWebApi.Domain.Flight;
using VacationPackageWebApi.Domain.Helpers;
using VacationPackageWebApi.Domain.Mas.Singleton;
using VacationPackageWebApi.Domain.PreferencesPackageRequest;
using VacationPackageWebApi.Domain.PreferencesPackageResponse;
using VacationPackageWebApi.Domain.PreferencesPackageResponse.FlightPreferencesResponse;
using static VacationPackageWebApi.Domain.Mas.BusinessLogic.CommonRecommendationLogic;

namespace VacationPackageWebApi.Domain.Mas.BusinessLogic;

public static class FlightRecommendationLogic
{
    private const int TotalFlightElementsToMatch = 3;
    private static readonly Random Random = new();

    private static FlightBusinessModel FindOptimalFlightByDirectionByUserPreference(

```

```
PreferencesRequest preferencesRequest,
HashSet<FlightBusinessModel> flights, string direction)
{
    var bestSimilarityRate = 0.0d;
    var randomFlightIndex = Random.Next(flights.Count);
    var optimalFlight = flights.ElementAt(randomFlightIndex);

    foreach (var flight in flights)
    {
        var currentSimilarityRate = CalculateFlightSimilarityRate(preferencesRequest, flight, direction);
        if (currentSimilarityRate > bestSimilarityRate)
        {
            optimalFlight = flight;
            bestSimilarityRate = currentSimilarityRate;
        }
    }

    return optimalFlight;
}

private static double CalculateFlightSimilarityRate(PreferencesRequest preferencesRequest,
FlightBusinessModel flight, string flightDirection)
{
    var groupFlightTimeByDayPeriods = GroupFlightTimeByDayPeriods(flight.AvailableDepartureTime.DepartureHour);

    switch (flightDirection)
    {
        case "Departure":
        {
            var flightDayMatch =
                CheckFlightDayMatch(preferencesRequest.DepartureDate.GetDayOfWeekFromDate(), flight);
            var flightCompaniesMatchDeparture = CheckFlightCompaniesMatch(
                preferencesRequest.CustomerFlightNavigation!.DepartureNavigation!.FlightCompaniesNavigationList,
                flight);
            var preferenceDeparturePeriodMatch = CheckPreferenceDeparturePeriodMatch(
                preferencesRequest.CustomerFlightNavigation.DepartureNavigation.DeparturePeriodPreference,
                groupFlightTimeByDayPeriods);
            return ((double) (flightCompaniesMatchDeparture ? Match : NoMatch) +
                (preferenceDeparturePeriodMatch ? Match : NoMatch) +
                (flightDayMatch ? Match : NoMatch)) /
                TotalFlightElementsToMatch;
        }
        case "Return":
        {
            var returnDate = GetFlightReturnDate(preferencesRequest);
            var flightDayMatch = CheckFlightDayMatch(returnDate.GetDayOfWeekFromDate(), flight);

            var flightCompaniesMatchReturn =
                CheckFlightCompaniesMatch(
                    preferencesRequest.CustomerFlightNavigation!.ReturnNavigation!.FlightCompaniesNavigationList,
                    flight);
            var preferenceReturnPeriodMatch =
                CheckPreferenceDeparturePeriodMatch(
                    preferencesRequest.CustomerFlightNavigation.ReturnNavigation.DeparturePeriodPreference,
                    groupFlightTimeByDayPeriods);
            return ((double) (flightCompaniesMatchReturn ? Match : NoMatch) +
                (preferenceReturnPeriodMatch ? Match : NoMatch) +
                (flightDayMatch ? Match : NoMatch)) /
                TotalFlightElementsToMatch;
        }
    }

    return 0.0d;
}

public static void FulfillFlightDefaultPreferencesWithCheapestOffer(ref PreferencesRequest preferencesRequest)
{
    preferencesRequest.CustomerFlightNavigation ??= new FlightDirectionPreferenceDto();
    preferencesRequest.CustomerFlightNavigation.DepartureNavigation ??= new FlightPreferenceDto();
    preferencesRequest.CustomerFlightNavigation.DepartureNavigation.Class ??= new FlightClassDto
}
```

```

{
    Class = (short) ClassTypeId.Economy
};
preferencesRequest.CustomerFlightNavigation.DepartureNavigation.StopsNavigation ??=
    new StopsTypePreferenceDto
    {
        Type = (short) StopsTypePreferenceId.Direct
    };

preferencesRequest.CustomerFlightNavigation.ReturnNavigation ??= new FlightPreferenceDto();
preferencesRequest.CustomerFlightNavigation.ReturnNavigation.Class ??= new FlightClassDto
{
    Class = (short) ClassTypeId.Economy
};
preferencesRequest.CustomerFlightNavigation.ReturnNavigation.StopsNavigation ??=
    new StopsTypePreferenceDto
    {
        Type = (short) StopsTypePreferenceId.Direct
    };
}

private static bool CheckFlightDayMatch(string dayOfFlightPreferred, FlightBusinessModel flight)
{
    return flight.WeekDaysOfFlight.DaysList.Contains(dayOfFlightPreferred);
}

private static bool CheckFlightCompaniesMatch(List<FlightCompaniesPreferenceDto>? preferredFlightCompanies,
    FlightBusinessModel flight)
{
    if (preferredFlightCompanies == null)
        return false;

    return preferredFlightCompanies.Any(flightCompany =>
        flight.Company.Name == flightCompany.Company.Name);
}

private static TimeOnly? GetEarliestFlightThatMatchWithPreferences(
    DeparturePeriodsPreferenceDto? departurePeriodsPreference,
    Dictionary<DayPeriods, List<TimeOnly>> groupedFlightsByDayPeriod)
{
    if (departurePeriodsPreference.IsFulfilledEarlyMorningPreference(groupedFlightsByDayPeriod))
        if (groupedFlightsByDayPeriod.ContainsKey(DayPeriods.EarlyMorning))
            return groupedFlightsByDayPeriod[DayPeriods.EarlyMorning].First();

    if (departurePeriodsPreference.IsFulfilledMorningPreference(groupedFlightsByDayPeriod))
        if (groupedFlightsByDayPeriod.ContainsKey(DayPeriods.Morning))
            return groupedFlightsByDayPeriod[DayPeriods.Morning].First();

    if (departurePeriodsPreference.IsFulfilledAfternoonPreference(groupedFlightsByDayPeriod))
        if (groupedFlightsByDayPeriod.ContainsKey(DayPeriods.Afternoon))
            return groupedFlightsByDayPeriod[DayPeriods.Afternoon].First();

    if (departurePeriodsPreference.IsFulfilledNightPreference(groupedFlightsByDayPeriod))
        if (groupedFlightsByDayPeriod.ContainsKey(DayPeriods.Night))
            return groupedFlightsByDayPeriod[DayPeriods.Night].First();

    return null;
}

private static TimeOnly GetEarliestFlightTimeBasedOnPeriodPreference(FlightBusinessModel flight,
    DeparturePeriodsPreferenceDto? departurePeriodsPreference)
{
    var groupedFlightsByDayPeriod = GroupFlightTimeByDayPeriods(flight.AvailableDepartureTime.DepartureHour);
    // check each true preference about part of the day with grouped flights by parts of the day
    // if false, get closest one
    TimeOnly? firstAvailableFlightTime = null;

    if (departurePeriodsPreference == null)

```

```
        foreach (var (_, value) in groupedFlightsByDayPeriod)
            if (value.Any())
                return value.First();

firstAvailableFlightTime =
    GetEarliestFlightThatMatchWithPreferences(departurePeriodsPreference, groupedFlightsByDayPeriod);

if (firstAvailableFlightTime != null)
    return (TimeOnly) firstAvailableFlightTime;

// if no matches, get closest flight for user preferred period. First find the earliest preferred and search
// for latest flights. If not found, search early than earliest preferred.

var earliestDayPeriodPreference = GetEarliestDayPeriodPreference(departurePeriodsPreference!);

if (earliestDayPeriodPreference != null)
{
    for (var dayPeriod = (DayPeriods) earliestDayPeriodPreference; dayPeriod <= DayPeriods.Night; dayPeriod++)
    {
        if (dayPeriod == DayPeriods.Night) break;

        var nexDayPeriod = dayPeriod + 1;
        if (groupedFlightsByDayPeriod[nexDayPeriod].Any())
            return groupedFlightsByDayPeriod[nexDayPeriod].First();
    }

    for (var dayPeriod = (DayPeriods) earliestDayPeriodPreference; dayPeriod >= DayPeriods.Morning; dayPeriod--)
    {
        if (dayPeriod == DayPeriods.Morning) return TimeOnly.MinValue;

        var previousDayPeriod = dayPeriod - 1;
        if (groupedFlightsByDayPeriod[previousDayPeriod].Any())
            return groupedFlightsByDayPeriod[previousDayPeriod].First();
    }
}

return TimeOnly.MinValue;
}

private static DayPeriods? GetEarliestDayPeriodPreference(DeparturePeriodsPreferenceDto departurePeriodsPreference)
{
    return departurePeriodsPreference.EarlyMorning ? DayPeriods.EarlyMorning :
        departurePeriodsPreference.Morning ? DayPeriods.Morning :
        departurePeriodsPreference.Afternoon ? DayPeriods.Afternoon :
        departurePeriodsPreference.Night ? DayPeriods.Night : null;
}

private static Dictionary<DayPeriods, List<TimeOnly>> GroupFlightTimeByDayPeriods(string flightDepartureTimeList)
{
    var departureHours = flightDepartureTimeList.ConvertStringTimeListToTimeOnly();
    var flightGroup = new Dictionary<DayPeriods, List<TimeOnly>>();

    for (var dayPeriod = DayPeriods.EarlyMorning; dayPeriod <= DayPeriods.Night; dayPeriod++)
        flightGroup.Add(dayPeriod, new List<TimeOnly>());

    foreach (var hour in departureHours)
    {
        if (hour.IsEarlyMorningTime())
        {
            flightGroup[DayPeriods.EarlyMorning].Add(hour);
            continue;
        }

        if (hour.IsMorningTime())
        {
            flightGroup[DayPeriods.Morning].Add(hour);
            continue;
        }

        if (hour.IsAfternoonTime())
```

```

    {
        flightGroup[DayPeriods.Afternoon].Add(hour);
        continue;
    }

    if (hour.IsNightTime()) flightGroup[DayPeriods.Night].Add(hour);
}

var sortedHours = new Dictionary<DayPeriods, List<TimeOnly>>>();
foreach (var (key, value) in flightGroup)
{
    var orderedTime = value.OrderBy(x => x).ToList();
    sortedHours.Add(key, orderedTime);
}

return sortedHours;
}

private static void StoreInMemoryDepartureFlightRecommendation(object recommendationPopulationLock,
    FlightRecommendationBModel flightRecommendation)
{
    lock (recommendationPopulationLock)
    {
        InitializeFlightRecommendationResponseIfNull();

        (MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] as PreferencesResponse)!
            .FlightRecommendationResponse!
            .FlightDirectionRecommendation!.DepartureFlightRecommendation ??= flightRecommendation;
    }
}

private static void StoreInMemoryReturnFlightRecommendation(object recommendationPopulationLock,
    FlightRecommendationBModel flightRecommendation)
{
    lock (recommendationPopulationLock)
    {
        InitializeFlightRecommendationResponseIfNull();

        (MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] as PreferencesResponse)!
            .FlightRecommendationResponse!
            .FlightDirectionRecommendation!.ReturnFlightRecommendation ??= flightRecommendation;
    }
}

private static void InitializeFlightRecommendationResponseIfNull()
{
    (MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] as PreferencesResponse)!
        .FlightRecommendationResponse ??= new FlightRecommendationResponse();

    (MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] as PreferencesResponse)!
        .FlightRecommendationResponse!.FlightDirectionRecommendation ??= new FlightDirectionRecommendationBModel();
}

public static bool FindOptimalDepartureFlightAndStoreInMemory(Guid sourceAgentId,
    string initialAssignedAgentName, object recommendationPopulationLock, TourismAgent tourismAgent,
    PreferencesRequest preferencesRequest)
{
    var departureCityFlights = FindFlightsByDepartureByDestination(tourismAgent,
        preferencesRequest.DepartureCityNavigation.Name, preferencesRequest.DestinationCityNavigation.Name);

    if (departureCityFlights.Any())
    {
        if (preferencesRequest.CustomerFlightNavigation?.DepartureNavigation == null)
        {
            var randomOptimalDepartureFlight =
                departureCityFlights.ElementAt(Random.Next(departureCityFlights.Count));

            var departureWeekDaysOfRandomFlight = randomOptimalDepartureFlight.WeekDaysOfFlight.DaysList
                .ConvertStringDaysOfWeekListToEnumList().ToList();

```



```
var earliestDepartureDateOfRandomFlight =
    (DateTime) GetTheEarliestDepartureDateOfFlight(departureWeekDaysOfRandomFlight,
        preferencesRequest.DepartureDate!);
preferencesRequest.DepartureDate = earliestDepartureDateOfRandomFlight;

var mostOptimalFlightTime = randomOptimalDepartureFlight.AvailableDepartureTime.DepartureHour
    .ConvertStringTimeListToTimeOnly().First();
var randomDepartureFlightRecommendation = CreateFlightRecommendationBModel(sourceAgentId,
    initialAssignedAgentName,
    randomOptimalDepartureFlight, (short) ClassTypeId.Economy,
    earliestDepartureDateOfRandomFlight, mostOptimalFlightTime);

StoreInMemoryDepartureFlightRecommendation(recommendationPopulationLock,
    randomDepartureFlightRecommendation);
return true;
}

var optimalDepartureFlight =
    FindOptimalFlightByDirectionByUserPreference(preferencesRequest, departureCityFlights,
        "Departure");

preferencesRequest.CustomerFlightNavigation.DepartureNavigation.Class ??= new FlightClassDto
{
    Class = (short) ClassTypeId.Economy
};

var departureWeekFlight = optimalDepartureFlight.WeekDaysOfFlight.DaysList
    .ConvertStringDaysOfWeekListToEnumList().ToList();
var earliestDepartureDateOfFlight =
    (DateTime) GetTheEarliestDepartureDateOfFlight(departureWeekFlight, preferencesRequest.DepartureDate!);
preferencesRequest.DepartureDate = earliestDepartureDateOfFlight;

var earliestFlightTime = GetEarliestFlightTimeBasedOnPeriodPreference(optimalDepartureFlight,
    preferencesRequest.CustomerFlightNavigation.DepartureNavigation.DeparturePeriodPreference);

var departureFlightRecommendation = CreateFlightRecommendationBModel(sourceAgentId,
    initialAssignedAgentName,
    optimalDepartureFlight, preferencesRequest.CustomerFlightNavigation.DepartureNavigation.Class.Class,
    earliestDepartureDateOfFlight, earliestFlightTime);

StoreInMemoryDepartureFlightRecommendation(recommendationPopulationLock, departureFlightRecommendation);
return true;
}

return false;
}

public static bool FindOptimalReturnFlightAndStoreInMemory(Guid sourceAgentId,
    string initialAssignedAgentName, object recommendationPopulationLock, TourismAgent tourismAgent,
    PreferencesRequest preferencesRequest)
{
    var returnCityFlights = FindFlightsByDepartureByDestination(tourismAgent,
        preferencesRequest.DestinationCityNavigation.Name, preferencesRequest.DepartureCityNavigation.Name);

    if (returnCityFlights.Any())
    {
        if (preferencesRequest.CustomerFlightNavigation?.ReturnNavigation == null)
        {
            var randomOptimalReturnFlight = returnCityFlights.ElementAt(Random.Next(returnCityFlights.Count));

            var preferredReturnFlightDate =
                preferencesRequest.DepartureDate.AddDays(preferencesRequest.HolidaysPeriod);

            var returnWeekDaysOfRandomFlight = randomOptimalReturnFlight.WeekDaysOfFlight.DaysList
                .ConvertStringDaysOfWeekListToEnumList().ToList();
            var earliestReturnDateOfRandomFlight = (DateTime) GetTheMostOptimalReturnDateOfFlight(
                returnWeekDaysOfRandomFlight, preferencesRequest.DepartureDate, preferredReturnFlightDate,
                preferencesRequest.HolidaysPeriod!);

            var mostOptimalFlightTime = randomOptimalReturnFlight.AvailableDepartureTime.DepartureHour
```

```

        .ConvertStringTimeListToTimeOnly().First();
    var randomReturnFlightRecommendation = CreateFlightRecommendationBModel(sourceAgentId,
        initialAssignedAgentName,
        randomOptimalReturnFlight, (short) ClassTypeId.Economy,
        earliestReturnDateOfRandomFlight, mostOptimalFlightTime);

    StoreInMemoryDepartureFlightRecommendation(recommendationPopulationLock,
        randomReturnFlightRecommendation);
    return true;
}

var optimalArrivalFlight =
    FindOptimalFlightByDirectionByUserPreference(preferencesRequest, returnCityFlights,
        "Return");

preferencesRequest.CustomerFlightNavigation.ReturnNavigation.Class ??= new FlightClassDto
{
    Class = (short) ClassTypeId.Economy
};

var earliestFlightTime = GetEarliestFlightTimeBasedOnPeriodPreference(optimalArrivalFlight,
    preferencesRequest.CustomerFlightNavigation.ReturnNavigation.DeparturePeriodPreference);
var flightReturnDate = GetFlightReturnDate(preferencesRequest);

var returnWeekDaysOfFlight = optimalArrivalFlight.WeekDaysOfFlight.DaysList
    .ConvertStringDaysOfWeekListToEnumList().ToList();
var earliestReturnDateOfFlight = (DateTime) GetTheMostOptimalReturnDateOfFlight(returnWeekDaysOfFlight,
    preferencesRequest.DepartureDate, flightReturnDate, preferencesRequest.HolidaysPeriod!);

var returnFlightRecommendation = CreateFlightRecommendationBModel(sourceAgentId, initialAssignedAgentName,
    optimalArrivalFlight, preferencesRequest.CustomerFlightNavigation.ReturnNavigation.Class,
    earliestReturnDateOfFlight, earliestFlightTime);

StoreInMemoryReturnFlightRecommendation(recommendationPopulationLock, returnFlightRecommendation);
return true;
}

return false;
}

private static DateTime? GetTheMostOptimalReturnDateOfFlight(List<DayOfWeek> availableReturnDaysOfWeekFlight,
    DateTime actualDepartureFlightDate, DateTime preferredReturnFlightDate, int vacationPeriod)
{
    if (DateTime.Compare(actualDepartureFlightDate, preferredReturnFlightDate) >= 0)
        preferredReturnFlightDate = actualDepartureFlightDate.AddDays(vacationPeriod);

    var returnPreferredFlightDayOfWeek = preferredReturnFlightDate.DayOfWeek;

    if (!availableReturnDaysOfWeekFlight.Contains(returnPreferredFlightDayOfWeek))
    {
        if (returnPreferredFlightDayOfWeek != DayOfWeek.Sunday)
        {
            for (var dayOfWeekDescInc = returnPreferredFlightDayOfWeek;
                dayOfWeekDescInc >= DayOfWeek.Sunday;
                --dayOfWeekDescInc)
            {
                if (availableReturnDaysOfWeekFlight.Contains(dayOfWeekDescInc))
                {
                    var daysEarlierPreferredReturnFlight = returnPreferredFlightDayOfWeek - dayOfWeekDescInc;
                    var mostEarlyCloseToPreferredReturnFlight =
                        preferredReturnFlightDate.AddDays(-daysEarlierPreferredReturnFlight);

                    if (DateTime.Compare(actualDepartureFlightDate, mostEarlyCloseToPreferredReturnFlight) < 0)
                        return mostEarlyCloseToPreferredReturnFlight;
                }
            }

            for (var weekEarlierInc = DayOfWeek.Saturday;
                weekEarlierInc > returnPreferredFlightDayOfWeek;
                weekEarlierInc--)
            {
                if (availableReturnDaysOfWeekFlight.Contains(weekEarlierInc))

```

```

    {
        var daysEarlierPreferredReturnFlight = (short) returnPreferredFlightDayOfWeek -
            (returnPreferredFlightDayOfWeek - DayOfWeek.Sunday) + (DayOfWeek.Saturday - weekEarlierInc);
        var mostEarlyCloseToPreferredReturnFlight =
            preferredReturnFlightDate.AddDays(-daysEarlierPreferredReturnFlight);

        if (DateTime.Compare(actualDepartureFlightDate, mostEarlyCloseToPreferredReturnFlight) < 0)
            return mostEarlyCloseToPreferredReturnFlight;
    }

    //Above is the search in early days than preferred one.
    //Now it's time to search in later days. It's the same
    //algorithm as for departure flight day search
    var earliestButLaterThanPreferredFlightDay =
        GetTheEarliestDepartureDateOfFlight(availableReturnDaysOfWeekFlight, preferredReturnFlightDate);
    return earliestButLaterThanPreferredFlightDay;
}
else
{
    returnPreferredFlightDayOfWeek = DayOfWeek.Saturday;
    for (var dayOfWeekDescInc = returnPreferredFlightDayOfWeek;
        dayOfWeekDescInc > DayOfWeek.Sunday;
        --dayOfWeekDescInc)
        if (availableReturnDaysOfWeekFlight.Contains(dayOfWeekDescInc))
        {
            var daysEarlierPreferredReturnFlight =
                returnPreferredFlightDayOfWeek - dayOfWeekDescInc +
                1; // +1 because we begun from Saturday, not Sunday
            var mostEarlyCloseToPreferredReturnFlight =
                preferredReturnFlightDate.AddDays(-daysEarlierPreferredReturnFlight);

            if (DateTime.Compare(actualDepartureFlightDate, mostEarlyCloseToPreferredReturnFlight) < 0)
                return mostEarlyCloseToPreferredReturnFlight;
        }

    var earliestButLaterThanPreferredFlightDay =
        GetTheEarliestDepartureDateOfFlight(availableReturnDaysOfWeekFlight, preferredReturnFlightDate);
    return earliestButLaterThanPreferredFlightDay;
}
}

return preferredReturnFlightDate;
}

private static DateTime? GetTheEarliestDepartureDateOfFlight(List<DayOfWeek> daysOfWeekFlight,
    DateTime preferredDepartureFlightDate)
{
    var preferredFlightDayOfWeek = preferredDepartureFlightDate.DayOfWeek;

    if (!daysOfWeekFlight.Contains(preferredFlightDayOfWeek))
    {
        if (preferredFlightDayOfWeek != DayOfWeek.Saturday)
        {
            for (var dayOfWeekAscInc = preferredFlightDayOfWeek + 1;
                dayOfWeekAscInc <= DayOfWeek.Saturday;
                dayOfWeekAscInc++)
                if (daysOfWeekFlight.Contains(dayOfWeekAscInc))
                {
                    //Select day of current week and exit
                    var dayToTheEarliestFlight = dayOfWeekAscInc - preferredFlightDayOfWeek;
                    return preferredDepartureFlightDate.AddDays(dayToTheEarliestFlight);
                }
        }

        for (var nextWeekInc = DayOfWeek.Sunday; nextWeekInc < preferredFlightDayOfWeek; nextWeekInc++)
            if (daysOfWeekFlight.Contains(nextWeekInc))
            {
                var dayToTheEarliestFlight =
                    (short) (DayOfWeek.Saturday - preferredFlightDayOfWeek) + (short) nextWeekInc +
                    1; // +1 because the counter is from 0
            }
    }
}

```

```

        return preferredDepartureFlightDate.AddDays(dayToTheEarliestFlight);
    }
}

for (var nextWeekInc = DayOfWeek.Sunday; nextWeekInc < preferredFlightDayOfWeek; nextWeekInc++)
    if (daysOfWeekFlight.Contains(nextWeekInc))
    {
        var dayToTheEarliestFlight =
            (short) (DayOfWeek.Saturday - preferredFlightDayOfWeek) + (short) nextWeekInc + 1;
        return preferredDepartureFlightDate.AddDays(dayToTheEarliestFlight);
    }
}
else
{
    return preferredDepartureFlightDate;
}

return null;
}

private static bool IsFulfilledEarlyMorningPreference(
    this DeparturePeriodsPreferenceDto? departurePeriodsPreference,
    IReadOnlyDictionary<DayPeriods, List<TimeOnly>> groupedFlightTimes)
{
    return departurePeriodsPreference!.EarlyMorning && groupedFlightTimes[DayPeriods.EarlyMorning].Any();
}

private static bool IsFulfilledMorningPreference(this DeparturePeriodsPreferenceDto? departurePeriodsPreference,
    IReadOnlyDictionary<DayPeriods, List<TimeOnly>> groupedFlightTimes)
{
    return departurePeriodsPreference!.Morning && groupedFlightTimes[DayPeriods.Morning].Any();
}

private static bool IsFulfilledAfternoonPreference(this DeparturePeriodsPreferenceDto? departurePeriodsPreference,
    IReadOnlyDictionary<DayPeriods, List<TimeOnly>> groupedFlightTimes)
{
    return departurePeriodsPreference!.Afternoon && groupedFlightTimes[DayPeriods.Afternoon].Any();
}

private static bool IsFulfilledNightPreference(this DeparturePeriodsPreferenceDto? departurePeriodsPreference,
    IReadOnlyDictionary<DayPeriods, List<TimeOnly>> groupedFlightTimes)
{
    return departurePeriodsPreference!.Night && groupedFlightTimes[DayPeriods.Night].Any();
}

private static bool CheckPreferenceDeparturePeriodMatch(DeparturePeriodsPreferenceDto? departurePeriodsPreference,
    IReadOnlyDictionary<DayPeriods, List<TimeOnly>> groupedFlightTimes)
{
    if (departurePeriodsPreference == null)
        return false;

    return departurePeriodsPreference.IsFulfilledEarlyMorningPreference(groupedFlightTimes) ||
        departurePeriodsPreference.IsFulfilledMorningPreference(groupedFlightTimes) ||
        departurePeriodsPreference.IsFulfilledAfternoonPreference(groupedFlightTimes) ||
        departurePeriodsPreference.IsFulfilledNightPreference(groupedFlightTimes);
}

private static FlightRecommendationBModel CreateFlightRecommendationBModel(Guid sourceAgentId,
    string initialAssignedAgentName, FlightBusinessModel optimalFlight, short flightClass,
    DateTime departureDate, TimeOnly mostOptimalFlightTime)
{
    var optimalTime = mostOptimalFlightTime.ToTimeSpan();
    var sas =
        DateTime.MinValue.AddHours(optimalTime.Hours).AddMinutes(optimalTime.Minutes);
    return new FlightRecommendationBModel
    {
        SourceAgentId = sourceAgentId,
        InitialAssignedAgentName = initialAssignedAgentName,
        DepartureTime =

```

```
        DateTime.MinValue.AddHours(optimalTime.Hours).AddMinutes(optimalTime.Minutes),
        FlightClass = flightClass,
        FlightConnection = new List<FlightConnectionBModel>
        {
            new()
            {
                Flight = optimalFlight
            }
        },
        FlightDate = departureDate,
        Status = "Up-to-date",
        Stops = 0
    };
}

private static HashSet<FlightBusinessModel> FindFlightsByDepartureByDestination(AgentLocalDb agentLocalDb,
    string startFlightCityName, string endFlightCityName)
{
    return agentLocalDb.FlightsList.Where(f =>
        f.DepartureAirport.City.Name == startFlightCityName &&
        f.ArrivalAirport.City.Name == endFlightCityName).ToHashSet();
}

private static DateTime GetFlightReturnDate(PreferencesRequest preferencesRequest)
{
    return preferencesRequest.DepartureDate.AddDays(preferencesRequest.HolidaysPeriod);
}
}using VacationPackageWebApi.Domain.AgentsEnvironment.AgentModels;
using VacationPackageWebApi.Domain.Enums;
using VacationPackageWebApi.Domain.Flight;
using VacationPackageWebApi.Domain.Helpers;
using VacationPackageWebApi.Domain.Mas.Singleton;
using VacationPackageWebApi.Domain.PreferencesPackageRequest;
using VacationPackageWebApi.Domain.PreferencesPackageResponse;
using VacationPackageWebApi.Domain.PreferencesPackageResponse.FlightPreferencesResponse;
using static VacationPackageWebApi.Domain.Mas.BusinessLogic.CommonRecommendationLogic;

namespace VacationPackageWebApi.Domain.Mas.BusinessLogic;

public static class FlightRecommendationLogic
{
    private const int TotalFlightElementsToMatch = 3;
    private static readonly Random Random = new();

    private static FlightBusinessModel FindOptimalFlightByDirectionByUserPreference(
        PreferencesRequest preferencesRequest,
        HashSet<FlightBusinessModel> flights, string direction)
    {
        var bestSimilarityRate = 0.0d;
        var randomFlightIndex = Random.Next(flights.Count);
        var optimalFlight = flights.ElementAt(randomFlightIndex);

        foreach (var flight in flights)
        {
            var currentSimilarityRate = CalculateFlightSimilarityRate(preferencesRequest, flight, direction);
            if (currentSimilarityRate > bestSimilarityRate)
            {
                optimalFlight = flight;
                bestSimilarityRate = currentSimilarityRate;
            }
        }

        return optimalFlight;
    }

    private static double CalculateFlightSimilarityRate(PreferencesRequest preferencesRequest,
        FlightBusinessModel flight, string flightDirection)
    {
        var groupFlightTimeByDayPeriods = GroupFlightTimeByDayPeriods(flight.AvailableDepartureTime.DepartureHour);
```

```

switch (flightDirection)
{
    case "Departure":
    {
        var flightDayMatch =
            CheckFlightDayMatch(preferencesRequest.DepartureDate.GetDayOfWeekFromDate(), flight);
        var flightCompaniesMatchDeparture = CheckFlightCompaniesMatch(
            preferencesRequest.CustomerFlightNavigation!.DepartureNavigation!.FlightCompaniesNavigationList,
            flight);
        var preferenceDeparturePeriodMatch = CheckPreferenceDeparturePeriodMatch(
            preferencesRequest.CustomerFlightNavigation.DepartureNavigation.DeparturePeriodPreference,
            groupFlightTimeByDayPeriods);
        return ((double) (flightCompaniesMatchDeparture ? Match : NoMatch) +
            (preferenceDeparturePeriodMatch ? Match : NoMatch) +
            (flightDayMatch ? Match : NoMatch)) /
            TotalFlightElementsToMatch;
    }
    case "Return":
    {
        var returnDate = GetFlightReturnDate(preferencesRequest);
        var flightDayMatch = CheckFlightDayMatch(returnDate.GetDayOfWeekFromDate(), flight);

        var flightCompaniesMatchReturn =
            CheckFlightCompaniesMatch(
                preferencesRequest.CustomerFlightNavigation!.ReturnNavigation!.FlightCompaniesNavigationList,
                flight);
        var preferenceReturnPeriodMatch =
            CheckPreferenceDeparturePeriodMatch(
                preferencesRequest.CustomerFlightNavigation.ReturnNavigation.DeparturePeriodPreference,
                groupFlightTimeByDayPeriods);
        return ((double) (flightCompaniesMatchReturn ? Match : NoMatch) +
            (preferenceReturnPeriodMatch ? Match : NoMatch) +
            (flightDayMatch ? Match : NoMatch)) /
            TotalFlightElementsToMatch;
    }
}

return 0.0d;
}

public static void FulfillFlightDefaultPreferencesWithCheapestOffer(ref PreferencesRequest preferencesRequest)
{
    preferencesRequest.CustomerFlightNavigation ??= new FlightDirectionPreferenceDto();
    preferencesRequest.CustomerFlightNavigation.DepartureNavigation ??= new FlightPreferenceDto();
    preferencesRequest.CustomerFlightNavigation.DepartureNavigation.Class ??= new FlightClassDto
    {
        Class = (short) ClassTypeId.Economy
    };
    preferencesRequest.CustomerFlightNavigation.DepartureNavigation.StopsNavigation ??=
        new StopsTypePreferenceDto
        {
            Type = (short) StopsTypePreferenceId.Direct
        };

    preferencesRequest.CustomerFlightNavigation.ReturnNavigation ??= new FlightPreferenceDto();
    preferencesRequest.CustomerFlightNavigation.ReturnNavigation.Class ??= new FlightClassDto
    {
        Class = (short) ClassTypeId.Economy
    };
    preferencesRequest.CustomerFlightNavigation.ReturnNavigation.StopsNavigation ??=
        new StopsTypePreferenceDto
        {
            Type = (short) StopsTypePreferenceId.Direct
        };
}

private static bool CheckFlightDayMatch(string dayOfWeekPreferred, FlightBusinessModel flight)
{

```

```
    return flight.WeekDaysOffFlight.DaysList.Contains(dayOfFlightPreferred);
}

private static bool CheckFlightCompaniesMatch(List<FlightCompaniesPreferenceDto>? preferredFlightCompanies,
FlightBusinessModel flight)
{
    if (preferredFlightCompanies == null)
        return false;

    return preferredFlightCompanies.Any(flightCompany =>
        flight.Company.Name == flightCompany.Company.Name);
}

private static TimeOnly? GetEarliestFlightThatMatchWithPreferences(
    DeparturePeriodsPreferenceDto? departurePeriodsPreference,
    Dictionary<DayPeriods, List<TimeOnly>> groupedFlightsByDayPeriod)
{
    if (departurePeriodsPreference.IsFulfilledEarlyMorningPreference(groupedFlightsByDayPeriod))
        if (groupedFlightsByDayPeriod.ContainsKey(DayPeriods.EarlyMorning))
            return groupedFlightsByDayPeriod[DayPeriods.EarlyMorning].First();

    if (departurePeriodsPreference.IsFulfilledMorningPreference(groupedFlightsByDayPeriod))
        if (groupedFlightsByDayPeriod.ContainsKey(DayPeriods.Morning))
            return groupedFlightsByDayPeriod[DayPeriods.Morning].First();

    if (departurePeriodsPreference.IsFulfilledAfternoonPreference(groupedFlightsByDayPeriod))
        if (groupedFlightsByDayPeriod.ContainsKey(DayPeriods.Afternoon))
            return groupedFlightsByDayPeriod[DayPeriods.Afternoon].First();

    if (departurePeriodsPreference.IsFulfilledNightPreference(groupedFlightsByDayPeriod))
        if (groupedFlightsByDayPeriod.ContainsKey(DayPeriods.Night))
            return groupedFlightsByDayPeriod[DayPeriods.Night].First();

    return null;
}

private static TimeOnly GetEarliestFlightTimeBasedOnPeriodPreference(FlightBusinessModel flight,
    DeparturePeriodsPreferenceDto? departurePeriodsPreference)
{
    var groupedFlightsByDayPeriod = GroupFlightTimeByDayPeriods(flight.AvailableDepartureTime, DepartureHour);
    // check each true preference about part of the day with grouped flights by parts of the day
    // if false, get closest one
    TimeOnly? firstAvailableFlightTime = null;

    if (departurePeriodsPreference == null)
        foreach (var (_, value) in groupedFlightsByDayPeriod)
            if (value.Any())
                return value.First();

    firstAvailableFlightTime =
        GetEarliestFlightThatMatchWithPreferences(departurePeriodsPreference, groupedFlightsByDayPeriod);

    if (firstAvailableFlightTime != null)
        return (TimeOnly) firstAvailableFlightTime;

    // if no matches, get closest flight for user preferred period. First find the earliest preferred and search
    // for latest flights. If not found, search early than earliest preferred.

    var earliestDayPeriodPreference = GetEarliestDayPeriodPreference(departurePeriodsPreference!);

    if (earliestDayPeriodPreference != null)
    {
        for (var dayPeriod = (DayPeriods) earliestDayPeriodPreference; dayPeriod <= DayPeriods.Night; dayPeriod++)
        {
            if (dayPeriod == DayPeriods.Night) break;

            var nexDayPeriod = dayPeriod + 1;
            if (groupedFlightsByDayPeriod[nexDayPeriod].Any())
                return groupedFlightsByDayPeriod[nexDayPeriod].First();
        }
    }
}
```

```

    for (var dayPeriod = (DayPeriods) earliestDayPeriodPreference; dayPeriod >= DayPeriods.Morning; dayPeriod--)
    {
        if (dayPeriod == DayPeriods.Morning) return TimeOnly.MinValue;

        var previousDayPeriod = dayPeriod - 1;
        if (groupedFlightsByDayPeriod[previousDayPeriod].Any())
            return groupedFlightsByDayPeriod[previousDayPeriod].First();
    }

    return TimeOnly.MinValue;
}

private static DayPeriods? GetEarliestDayPeriodPreference(DeparturePeriodsPreferenceDto departurePeriodsPreference)
{
    return departurePeriodsPreference.EarlyMorning ? DayPeriods.EarlyMorning :
        departurePeriodsPreference.Morning ? DayPeriods.Morning :
        departurePeriodsPreference.Afternoon ? DayPeriods.Afternoon :
        departurePeriodsPreference.Night ? DayPeriods.Night : null;
}

private static Dictionary<DayPeriods, List<TimeOnly>> GroupFlightTimeByDayPeriods(string flightDepartureTimeList)
{
    var departureHours = flightDepartureTimeList.ConvertStringTimeListToTimeOnly();
    var flightGroup = new Dictionary<DayPeriods, List<TimeOnly>>();

    for (var dayPeriod = DayPeriods.EarlyMorning; dayPeriod <= DayPeriods.Night; dayPeriod++)
        flightGroup.Add(dayPeriod, new List<TimeOnly>());

    foreach (var hour in departureHours)
    {
        if (hour.IsEarlyMorningTime())
        {
            flightGroup[DayPeriods.EarlyMorning].Add(hour);
            continue;
        }

        if (hour.IsMorningTime())
        {
            flightGroup[DayPeriods.Morning].Add(hour);
            continue;
        }

        if (hour.IsAfternoonTime())
        {
            flightGroup[DayPeriods.Afternoon].Add(hour);
            continue;
        }

        if (hour.IsNightTime()) flightGroup[DayPeriods.Night].Add(hour);
    }

    var sortedHours = new Dictionary<DayPeriods, List<TimeOnly>>();
    foreach (var (key, value) in flightGroup)
    {
        var orderedTime = value.OrderBy(x => x).ToList();
        sortedHours.Add(key, orderedTime);
    }

    return sortedHours;
}

private static void StoreInMemoryDepartureFlightRecommendation(object recommendationPopulationLock,
    FlightRecommendationBModel flightRecommendation)
{
    lock (recommendationPopulationLock)
    {
        InitializeFlightRecommendationResponseIfNull();
    }
}

```



```

        (MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] as PreferencesResponse)!
        .FlightRecommendationResponse!
        .FlightDirectionRecommendation!.DepartureFlightRecommendation ??= flightRecommendation;
    }
}

private static void StoreInMemoryReturnFlightRecommendation(object recommendationPopulationLock,
    FlightRecommendationBModel flightRecommendation)
{
    lock (recommendationPopulationLock)
    {
        InitializeFlightRecommendationResponseIfNull();

        (MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] as PreferencesResponse)!
        .FlightRecommendationResponse!
        .FlightDirectionRecommendation!.ReturnFlightRecommendation ??= flightRecommendation;
    }
}

private static void InitializeFlightRecommendationResponseIfNull()
{
    (MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] as PreferencesResponse)!
    .FlightRecommendationResponse ??= new FlightRecommendationResponse();

    (MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] as PreferencesResponse)!
    .FlightRecommendationResponse!.FlightDirectionRecommendation ??= new FlightDirectionRecommendationBModel();
}

public static bool FindOptimalDepartureFlightAndStoreInMemory(Guid sourceAgentId,
    string initialAssignedAgentName, object recommendationPopulationLock, TourismAgent tourismAgent,
    PreferencesRequest preferencesRequest)
{
    var departureCityFlights = FindFlightsByDepartureByDestination(tourismAgent,
        preferencesRequest.DepartureCityNavigation.Name, preferencesRequest.DestinationCityNavigation.Name);

    if (departureCityFlights.Any())
    {
        if (preferencesRequest.CustomerFlightNavigation?.DepartureNavigation == null)
        {
            var randomOptimalDepartureFlight =
                departureCityFlights.ElementAt(Random.Next(departureCityFlights.Count));

            var departureWeekDaysOfRandomFlight = randomOptimalDepartureFlight.WeekDaysOfFlight.DaysList
                .ConvertStringDaysOfWeekListToEnumList().ToList();
            var earliestDepartureDateOfRandomFlight =
                (DateTime) GetTheEarliestDepartureDateOffFlight(departureWeekDaysOfRandomFlight,
                    preferencesRequest.DepartureDate!);
            preferencesRequest.DepartureDate = earliestDepartureDateOfRandomFlight;

            var mostOptimalFlightTime = randomOptimalDepartureFlight.AvailableDepartureTime.DepartureHour
                .ConvertStringTimeListToTimeOnly().First();
            var randomDepartureFlightRecommendation = CreateFlightRecommendationBModel(sourceAgentId,
                initialAssignedAgentName,
                randomOptimalDepartureFlight, (short) ClassTypeId.Economy,
                earliestDepartureDateOfRandomFlight, mostOptimalFlightTime);

            StoreInMemoryDepartureFlightRecommendation(recommendationPopulationLock,
                randomDepartureFlightRecommendation);
            return true;
        }

        var optimalDepartureFlight =
            FindOptimalFlightByDirectionByUserPreference(preferencesRequest, departureCityFlights,
                "Departure");

        preferencesRequest.CustomerFlightNavigation.DepartureNavigation.Class ??= new FlightClassDto
        {
            Class = (short) ClassTypeId.Economy
        };
    }
}

```

```

var departureWeekFlight = optimalDepartureFlight.WeekDaysOfFlight.DaysList
    .ConvertStringDaysOfWeekListToEnumList().ToList();
var earliestDepartureDateOfFlight =
    (DateTime) GetTheEarliestDepartureDateOfFlight(departureWeekFlight, preferencesRequest.DepartureDate!);
preferencesRequest.DepartureDate = earliestDepartureDateOfFlight;

var earliestFlightTime = GetEarliestFlightTimeBasedOnPeriodPreference(optimalDepartureFlight,
    preferencesRequest.CustomerFlightNavigation.DepartureNavigation.DeparturePeriodPreference);

var departureFlightRecommendation = CreateFlightRecommendationBModel(sourceAgentId,
    initialAssignedAgentName,
    optimalDepartureFlight, preferencesRequest.CustomerFlightNavigation.DepartureNavigation.Class.Class,
    earliestDepartureDateOfFlight, earliestFlightTime);

StoreInMemoryDepartureFlightRecommendation(recommendationPopulationLock, departureFlightRecommendation);
return true;
}

return false;
}

public static bool FindOptimalReturnFlightAndStoreInMemory(Guid sourceAgentId,
    string initialAssignedAgentName, object recommendationPopulationLock, TourismAgent tourismAgent,
    PreferencesRequest preferencesRequest)
{
    var returnCityFlights = FindFlightsByDepartureByDestination(tourismAgent,
        preferencesRequest.DestinationCityNavigation.Name, preferencesRequest.DepartureCityNavigation.Name);

    if (returnCityFlights.Any())
    {
        if (preferencesRequest.CustomerFlightNavigation?.ReturnNavigation == null)
        {
            var randomOptimalReturnFlight = returnCityFlights.ElementAt(Random.Next(returnCityFlights.Count));

            var preferredReturnFlightDate =
                preferencesRequest.DepartureDate.AddDays(preferencesRequest.HolidaysPeriod);

            var returnWeekDaysOfRandomFlight = randomOptimalReturnFlight.WeekDaysOfFlight.DaysList
                .ConvertStringDaysOfWeekListToEnumList().ToList();
            var earliestReturnDateOfRandomFlight = (DateTime) GetTheMostOptimalReturnDateOfFlight(
                returnWeekDaysOfRandomFlight, preferencesRequest.DepartureDate, preferredReturnFlightDate,
                preferencesRequest.HolidaysPeriod!);

            var mostOptimalFlightTime = randomOptimalReturnFlight.AvailableDepartureTime.DepartureHour
                .ConvertStringTimeListToTimeOnly().First();
            var randomReturnFlightRecommendation = CreateFlightRecommendationBModel(sourceAgentId,
                initialAssignedAgentName,
                randomOptimalReturnFlight, (short) ClassTypeId.Economy,
                earliestReturnDateOfRandomFlight, mostOptimalFlightTime);

            StoreInMemoryDepartureFlightRecommendation(recommendationPopulationLock,
                randomReturnFlightRecommendation);
            return true;
        }

        var optimalArrivalFlight =
            FindOptimalFlightByDirectionByUserPreference(preferencesRequest, returnCityFlights,
                "Return");

        preferencesRequest.CustomerFlightNavigation.ReturnNavigation.Class ??= new FlightClassDto
        {
            Class = (short) ClassTypeId.Economy
        };

        var earliestFlightTime = GetEarliestFlightTimeBasedOnPeriodPreference(optimalArrivalFlight,
            preferencesRequest.CustomerFlightNavigation.ReturnNavigation.DeparturePeriodPreference);
        var flightReturnDate = GetFlightReturnDate(preferencesRequest);
    }
}

```

```

    var returnWeekDaysOfFlight = optimalArrivalFlight.WeekDaysOfFlight.DaysList
        .ConvertStringDaysOfWeekListToEnumList().ToList();
    var earliestReturnDateOfFlight = (DateTime) GetTheMostOptimalReturnDateOfFlight(returnWeekDaysOfFlight,
        preferencesRequest.DepartureDate, flightReturnDate, preferencesRequest.HolidaysPeriod!);

    var returnFlightRecommendation = CreateFlightRecommendationBModel(sourceAgentId, initialAssignedAgentName,
        optimalArrivalFlight, preferencesRequest.CustomerFlightNavigation.ReturnNavigation.Class.Class,
        earliestReturnDateOfFlight, earliestFlightTime);

    StoreInMemoryReturnFlightRecommendation(recommendationPopulationLock, returnFlightRecommendation);
    return true;
}

return false;
}

private static DateTime? GetTheMostOptimalReturnDateOfFlight(List<DayOfWeek> availableReturnDaysOfWeekFlight,
    DateTime actualDepartureFlightDate, DateTime preferredReturnFlightDate, int vacationPeriod)
{
    if (DateTime.Compare(actualDepartureFlightDate, preferredReturnFlightDate) >= 0)
        preferredReturnFlightDate = actualDepartureFlightDate.AddDays(vacationPeriod);

    var returnPreferredFlightDayOfWeek = preferredReturnFlightDate.DayOfWeek;

    if (!availableReturnDaysOfWeekFlight.Contains(returnPreferredFlightDayOfWeek))
    {
        if (returnPreferredFlightDayOfWeek != DayOfWeek.Sunday)
        {
            for (var dayOfWeekDescInc = returnPreferredFlightDayOfWeek;
                dayOfWeekDescInc >= DayOfWeek.Sunday;
                --dayOfWeekDescInc)
            {
                if (availableReturnDaysOfWeekFlight.Contains(dayOfWeekDescInc))
                {
                    var daysEarlierPreferredReturnFlight = returnPreferredFlightDayOfWeek - dayOfWeekDescInc;
                    var mostEarlyCloseToPreferredReturnFlight =
                        preferredReturnFlightDate.AddDays(-daysEarlierPreferredReturnFlight);

                    if (DateTime.Compare(actualDepartureFlightDate, mostEarlyCloseToPreferredReturnFlight) < 0)
                        return mostEarlyCloseToPreferredReturnFlight;
                }
            }

            for (var weekEarlierInc = DayOfWeek.Saturday;
                weekEarlierInc > returnPreferredFlightDayOfWeek;
                weekEarlierInc--)
            {
                if (availableReturnDaysOfWeekFlight.Contains(weekEarlierInc))
                {
                    var daysEarlierPreferredReturnFlight = (short) returnPreferredFlightDayOfWeek -
                        (returnPreferredFlightDayOfWeek - DayOfWeek.Sunday) + (DayOfWeek.Saturday - weekEarlierInc);
                    var mostEarlyCloseToPreferredReturnFlight =
                        preferredReturnFlightDate.AddDays(-daysEarlierPreferredReturnFlight);

                    if (DateTime.Compare(actualDepartureFlightDate, mostEarlyCloseToPreferredReturnFlight) < 0)
                        return mostEarlyCloseToPreferredReturnFlight;
                }
            }

            //Above is the search in early days than preferred one.
            //Now it's time to search in later days. It's the same
            //algorithm as for departure flight day search
            var earliestButLaterThanPreferredFlightDay =
                GetTheEarliestDepartureDateOfFlight(availableReturnDaysOfWeekFlight, preferredReturnFlightDate);
            return earliestButLaterThanPreferredFlightDay;
        }
        else
        {
            returnPreferredFlightDayOfWeek = DayOfWeek.Saturday;
            for (var dayOfWeekDescInc = returnPreferredFlightDayOfWeek;
                dayOfWeekDescInc > DayOfWeek.Sunday;
                --dayOfWeekDescInc)
            {
                if (availableReturnDaysOfWeekFlight.Contains(dayOfWeekDescInc))
                {

```

```

        var daysEarlierPreferredReturnFlight =
            returnPreferredFlightDayOfWeek - dayOfWeekDescInc +
            1; // +1 because we begun from Saturday, not Sunday
        var mostEarlyCloseToPreferredReturnFlight =
            preferredReturnFlightDate.AddDays(-daysEarlierPreferredReturnFlight);

        if (DateTime.Compare(actualDepartureFlightDate, mostEarlyCloseToPreferredReturnFlight) < 0)
            return mostEarlyCloseToPreferredReturnFlight;
    }

    var earliestButLaterThanPreferredFlightDay =
        GetTheEarliestDepartureDateOfFlight(availableReturnDaysOfWeekFlight, preferredReturnFlightDate);
    return earliestButLaterThanPreferredFlightDay;
}

return preferredReturnFlightDate;
}

private static DateTime? GetTheEarliestDepartureDateOfFlight(List<DayOfWeek> daysOfWeekFlight,
    DateTime preferredDepartureFlightDate)
{
    var preferredFlightDayOfWeek = preferredDepartureFlightDate.DayOfWeek;

    if (!daysOfWeekFlight.Contains(preferredFlightDayOfWeek))
    {
        if (preferredFlightDayOfWeek != DayOfWeek.Saturday)
        {
            for (var dayOfWeekAscInc = preferredFlightDayOfWeek + 1;
                dayOfWeekAscInc <= DayOfWeek.Saturday;
                dayOfWeekAscInc++)
            {
                if (daysOfWeekFlight.Contains(dayOfWeekAscInc))
                {
                    //Select day of current week and exit
                    var dayToTheEarliestFlight = dayOfWeekAscInc - preferredFlightDayOfWeek;
                    return preferredDepartureFlightDate.AddDays(dayToTheEarliestFlight);
                }
            }

            for (var nextWeekInc = DayOfWeek.Sunday; nextWeekInc < preferredFlightDayOfWeek; nextWeekInc++)
            {
                if (daysOfWeekFlight.Contains(nextWeekInc))
                {
                    var dayToTheEarliestFlight =
                        (short) (DayOfWeek.Saturday - preferredFlightDayOfWeek) + (short) nextWeekInc +
                        1; // +1 because the counter is from 0
                    return preferredDepartureFlightDate.AddDays(dayToTheEarliestFlight);
                }
            }

            for (var nextWeekInc = DayOfWeek.Sunday; nextWeekInc < preferredFlightDayOfWeek; nextWeekInc++)
            {
                if (daysOfWeekFlight.Contains(nextWeekInc))
                {
                    var dayToTheEarliestFlight =
                        (short) (DayOfWeek.Saturday - preferredFlightDayOfWeek) + (short) nextWeekInc + 1;
                    return preferredDepartureFlightDate.AddDays(dayToTheEarliestFlight);
                }
            }
        }
        else
        {
            return preferredDepartureFlightDate;
        }
    }

    return null;
}

private static bool IsFulfilledEarlyMorningPreference(
    this DeparturePeriodsPreferenceDto? departurePeriodsPreference,
    IReadOnlyDictionary<DayPeriods, List<TimeOnly>> groupedFlightTimes)
{

```

```

    return departurePeriodsPreference!.EarlyMorning && groupedFlightTimes[DayPeriods.EarlyMorning].Any();
}

private static bool IsFulfilledMorningPreference(this DeparturePeriodsPreferenceDto? departurePeriodsPreference,
    IReadOnlyDictionary<DayPeriods, List<TimeOnly>> groupedFlightTimes)
{
    return departurePeriodsPreference!.Morning && groupedFlightTimes[DayPeriods.Morning].Any();
}

private static bool IsFulfilledAfternoonPreference(this DeparturePeriodsPreferenceDto? departurePeriodsPreference,
    IReadOnlyDictionary<DayPeriods, List<TimeOnly>> groupedFlightTimes)
{
    return departurePeriodsPreference!.Afternoon && groupedFlightTimes[DayPeriods.Afternoon].Any();
}

private static bool IsFulfilledNightPreference(this DeparturePeriodsPreferenceDto? departurePeriodsPreference,
    IReadOnlyDictionary<DayPeriods, List<TimeOnly>> groupedFlightTimes)
{
    return departurePeriodsPreference!.Night && groupedFlightTimes[DayPeriods.Night].Any();
}

private static bool CheckPreferenceDeparturePeriodMatch(DeparturePeriodsPreferenceDto? departurePeriodsPreference,
    IReadOnlyDictionary<DayPeriods, List<TimeOnly>> groupedFlightTimes)
{
    if (departurePeriodsPreference == null)
        return false;

    return departurePeriodsPreference.IsFulfilledEarlyMorningPreference(groupedFlightTimes) ||
        departurePeriodsPreference.IsFulfilledMorningPreference(groupedFlightTimes) ||
        departurePeriodsPreference.IsFulfilledAfternoonPreference(groupedFlightTimes) ||
        departurePeriodsPreference.IsFulfilledNightPreference(groupedFlightTimes);
}

private static FlightRecommendationBModel CreateFlightRecommendationBModel(Guid sourceAgentId,
    string initialAssignedAgentName, FlightBusinessModel optimalFlight, short flightClass,
    DateTime departureDate, TimeOnly mostOptimalFlightTime)
{
    var optimalTime = mostOptimalFlightTime.ToTimeSpan();
    var sas =
        DateTime.MinValue.AddHours(optimalTime.Hours).AddMinutes(optimalTime.Minutes);
    return new FlightRecommendationBModel
    {
        SourceAgentId = sourceAgentId,
        InitialAssignedAgentName = initialAssignedAgentName,
        DepartureTime =
            DateTime.MinValue.AddHours(optimalTime.Hours).AddMinutes(optimalTime.Minutes),
        FlightClass = flightClass,
        FlightConnection = new List<FlightConnectionBModel>
        {
            new()
            {
                Flight = optimalFlight
            }
        },
        FlightDate = departureDate,
        Status = "Up-to-date",
        Stops = 0
    };
}

private static HashSet<FlightBusinessModel> FindFlightsByDepartureByDestination(AgentLocalDb agentLocalDb,
    string startFlightCityName, string endFlightCityName)
{
    return agentLocalDb.FlightsList.Where(f =>
        f.DepartureAirport.City.Name == startFlightCityName &&
        f.ArrivalAirport.City.Name == endFlightCityName).ToHashSet();
}

private static DateTime GetFlightReturnDate(PreferencesRequest preferencesRequest)
{

```

```

    }
    return preferencesRequest.DepartureDate.AddDays(preferencesRequest.HolidaysPeriod);
}
}

```

PropertyRecommendationLogic.cs

```

using VacationPackageWebApi.Domain.AgentsEnvironment.AgentModels;
using VacationPackageWebApi.Domain.Enums;
using VacationPackageWebApi.Domain.Mas.Singleton;
using VacationPackageWebApi.Domain.PreferencesPackageRequest;
using VacationPackageWebApi.Domain.PreferencesPackageResponse;
using VacationPackageWebApi.Domain.PreferencesPackageResponse.PropertyPreferencesResponse;
using VacationPackageWebApi.Domain.Property;
using static VacationPackageWebApi.Domain.Mas.BusinessLogic.CommonRecommendationLogic;

namespace VacationPackageWebApi.Domain.Mas.BusinessLogic;

public static class PropertyRecommendationLogic
{
    private static readonly Random Random = new();

    public static bool FindOptimalPropertyAndStoreInMemory(Guid sourceAgentId,
        string initialAssignedAgentName, object recommendationPopulationLock, TourismAgent tourismAgent,
        PreferencesRequest preferencesRequest)
    {
        var cityProperties = FindPropertiesByCity(tourismAgent, preferencesRequest.DestinationCityNavigation.Name);

        if (cityProperties.Any())
        {
            var optimalProperty =
                FindOptimalPropertyBasedOnUserPreferences(preferencesRequest, cityProperties);

            var propertyRecommendation =
                CreatePropertyRecommendationBModel(sourceAgentId, initialAssignedAgentName, optimalProperty);

            StoreInMemoryPropertyRecommendation(recommendationPopulationLock, propertyRecommendation);
            return true;
        }

        return false;
    }

    private static void StoreInMemoryPropertyRecommendation(object recommendationPopulationLock,
        PropertyRecommendationBModel propertyRecommendation)
    {
        var propertyPreferencesResponse = PopulatePropertyRecommendationResponse(propertyRecommendation);

        lock (recommendationPopulationLock)
        {
            (MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] as PreferencesResponse)!
                .PropertyPreferencesResponse ??= propertyPreferencesResponse;
        }
    }

    private static PropertyRecommendationResponse PopulatePropertyRecommendationResponse(
        PropertyRecommendationBModel propertyRecommendation)
    {
        return new PropertyRecommendationResponse
        {
            PropertyRecommendationBModel = propertyRecommendation
        };
    }

    private static PropertyRecommendationBModel CreatePropertyRecommendationBModel(Guid sourceAgentId,
        string initialAssignedAgentName, PropertyBusinessModel optimalProperty)
    {
        return new PropertyRecommendationBModel
        {
            SourceAgentId = sourceAgentId,

```

```
        InitialAssignedAgentName = initialAssignedAgentName,
        Property = optimalProperty,
        Status = "Up-to-date"
    };
}

private static PropertyBusinessModel FindOptimalPropertyBasedOnUserPreferences(
    PreferencesRequest preferencesRequest, HashSet<PropertyBusinessModel> properties)
{
    var bestSimilarityRate = 0.0d;
    var randomPropertyIndex = Random.Next(properties.Count);
    var optimalProperty = properties.ElementAt(randomPropertyIndex);

    if (preferencesRequest.CustomerPropertyNavigation == null) return optimalProperty;

    foreach (var property in properties)
    {
        var currentSimilarityRate = CalculatePropertySimilarityRate(preferencesRequest, property);
        if (currentSimilarityRate > bestSimilarityRate)
        {
            optimalProperty = property;
            bestSimilarityRate = currentSimilarityRate;
        }
    }

    return optimalProperty;
}

private static double CalculatePropertySimilarityRate(PreferencesRequest preferencesRequest,
    PropertyBusinessModel property)
{
    var propertyTypePreferenceRate =
        CheckPropertyTypePreferenceMatch(preferencesRequest.CustomerPropertyNavigation!.PropertyTypeNavigation,
            property);
    var placeTypePreferenceRate =
        CheckPlaceTypePreferenceMatch(preferencesRequest.CustomerPropertyNavigation.PlaceTypeNavigation, property);

    var roomsAndBedsSimilarityRate =
        RoomsAndBedsSimilarityRate(preferencesRequest.CustomerPropertyNavigation.RoomsAndBedsNavigation, property);
    var amenitiesPreferenceRate =
        AmenitiesSimilarityRate(preferencesRequest.CustomerPropertyNavigation.AmenitiesNavigation, property);
    var petsPreferenceRate = preferencesRequest.CustomerPropertyNavigation.Pets == property.Pet ? 1 : 0;
    var combinedPreferencesRate =
        CalculateSimilarityRateBasedOnPreferenceRate(propertyTypePreferenceRate, placeTypePreferenceRate);
    return (combinedPreferencesRate + roomsAndBedsSimilarityRate + amenitiesPreferenceRate + petsPreferenceRate) /
        4;
}

private static double CalculateSimilarityRateBasedOnPreferenceRate(bool propertyTypePreferenceRate,
    bool placeTypePreferenceRate)
{
    return ((propertyTypePreferenceRate
        ? Match
        : NoMatch) +
        (double) (placeTypePreferenceRate
        ? Match
        : NoMatch)) / 2;
}

private static bool CheckPropertyTypePreferenceMatch(PropertyTypePreferenceDto? propertyTypePreference,
    PropertyBusinessModel property)
{
    if (propertyTypePreference == null) return false;

    return propertyTypePreference.Apartment == (property.PropertyType.Id == (short) PropertyTypeId.Apartment) ||
        propertyTypePreference.Hotel == (property.PropertyType.Id == (short) PropertyTypeId.Hotel) ||
        propertyTypePreference.House == (property.PropertyType.Id == (short) PropertyTypeId.House) ||
        propertyTypePreference.GuestHouse == (property.PropertyType.Id == (short) PropertyTypeId.GuestHouse);
}
```

```

private static bool CheckPlaceTypePreferenceMatch(PlaceTypePreferenceDto? placeTypePreference,
PropertyBusinessModel property)
{
    if (placeTypePreference == null) return false;

    return placeTypePreference.EntirePlace == (property.PlaceType.Id == (short) PlaceTypeId.EntirePlace) ||
        placeTypePreference.PrivateRoom == (property.PlaceType.Id == (short) PlaceTypeId.PrivateRoom) ||
        placeTypePreference.SharedRoom == (property.PlaceType.Id == (short) PlaceTypeId.SharedRoom);
}

private static double RoomsAndBedsSimilarityRate(RoomsAndBedsPreferenceDto? roomsAndBedsPreference,
PropertyBusinessModel property)
{
    if (roomsAndBedsPreference == null) return 0.0d;

    var bathroomPreferenceMatch = roomsAndBedsPreference.Bathrooms == property.RoomAndBed.Bathroom
        ? Match
        : NoMatch;

    var bedsPreferenceMatch = roomsAndBedsPreference.Beds == property.RoomAndBed.Bed
        ? Match
        : NoMatch;

    var bathRoomsPreferenceMatch = roomsAndBedsPreference.Bathrooms == property.RoomAndBed.Bathroom
        ? Match
        : NoMatch;

    return (double) (bathroomPreferenceMatch + bedsPreferenceMatch + bathRoomsPreferenceMatch) / 3;
}

private static double AmenitiesSimilarityRate(AmenitiesPreferenceDto? amenitiesPreference,
PropertyBusinessModel property)
{
    if (amenitiesPreference == null) return 0.0d;

    var wiFiPreferenceMatch = amenitiesPreference.WiFi == property.AmenitiesPackage.WiFi
        ? Match
        : NoMatch;

    var kitchenPreferenceMatch = amenitiesPreference.Kitchen == property.AmenitiesPackage.Kitchen
        ? Match
        : NoMatch;

    var washerPreferenceMatch = amenitiesPreference.Washer == property.AmenitiesPackage.Washer
        ? Match
        : NoMatch;

    var dryerPreferenceMatch = amenitiesPreference.Dryer == property.AmenitiesPackage.Dryer
        ? Match
        : NoMatch;

    var airConditioningPreferenceMatch =
        amenitiesPreference.AirConditioning == property.AmenitiesPackage.AirConditioning
        ? Match
        : NoMatch;

    var heatingPreferenceMatch = amenitiesPreference.Heating == property.AmenitiesPackage.Heating
        ? Match
        : NoMatch;

    var tvPreferenceMatch = amenitiesPreference.Tv == property.AmenitiesPackage.Tv
        ? Match
        : NoMatch;

    var ironPreferenceMatch = amenitiesPreference.Iron == property.AmenitiesPackage.Iron
        ? Match
        : NoMatch;

    return (double) (wiFiPreferenceMatch + kitchenPreferenceMatch + washerPreferenceMatch + dryerPreferenceMatch +

```



```
        airConditioningPreferenceMatch + heatingPreferenceMatch + tvPreferenceMatch +  
        ironPreferenceMatch) / 8;  
    }  
  
    private static HashSet<PropertyBusinessModel> FindPropertiesByCity(AgentLocalDb agentLocalDb, string city)  
    {  
        return agentLocalDb.StaysList.Where(p => p.City.Name == city).ToHashSet();  
    }  
}
```

MasEnvVarsInitializer.cs

```
using VacationPackageWebApi.Domain.AgentsEnvironment.AgentModels;  
using VacationPackageWebApi.Domain.Enums;  
using VacationPackageWebApi.Domain.Mas.Singleton;  
using VacationPackageWebApi.Domain.PreferencesPackageRequest;  
using VacationPackageWebApi.Domain.PreferencesPackageResponse;  
  
namespace VacationPackageWebApi.Domain.Mas.Initializer;  
  
public static class MasEnvVarsInitializer  
{  
    public static Task InitializeAll()  
    {  
        MasEnvironmentSingleton.Instance.Memory["AvailableAgents"] = new List<string>();  
        MasEnvironmentSingleton.Instance.Memory["AvailableTasks"] = new List<TaskType>  
        {  
            TaskType.Attractions,  
            TaskType.Flight,  
            TaskType.Property  
        };  
  
        MasEnvironmentSingleton.Instance.Memory["PreferencesPayload"] = new PreferencesRequest();  
        MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] = new PreferencesResponse();  
        MasEnvironmentSingleton.Instance.Memory["PreferencesResponseStatus"] = "undone";  
        MasEnvironmentSingleton.Instance.Memory["AgentsTrustRates"] =  
            new Dictionary<Guid, List<TrustAgentRateBusinessModel>>();  
        return Task.CompletedTask;  
    }  
  
    public static void ResetAll()  
    {  
        var allAgents = MasEnvironmentSingleton.Instance.AllAgents();  
        if (allAgents.Contains("Coordinator"))  
            allAgents.Remove("Coordinator");  
        MasEnvironmentSingleton.Instance.Memory["AvailableAgents"] = allAgents;  
        MasEnvironmentSingleton.Instance.Memory["AvailableTasks"] = new List<TaskType>  
        {  
            TaskType.Attractions,  
            TaskType.Flight,  
            TaskType.Property  
        };  
  
        MasEnvironmentSingleton.Instance.Memory["PreferencesPayload"] = new PreferencesRequest();  
        MasEnvironmentSingleton.Instance.Memory["PreferencesResponse"] = new PreferencesResponse();  
        MasEnvironmentSingleton.Instance.Memory["AgentsTrustRates"] =  
            new Dictionary<Guid, List<TrustAgentRateBusinessModel>>();  
    }  
  
    public static void ResetTourismAgent(ref TourismAgent tourismAgent)  
    {  
        tourismAgent.Status = true;  
        tourismAgent.CurrentTask = TaskType.Default;  
    }  
}
```

MasCoordinatorSingleton.cs

```
namespace VacationPackageWebApi.Domain.Mas.Singleton;
```

```

public sealed class MasCoordinatorSingleton
{
    private static readonly Lazy<MasCoordinatorAgent> _lazy = new(() => new MasCoordinatorAgent());

    private MasCoordinatorSingleton()
    {
    }

    public static MasCoordinatorAgent Instance => _lazy.Value;
}

```

MasEnvironmentSingleton.cs

```

using ActressMas;

namespace VacationPackageWebApi.Domain.Mas.Singleton;

public sealed class MasEnvironmentSingleton
{
    private static readonly Lazy<EnvironmentMas> _lazy = new(() => new EnvironmentMas());

    private MasEnvironmentSingleton()
    {
    }

    public static EnvironmentMas Instance => _lazy.Value;
}

```

MasCoordinatorAgent.cs

```

using ActressMas;
using VacationPackageWebApi.Domain.Enums;
using VacationPackageWebApi.Domain.Helpers;
using VacationPackageWebApi.Domain.Mas.BusinessLogic;

namespace VacationPackageWebApi.Domain.Mas;

public class MasCoordinatorAgent : Agent
{
    private const int TotalNumberOfServices = 4;
    private readonly List<CoordinatorTasksDone> _tasksDone = new();

    public override void Setup()
    {
    }

    public override void Act(Message message)
    {
        try
        {
            Console.WriteLine($"{message.Format()}");
            message.Parse(out var action, out string parameters);

            switch (action)
            {
                case "departure_flight_recommendation_done":
                    _tasksDone.Add(CoordinatorTasksDone.DepartureFlight);
                    break;
                case "return_flight_recommendation_done":
                    _tasksDone.Add(CoordinatorTasksDone.ReturnFlight);
                    break;
                case "property_recommendation_done":
                    _tasksDone.Add(CoordinatorTasksDone.Property);
                    break;
                case "attraction_recommendation_done":
                    _tasksDone.Add(CoordinatorTasksDone.Attraction);
                    break;
            }
        }
    }
}

```

```
    }

    if (_tasksDone.Count == TotalNumberOfServices)
    {
        _tasksDone.Clear();
        CommonRecommendationLogic.SetPreferencesResponseStatusDone();
        "_____".WriteDebug();
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
}

public override void ActDefault()
{
}
}
```

MasVacationAgent.cs

```
using ActressMas;
using VacationPackageWebApi.Domain.AgentsEnvironment.AgentModels;
using VacationPackageWebApi.Domain.Enums;
using VacationPackageWebApi.Domain.Helpers;
using VacationPackageWebApi.Domain.Mas.BusinessLogic;
using VacationPackageWebApi.Domain.Mas.Initializer;
using VacationPackageWebApi.Domain.PreferencesPackageRequest;

namespace VacationPackageWebApi.Domain.Mas;

public class MasVacationAgent : Agent
{
    private static readonly object TaskDistributionLock = new();
    private static readonly object RecommendationPopulationLock = new();
    private PreferencesRequest _preferencesRequest;

    public TourismAgent TourismAgent;

    public MasVacationAgent(TourismAgent tourismAgent)
    {
        TourismAgent = tourismAgent;
        _preferencesRequest = new PreferencesRequest();
    }

    public override void Setup()
    {
    }

    public override async void Act(Message message)
    {
        try
        {
            Console.WriteLine($"{message.Format()}");
            message.Parse(out var action, out string _);
            switch (action)
            {
                case "new_recommendation_request":
                    ("Agent " + Name + " got request").WriteDebug();
                    Console.WriteLine("Agent " + Name + " got request");
                    _preferencesRequest = CommonRecommendationLogic.GetPreferencesPayload();

                    TourismAgent.ConfInd =
                        CommonRecommendationLogic.GetCurrentAgentCustomizedExpertRate(TourismAgent.Id,
                            _preferencesRequest.CustomizedExpertAgentRates!);

                    var taskType =
```

```

CommonRecommendationLogic.AccessPreferencesAndChoseTask(TaskDistributionLock, TourismAgent);
("Agent " + Name + " got task type " + taskType).WriteDebug();
Console.WriteLine("Agent " + Name + " got task type " + taskType);
switch (taskType)
{
    case TaskType.Default:
        return;
    case TaskType.Flight:
    {
        TourismAgent.Status = false;
        TourismAgent.CurrentTask = TaskType.Flight;
        List<string>? availableAgents = null;

        FlightRecommendationLogic.FulfillFlightDefaultPreferencesWithCheapestOffer(
            ref _preferencesRequest);

        var optimalDepartureFlightSolutionStoredSuccess =
            FlightRecommendationLogic.FindOptimalDepartureFlightAndStoreInMemory(TourismAgent.Id,
                TourismAgent.Name, RecommendationPopulationLock, TourismAgent, _preferencesRequest);

        if (optimalDepartureFlightSolutionStoredSuccess == false)
        {
            TourismAgent.TrustGradeInOtherAgent =
                CommonRecommendationLogic.GetAgentTrustRateOfAgentWithId(TourismAgent.Id!)
                    .OrderByDescending(ta => ta.FlightTrust.PositiveEvaluation).ToList();

            availableAgents =
                await CommonRecommendationLogic.GetListOfAllAgentsExceptCurrentAndCoordinator(
                    TourismAgent);

            if (availableAgents.Contains(TourismAgent.Name))
                availableAgents.Remove(TourismAgent.Name);

            if (!availableAgents.Any()) return;

            foreach (var trustRateInAgent in TourismAgent.TrustGradeInOtherAgent)
            {
                if (availableAgents.Contains(trustRateInAgent.TrustedAgentName))
                {
                    (Name + " send to " + trustRateInAgent.TrustedAgentName + " " +
                        "departure_flight_recommendation_request").WriteDebug();
                    Send(trustRateInAgent.TrustedAgentName,
                        "departure_flight_recommendation_request");
                }
            }
        }
        else
        {
            (Name + " send to Coordinator " + "departure_flight_recommendation_done").WriteDebug();
            Send("Coordinator", "departure_flight_recommendation_done");
        }

        var optimalReturnFlightSolutionStoredSuccess =
            FlightRecommendationLogic.FindOptimalReturnFlightAndStoreInMemory(TourismAgent.Id,
                TourismAgent.Name, RecommendationPopulationLock, TourismAgent, _preferencesRequest);

        if (optimalReturnFlightSolutionStoredSuccess == false)
        {
            if (optimalDepartureFlightSolutionStoredSuccess)
            {
                availableAgents =
                    await CommonRecommendationLogic.GetListOfAllAgentsExceptCurrentAndCoordinator(
                        TourismAgent);

                if (availableAgents.Contains(TourismAgent.Name))
                    availableAgents.Remove(TourismAgent.Name);

                if (!availableAgents.Any()) return;
            }

            TourismAgent.TrustGradeInOtherAgent =

```

```
CommonRecommendationLogic.GetAgentTrustRateOfAgentWithId(TourismAgent.Id)!
    .OrderByDescending(ta => ta.FlightTrust.PositiveEvaluation).ToList();

foreach (var trustRateInAgent in TourismAgent.TrustGradeInOtherAgent)
    if (availableAgents!.Contains(trustRateInAgent.TrustedAgentName))
    {
        (Name + " send to " + trustRateInAgent.TrustedAgentName + " " +
            "return_flight_recommendation_request").WriteDebug();

        Send(trustRateInAgent.TrustedAgentName, "return_flight_recommendation_request");
    }
}
else
{
    (Name + " send to Coordinator " + "return_flight_recommendation_done").WriteDebug();

    Send("Coordinator", "return_flight_recommendation_done");
}
}
break;
case TaskType.Property:
{
    TourismAgent.CurrentTask = TaskType.Property;
    TourismAgent.Status = false;

    var optimalPropertySolutionStoredSuccess =
        PropertyRecommendationLogic.FindOptimalPropertyAndStoreInMemory(TourismAgent.Id,
            TourismAgent.Name, RecommendationPopulationLock, TourismAgent, _preferencesRequest);

    if (optimalPropertySolutionStoredSuccess == false)
    {
        TourismAgent.TrustGradeInOtherAgent =
            CommonRecommendationLogic.GetAgentTrustRateOfAgentWithId(TourismAgent.Id)!
                .OrderByDescending(ta => ta.FlightTrust.PositiveEvaluation).ToList();

        var availableAgents =
            await CommonRecommendationLogic.GetListOfAllAgentsExceptCurrentAndCoordinator(
                TourismAgent);

        if (availableAgents.Contains(TourismAgent.Name))
            availableAgents.Remove(TourismAgent.Name);

        if (!availableAgents.Any()) return;

        foreach (var trustRateInAgent in TourismAgent.TrustGradeInOtherAgent)
            if (availableAgents.Contains(trustRateInAgent.TrustedAgentName))
            {
                (Name + " send to " + trustRateInAgent.TrustedAgentName + " " +
                    "property_recommendation_request").WriteDebug();

                Send(trustRateInAgent.TrustedAgentName, "property_recommendation_request");
            }
    }
    else
    {
        (Name + " send to Coordinator " + "property_recommendation_done").WriteDebug();

        Send("Coordinator", "property_recommendation_done");
    }
}
break;
case TaskType.Attractions:
{
    TourismAgent.CurrentTask = TaskType.Attractions;
    TourismAgent.Status = false;

    var optimalAttractionSolutionStoredSuccess =
        AttractionsRecommendationLogic.FindOptimalAttractionAndStoreInMemory(TourismAgent.Id,
            TourismAgent.Name, RecommendationPopulationLock, TourismAgent, _preferencesRequest);
```

```

    if (optimalAttractionSolutionStoredSuccess == false)
    {
        TourismAgent.TrustGradeInOtherAgent =
            CommonRecommendationLogic.GetAgentTrustRateOfAgentWithId(TourismAgent.Id)!
                .OrderByDescending(ta => ta.FlightTrust.PositiveEvaluation).ToList();

        var availableAgents =
            await CommonRecommendationLogic.GetListOfAllAgentsExceptCurrentAndCoordinator(
                TourismAgent);

        if (availableAgents.Contains(TourismAgent.Name))
            availableAgents.Remove(TourismAgent.Name);

        if (!availableAgents.Any()) return;

        foreach (var trustRateInAgent in TourismAgent.TrustGradeInOtherAgent)
            if (availableAgents.Contains(trustRateInAgent.TrustedAgentName))
            {
                (Name + " send to Coordinator " + "attractions_recommendation_request")
                    .WriteDebug();

                Send(trustRateInAgent.TrustedAgentName, "attractions_recommendation_request");
            }
        else
        {
            (Name + " send to Coordinator " + "attraction_recommendation_done").WriteDebug();

            Send("Coordinator", "attraction_recommendation_done");
        }
    }
    break;
}

break;
case "departure_flight_recommendation_request":
{
    TourismAgent.CurrentTask = TaskType.Flight;
    TourismAgent.Status = false;

    if (CommonRecommendationLogic.IsDepartureFlightRecommendationDone()) return;

    CommonRecommendationLogic.RemoveAgentFromAvailableAgentsList(TourismAgent.Name);
    var departureFlightRecommendationSuccess =
        FlightRecommendationLogic.FindOptimalDepartureFlightAndStoreInMemory(TourismAgent.Id,
            message.Sender, RecommendationPopulationLock, TourismAgent, _preferencesRequest);
    if (departureFlightRecommendationSuccess)
    {
        (Name + " send to Coordinator " + "departure_flight_recommendation_done").WriteDebug();

        Send("Coordinator", "departure_flight_recommendation_done");
    }
}
break;
case "return_flight_recommendation_request":
{
    TourismAgent.CurrentTask = TaskType.Flight;
    TourismAgent.Status = false;

    if (CommonRecommendationLogic.IsReturnFlightRecommendationDone()) return;

    CommonRecommendationLogic.RemoveAgentFromAvailableAgentsList(TourismAgent.Name);
    var returnFlightRecommendationSuccess =
        FlightRecommendationLogic.FindOptimalReturnFlightAndStoreInMemory(TourismAgent.Id,
            message.Sender, RecommendationPopulationLock, TourismAgent, _preferencesRequest);
    if (returnFlightRecommendationSuccess)
    {
        (Name + " send to Coordinator " + "return_flight_recommendation_done").WriteDebug();
    }
}

```

```
        Send("Coordinator", "return_flight_recommendation_done");
    }
}
break;
case "property_recommendation_request":
{
    TourismAgent.CurrentTask = TaskType.Property;
    TourismAgent.Status = false;

    if (CommonRecommendationLogic.IsPropertyRecommendationDone()) return;

    CommonRecommendationLogic.RemoveAgentFromAvailableAgentsList(TourismAgent.Name);
    var optimalPropertySolutionStoredSuccess =
        PropertyRecommendationLogic.FindOptimalPropertyAndStoreInMemory(TourismAgent.Id,
            message.Sender, RecommendationPopulationLock, TourismAgent, _preferencesRequest);
    if (optimalPropertySolutionStoredSuccess)
    {
        (Name + " send to Coordinator " + "property_recommendation_done").WriteDebug();

        Send("Coordinator", "property_recommendation_done");
    }
}
break;
case "attractions_recommendation_request":
{
    TourismAgent.CurrentTask = TaskType.Attractions;
    TourismAgent.Status = false;

    if (CommonRecommendationLogic.IsAttractionsRecommendationDone()) return;

    CommonRecommendationLogic.RemoveAgentFromAvailableAgentsList(TourismAgent.Name);
    var optimalAttractionSolutionStoredSuccess =
        AttractionsRecommendationLogic.FindOptimalAttractionAndStoreInMemory(TourismAgent.Id,
            message.Sender, RecommendationPopulationLock, TourismAgent, _preferencesRequest);
    if (optimalAttractionSolutionStoredSuccess)
    {
        (Name + " send to Coordinator " + "attraction_recommendation_done").WriteDebug();

        Send("Coordinator", "attraction_recommendation_done");
    }
}
break;
}

MasEnvVarsInitializer.ResetTourismAgent(ref TourismAgent);
await CommonRecommendationLogic.InsertAgentNameToAvailableAgents(TourismAgent.Name);
}
catch (Exception ex)
{
    Console.WriteLine("FromMasVacationAgent " + ex.Message);
}
}

public override void ActDefault()
{
}
}
```

Annex 3: Web Application

AttractionsController.cs

```
using Microsoft.AspNetCore.Mvc;
using VacationPackageWepApp.UiDataStoring.Preference;
using VacationPackageWepApp.UiDataStoring.Singleton;

namespace VacationPackageWepApp.Controllers;

[Route("[controller]")]
```



```

public class AttractionsController : Controller
{
    [HttpPost("[action]/{attractionsPreferences}")]
    public void StoreAttractionsPreferences(string attractionsPreferences)
    {
        var attractionPref = new AttractionPreferenceDto();

        var attractionsPreferencesList = attractionsPreferences.Split(", ").ToList();

        attractionPref.Natural = attractionsPreferencesList[0].Equals("true");
        attractionPref.Cultural = attractionsPreferencesList[1].Equals("true");
        attractionPref.Historical = attractionsPreferencesList[2].Equals("true");
        attractionPref.Religion = attractionsPreferencesList[3].Equals("true");
        attractionPref.Architecture = attractionsPreferencesList[4].Equals("true");
        attractionPref.IndustrialFacilities = attractionsPreferencesList[5].Equals("true");
        attractionPref.Other = attractionsPreferencesList[6].Equals("true");

        PreferencesPayloadSingleton.Instance.CustomerAttractionNavigation = attractionPref;
    }
}

```

FlightController.cs

```

using System.Text.Json;
using HttpClients;
using Microsoft.AspNetCore.Mvc;
using VacationPackageWepApp.UiDataStoring.Enums;
using VacationPackageWepApp.UiDataStoring.Preference;
using VacationPackageWepApp.UiDataStoring.PreferencesPackageResponse;
using VacationPackageWepApp.UiDataStoring.Singleton;

namespace VacationPackageWepApp.Controllers;

[Route("[controller]")]
public class FlightController : Controller
{
    private bool disposed = false;
    private GenericRestfulCrudHttpClient<string, string> flightClient =
        new("http://localhost:7071/", "Flight");

    private GenericRestfulCrudHttpClient<PreferencesRequest, PreferencesResponse> preferencesPackageClient =
        new("http://localhost:7071/", "VacationPackage/RequestVacationRecommendation/");

    [HttpGet("[action]")]
    public IActionResult GetFlightDepartureCities()
    {
        flightClient.addressSuffix = "Flight/GetFlightDepartureCities";
        var cities = flightClient.GetManyAsync();
        cities.Wait();
        var response = cities.Result;
        return new JsonResult(response.ToList());
    }

    [HttpPost("[action]/{flightDepartureCity}")]
    public IActionResult GetFlightArrivalCities(string flightDepartureCity)
    {
        flightClient.addressSuffix = "Flight/GetFlightArrivalCities/" + flightDepartureCity;
        var cities = flightClient.GetManyAsync();
        cities.Wait();
        var response = cities.Result;
        return new JsonResult(response.ToList());
    }

    [HttpGet("[action]")]
    public async Task SendVacationPackagePreferencesToTheServer()
    {
        PreferencesPayloadSingleton.Instance.CustomerId = new Guid("141fcf23-a053-1d6e-b5df-c0cacbb84b21");

        var preferencesResponse = await
        preferencesPackageClient.PostAsync<PreferencesResponse>(PreferencesPayloadSingleton.Instance);
    }
}

```

```
ResetPreferencesPayload();
}

[HttpPost("{action}/{departureCity}")]
public void StoreSelectedDepartureCity(string departureCity)
{
    PreferencesPayloadSingleton.Instance.DepartureCityNavigation = new CityDto()
    {
        Name = departureCity
    };
}

[HttpPost("{action}/{destinationCity}")]
public void StoreSelectedDestinationCity(string destinationCity)
{
    PreferencesPayloadSingleton.Instance.DestinationCityNavigation = new CityDto()
    {
        Name = destinationCity
    };
}

[HttpPost("{action}/{personsByAge}")]
public void StorePersonsByAge(string personsByAge)
{
    var ageList = personsByAge.Split(" ").ToList();

    if (ageList.Count != 3)
    {
        ageList = personsByAge.Split(",").ToList();
    }
    PreferencesPayloadSingleton.Instance.PersonsByAgeNavigation = new AgeCategoryPreferenceDto();

    if (ageList[0] != string.Empty && ageList[0] != " ") PreferencesPayloadSingleton.Instance.PersonsByAgeNavigation.Adult =
short.Parse(ageList[0]);
    if (ageList[1] != string.Empty && ageList[1] != " ") PreferencesPayloadSingleton.Instance.PersonsByAgeNavigation.Children
= short.Parse(ageList[1]);
    if (ageList[2] != string.Empty && ageList[2] != " ") PreferencesPayloadSingleton.Instance.PersonsByAgeNavigation.Infant =
short.Parse(ageList[2]);
}

[HttpPost("{action}/{stopsType}")]
public void StoreReturnStops(string? stopsType)
{
    if (stopsType is null or "null")
        return;

    var type = stopsType switch
    {
        "Direct" => (short) StopsTypePreferenceId.Direct,
        "OneStop" => (short) StopsTypePreferenceId.OneStop,
        "TwoOrMoreStops" => (short) StopsTypePreferenceId.TwoOrMoreStops,
        _ => (short) StopsTypePreferenceId.Direct
    };

    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation ??= new FlightDirectionPreferenceDto();
    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation ??= new FlightPreferenceDto();
    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation.StopsNavigation =
        new StopsTypePreferenceDto()
        {
            Type = type
        };
}

[HttpPost("{action}/{classType}")]
public void StoreDepartureClass(string classType)
{
    if (classType is null or "null")
        return;
}
```

```

var type = classType switch
{
    "Economy" => (short) ClassTypeId.Economy,
    "Business" => (short) ClassTypeId.Business,
    "First" => (short) ClassTypeId.First,
    _ => (short) ClassTypeId.Economy
};

PreferencesPayloadSingleton.Instance.CustomerFlightNavigation ??= new FlightDirectionPreferenceDto();
PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.DepartureNavigation ??= new FlightPreferenceDto();
PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.DepartureNavigation.Class =
    new FlightClassDto()
    {
        Class = type
    };
}

[HttpPost("{action}/{classType}")]
public void StoreReturnClass(string? classType)
{
    if (classType is null or "null")
        return;

    var type = classType switch
    {
        "Economy" => (short) ClassTypeId.Economy,
        "Business" => (short) ClassTypeId.Business,
        "First" => (short) ClassTypeId.First,
        _ => (short) ClassTypeId.Economy
    };

    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation ??= new FlightDirectionPreferenceDto();
    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation ??= new FlightPreferenceDto();
    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation.Class =
        new FlightClassDto()
        {
            Class = type
        };
}

[HttpPost("{action}/{stopsType}")]
public void StoreDepartureStops(string? stopsType)
{
    if (stopsType is null or "null")
        return;

    var type = stopsType switch
    {
        "Direct" => (short) StopsTypePreferenceId.Direct,
        "OneStop" => (short) StopsTypePreferenceId.OneStop,
        "TwoOrMoreStops" => (short) StopsTypePreferenceId.TwoOrMoreStops,
        _ => (short) StopsTypePreferenceId.Direct
    };

    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation ??= new FlightDirectionPreferenceDto();
    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.DepartureNavigation ??= new FlightPreferenceDto();
    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.DepartureNavigation.StopsNavigation =
        new StopsTypePreferenceDto()
        {
            Type = type
        };
}

[HttpPost("{action}/{days}")]
public void StoreVacationPeriod(short days)
{
    PreferencesPayloadSingleton.Instance.HolidaysPeriod = days;
}

```

```

[HttpPost("{action}/{date}")] // date format yyyy-mm-dd
public void StoreSelectedDepartureDate(string? date)
{
    PreferencesPayloadSingleton.Instance.DepartureDate = Convert.ToDateTime(date);
}

[HttpGet("{action}")]
public void CopyOptionalDepartureFlightPreferencesToReturn()
{
    if (PreferencesPayloadSingleton.Instance.CustomerFlightNavigation?.DepartureNavigation?.StopsNavigation != null)
    {
        PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation ??= new FlightPreferenceDto();
        PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation.StopsNavigation =
            PreferencesPayloadSingleton.Instance.CustomerFlightNavigation?.DepartureNavigation?.StopsNavigation;
    }

    if (PreferencesPayloadSingleton.Instance.CustomerFlightNavigation?.DepartureNavigation?.Class != null)
    {
        PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation ??= new FlightPreferenceDto();
        PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation.Class =
            PreferencesPayloadSingleton.Instance.CustomerFlightNavigation?.DepartureNavigation?.Class;
    }

    if (PreferencesPayloadSingleton.Instance.CustomerFlightNavigation?.DepartureNavigation?.DeparturePeriodPreference !=
null)
    {
        PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation ??= new FlightPreferenceDto();
        PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation.DeparturePeriodPreference =
            PreferencesPayloadSingleton.Instance.CustomerFlightNavigation?.DepartureNavigation?.DeparturePeriodPreference;
    }

    if (PreferencesPayloadSingleton.Instance.CustomerFlightNavigation?.DepartureNavigation?.FlightCompaniesNavigationList !=
null)
    {
        PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation ??= new FlightPreferenceDto();
        PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation.FlightCompaniesNavigationList =
            PreferencesPayloadSingleton.Instance.CustomerFlightNavigation?.DepartureNavigation?.FlightCompaniesNavigationList;
    }
}

[HttpPost("{action}/{returnFlightCompanies}")]
public void StoreSelectedReturnFlightCompanies(string? returnFlightCompanies)
{
    var listOfReturnFlightCompanies = returnFlightCompanies!.Split(", ").ToList();
    listOfReturnFlightCompanies.RemoveAll(e => e.Equals(string.Empty));
    if(listOfReturnFlightCompanies.Count == 0)
        return;

    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation ??= new FlightDirectionPreferenceDto();
    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation ??= new FlightPreferenceDto();
    var preferredDepartureFlightCompanies = listOfReturnFlightCompanies.Select(flightCompany => new
FlightCompaniesPreferenceDto() {Company = new FlightCompanyDto() {Name = flightCompany}}).ToList();

    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation
        .FlightCompaniesNavigationList = preferredDepartureFlightCompanies;
}

[HttpPost("{action}/{departureFlightCompanies}")]
public void StoreSelectedDepartureFlightCompanies(string? departureFlightCompanies)
{
    var listOfDepartureFlightCompanies = departureFlightCompanies!.Split(", ").ToList();
    listOfDepartureFlightCompanies.RemoveAll(e => e.Equals(string.Empty));
    if(listOfDepartureFlightCompanies.Count == 0)
        return;

    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation ??= new FlightDirectionPreferenceDto();
    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.DepartureNavigation ??= new FlightPreferenceDto();

```

```

    var preferredDepartureFlightCompanies = listOfDepartureFlightCompanies.Select(flightCompany => new
FlightCompaniesPreferenceDto() {Company = new FlightCompanyDto() {Name = flightCompany}}).ToList();

    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.DepartureNavigation
        .FlightCompaniesNavigationList = preferredDepartureFlightCompanies;
}

[HttpPost("{action}/{departurePeriods}")]
public void StoreSelectedDeparturePeriodsReturnFlight(string? departurePeriods)
{
    var listOfDeparturePeriods = departurePeriods!.Split(", ").ToList();
    listOfDeparturePeriods.RemoveAll(e => e.Equals(string.Empty));
    if(listOfDeparturePeriods.Count == 0)
        return;

    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation ??= new FlightDirectionPreferenceDto();

    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation ??= new FlightPreferenceDto();

    var departurePeriodPreference = new DeparturePeriodsPreferenceDto()
    {
        EarlyMorning = false,
        Afternoon = false,
        Morning = false,
        Night = false
    };

    foreach (var departurePeriod in listOfDeparturePeriods)
    {
        switch (departurePeriod)
        {
            case "Early Morning" : departurePeriodPreference.EarlyMorning = true;
                                break;
            case "Afternoon" : departurePeriodPreference.Afternoon = true;
                                break;
            case "Morning" : departurePeriodPreference.Morning = true;
                                break;
            case "Night" : departurePeriodPreference.Night = true;
                                break;
        }
    }

    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.ReturnNavigation.DeparturePeriodPreference =
        departurePeriodPreference;
}

[HttpPost("{action}/{departurePeriods}")]
public void StoreSelectedDeparturePeriodsDepartureFlight(string? departurePeriods)
{
    var listOfDeparturePeriods = departurePeriods!.Split(", ").ToList();
    listOfDeparturePeriods.RemoveAll(e => e.Equals(string.Empty));
    if(listOfDeparturePeriods.Count == 0)
        return;

    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation ??= new FlightDirectionPreferenceDto();

    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.DepartureNavigation ??= new FlightPreferenceDto();

    var departurePeriodPreference = new DeparturePeriodsPreferenceDto()
    {
        EarlyMorning = false,
        Afternoon = false,
        Morning = false,
        Night = false
    };

    foreach (var departurePeriod in listOfDeparturePeriods)
    {
        switch (departurePeriod)

```

```
        {
            case "Early Morning" : departurePeriodPreference.EarlyMorning = true;
                break;
            case "Afternoon" : departurePeriodPreference.Afternoon = true;
                break;
            case "Morning" : departurePeriodPreference.Morning = true;
                break;
            case "Night" : departurePeriodPreference.Night = true;
                break;
        }
    }

    PreferencesPayloadSingleton.Instance.CustomerFlightNavigation.DepartureNavigation.DeparturePeriodPreference =
        departurePeriodPreference;
}

[HttpGet("[action]")]
public void ResetPreferencesPayload()
{
    PreferencesPayloadSingleton.ResetInstance();
}
}
```

HomeController.cs

```
using System.Diagnostics;
using Microsoft.AspNetCore.Mvc;
using VacationPackageWepApp.Models;

namespace VacationPackageWepApp.Controllers;

public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;

    public HomeController(ILogger<HomeController> logger)
    {
        _logger = logger;
    }

    public IActionResult Index()
    {
        return View();
    }

    public IActionResult Privacy()
    {
        return View();
    }

    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
    public IActionResult Error()
    {
        return View(new ErrorViewModel {RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier});
    }
}
```

PropertyController.cs

```
using Microsoft.AspNetCore.Mvc;
using VacationPackageWepApp.UiDataStoring.Preference;
using VacationPackageWepApp.UiDataStoring.Singleton;

namespace VacationPackageWepApp.Controllers;

[Route("[controller]")]
public class PropertyController : Controller
{
    [HttpPost("[action]/[amenities]")]
```

```

public void StoreAmenities(string? amenities)
{
    var listOfAmenities = amenities!.Split(", ").ToList();
    listOfAmenities.RemoveAll(e => e.Equals(string.Empty));
    if (listOfAmenities.Count == 0)
    {
        if (PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation != null)
            PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation.AmenitiesNavigation = null;
        return;
    }

    PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation ??= new PropertyPreferenceDto();

    var amenitiesPreference = new AmenitiesPreferenceDto()
    {
        WiFi = false,
        AirConditioning = false,
        Dryer = false,
        Heating = false,
        Iron = false,
        Kitchen = false,
        Tv = false,
        Washer = false
    };

    foreach (var amenity in listOfAmenities)
    {
        switch (amenity)
        {
            case "WiFi":
                amenitiesPreference.WiFi = true;
                break;
            case "AirConditioning":
                amenitiesPreference.AirConditioning = true;
                break;
            case "Dryer":
                amenitiesPreference.Dryer = true;
                break;
            case "Heating":
                amenitiesPreference.Heating = true;
                break;
            case "Iron":
                amenitiesPreference.Iron = true;
                break;
            case "Kitchen":
                amenitiesPreference.Kitchen = true;
                break;
            case "Tv":
                amenitiesPreference.Tv = true;
                break;
            case "Washer":
                amenitiesPreference.Washer = true;
                break;
        }
    }

    PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation.AmenitiesNavigation = amenitiesPreference;
}

[HttpPost("{action}/{placesType}")]
public void StorePlaceType(string? placesType)
{
    var listOfPlacesType = placesType!.Split(", ").ToList();
    listOfPlacesType.RemoveAll(e => e.Equals(string.Empty));
    if (listOfPlacesType.Count == 0)
    {
        if (PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation != null)
            PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation.PlaceTypeNavigation = null;
        return;
    }
}

```



```
PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation ??= new PropertyPreferenceDto();

var propertyType = new PlaceTypePreferenceDto()
{
    EntirePlace = false,
    PrivateRoom = false,
    SharedRoom = false,
};

foreach (var propType in listOfPlacesType)
    switch (propType)
    {
        case "Entire Place":
            propertyType.EntirePlace = true;
            break;
        case "Private Room":
            propertyType.PrivateRoom = true;
            break;
        case "Shared Room":
            propertyType.SharedRoom = true;
            break;
    }

PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation.PlaceTypeNavigation = propertyType;
}

[HttpPost("{action}/{pets}")]
public void StorePetsPreference(string pets)
{
    PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation ??= new PropertyPreferenceDto();
    PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation.Pets = pets == "true";
}

[HttpPost("{action}/{roomsAndBeds}")]
public void StoreRoomsAndBedsPreference(string roomsAndBeds)
{
    var roomsAndBedsList = roomsAndBeds.Split(", ").ToList();

    if (roomsAndBedsList.Count != 3)
    {
        roomsAndBedsList = roomsAndBeds.Split("; ").ToList();
    }

    PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation ??= new PropertyPreferenceDto();
    PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation.RoomsAndBedsNavigation =
        new RoomsAndBedsPreferenceDto();

    if (roomsAndBedsList[0] != string.Empty && roomsAndBedsList[0] != "")
        PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation.RoomsAndBedsNavigation.Bathrooms =
            short.Parse(roomsAndBedsList[0]);
    if (roomsAndBedsList[1] != string.Empty && roomsAndBedsList[1] != "")
        PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation.RoomsAndBedsNavigation.Beds =
            short.Parse(roomsAndBedsList[1]);
    if (roomsAndBedsList[2] != string.Empty && roomsAndBedsList[2] != "")
        PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation.RoomsAndBedsNavigation.Bedrooms =
            short.Parse(roomsAndBedsList[2]);
}

[HttpPost("{action}/{propertiesType}")]
public void StorePropertyType(string? propertiesType)
{
    var listOfTypeProperties = propertiesType!.Split(", ").ToList();
    listOfTypeProperties.RemoveAll(e => e.Equals(string.Empty));
    if (listOfTypeProperties.Count == 0)
    {
        if (PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation != null)
            PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation.PropertyTypeNavigation = null;
        return;
    }
}
```

```
PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation ??= new PropertyPreferenceDto();
```

```
var propertyType = new PropertyTypePreferenceDto
{
    Apartment = false,
    GuestHouse = false,
    Hotel = false,
    House = false
};
```

```
foreach (var propType in listOfTypeProperties)
    switch (propType)
    {
        case "House":
            propertyType.House = true;
            break;
        case "Apartment":
            propertyType.Apartment = true;
            break;
        case "Guest House":
            propertyType.GuestHouse = true;
            break;
        case "Hotel":
            propertyType.Hotel = true;
            break;
    }
```

```
PreferencesPayloadSingleton.Instance.CustomerPropertyNavigation.PropertyTypeNavigation = propertyType;
```

```
}
}
```

_FlightSearch.cshtml

```
<script>
/* Store Selected Departure Date */
/* Store Selected Departure Day Periods Departure Flight */
/* Store Selected Departure Day Periods Return Flight */
/* Store Selected Departure Flight Companies */
/* Store Selected Return Flight Companies */
$(document).ready(function(){

    $('input[name="returnFlightCompanies"]').on('change', function() {
        var flightCompanies = [];
        $("input:checkbox[name='returnFlightCompanies']:checked").each(function(){
            flightCompanies.push($(this).attr("value"));
        });
        var selectedFlightCompanies = flightCompanies.join(", ");
        $.ajax({
            type: 'POST',
            url: '/Flight/StoreSelectedReturnFlightCompanies/' + selectedFlightCompanies,
            success: function (response) {
            },
            error: function (response) {
                // code on failure
            }
        });
    });

    $('input[name="departureFlightCompanies"]').on('change', function() {
        var flightCompanies = [];
        $("input:checkbox[name='departureFlightCompanies']:checked").each(function(){
            flightCompanies.push($(this).attr("value"));
        });
        var selectedFlightCompanies = flightCompanies.join(", ");
        $.ajax({
            type: 'POST',
            url: '/Flight/StoreSelectedDepartureFlightCompanies/' + selectedFlightCompanies,
            success: function (response) {
            }
        });
    });
});
```

```
        },
        error: function (response) {
            // code on failure
        }
    });
});

$('#check-in-date').change(function() {
    var date = $(this).val();

    $.ajax({
        type: 'POST',
        url: '/Flight/StoreSelectedDepartureDate/' + date,
        success: function (response) {

        },
        error: function (response) {
            // code on failure
        }
    });
});

//for change event
$( 'input[name="departurePeriodsValue"]' ).on( 'change', function() {
    var departurePeriods = [];
    $( "input:checkbox[name='departurePeriodsValue']:checked" ).each( function() {
        departurePeriods.push( $(this).attr("value") );
    });
    var selectedDeparturePeriods = departurePeriods.join(", ");
    $.ajax({
        type: 'POST',
        url: '/Flight/StoreSelectedDeparturePeriodsDepartureFlight/' + selectedDeparturePeriods,
        success: function (response) {

        },
        error: function (response) {
            // code on failure
        }
    });
});
});

$(document).ready(function(){
    //for change event
    $( 'input[name="departurePeriodsReturn Value"]' ).on( 'change', function() {
        var departurePeriods = [];
        $( "input:checkbox[name='departurePeriodsReturn Value']:checked" ).each( function() {
            departurePeriods.push( $(this).attr("value") );
        });
        var selectedDeparturePeriods = departurePeriods.join(", ");
        $.ajax({
            type: 'POST',
            url: '/Flight/StoreSelectedDeparturePeriodsReturnFlight/' + selectedDeparturePeriods,
            success: function (response) {

            },
            error: function (response) {
                // code on failure
            }
        });
    });
});
</script>
<script>
//Get from db all departure cities
$(document).ready(function(){
    var firstOption = "<option value='none' selected disabled hidden>SELECT CITY</option>";
    $('#departure').html(firstOption);
    $.ajax({
        type: 'GET',
        url: '/Flight/GetFlightDepartureCities',
```

```

    success: function (response) {
        var constructCities = "";
        for (let i = 0; i < response.length; i++)
        {
            constructCities += "<option>" + response[i] + "</option>";
        }

        $('#departure').html(firstOption + constructCities);
    },
    error: function (response) {
        // code on failure
    }
});
});
</script>

<script>

function getCharsBefore(str, chr) {
    let index = str.indexOf(chr);
    if (index !== -1) {
        return (str.substring(0, index));
    }
    return ("");
}

/* Store new selected destination */
$('#destination').change(function ()
{
    let selectedDestinationCityAndCountry = jQuery(this).val();
    let selectedDestinationCity = getCharsBefore(selectedDestinationCityAndCountry, ",");
    let storeDestinationUrl = '/Flight/StoreSelectedDestinationCity/' + selectedDestinationCity;
    $.ajax({
        type: 'POST',
        url: storeDestinationUrl,
        success: function (response) {
        },
        error: function (response) {
            // code on failure
        }
    });
});

/* Get from database possible destination based on departure city. */
/* Store selected departure city */
$('#departure').change(function()
{
    let departureFirstOption = "<option value=\"none\" selected disabled hidden>SELECT CITY</option>";
    let selectedDepartureCityAndCountry = jQuery(this).val();
    if(selectedDepartureCityAndCountry === "SELECT CITY")
        return;
    let selectedDepartureCity = getCharsBefore(selectedDepartureCityAndCountry, ",");
    let url = '/Flight/GetFlightArrivalCities/' + selectedDepartureCity;
    $.ajax({
        type: 'POST',
        url: url,
        success: function (response) {
            let constructCities = "";
            for (let i = 0; i < response.length; i++)
            {
                constructCities += "<option>" + response[i] + "</option>";
            }
            $('#destination').html(departureFirstOption + constructCities);
        },
        error: function (response) {
            // code on failure
        }
    });
});

```

```
let storeUrl = '/Flight/StoreSelectedDepartureCity/' + selectedDepartureCity;
$.ajax({
  type: 'POST',
  url: storeUrl,
  success: function (response) {
  },
  error: function (response) {
    // code on failure
  }
});

});
</script>
```

```
<script>
$(document).ready(function() {

  $.ajax({
    type: 'GET',
    url: '/Flight/ResetPreferencesPayload',
    success: function (response) {
    },
    error: function (response) {
      // code on failure
    }
  });

});
</script>
```

```
<script>
$(document).ready(function() {

  function getCharsAfter(str, chr) {
    return str.substr(str.lastIndexOf(chr) + 2);
  }

});
</script>
```

_Layout.cshtml

```
<script>
$(document).ready(function(){
  $('#SameForReturnFlight').change(function(){
    if(this.checked)
    {
      $('#returnFlightPreferencesUI').fadeOut(1000);
    }
    else
      $('#returnFlightPreferencesUI').fadeIn(1000);
  }

  });
</script>
<script>
let propertyTypeList = document.getElementById('propertyTypeUIList');
propertyTypeList.getElementsByClassName('anchor')[0].onclick = function(evt) {
  if (propertyTypeList.classList.contains('visible'))
    propertyTypeList.classList.remove('visible');
  else
    propertyTypeList.classList.add('visible');
}
let placeTypeList = document.getElementById('placeTypeUIList');
placeTypeList.getElementsByClassName('anchor')[0].onclick = function(evt) {
  if (placeTypeList.classList.contains('visible'))
    placeTypeList.classList.remove('visible');
  else
    placeTypeList.classList.add('visible');
```

```

}
let amenitiesList = document.getElementById('amenitiesUIList');
amenitiesList.getElementsByClassName('anchor')[0].onclick = function(evt) {
  if (amenitiesList.classList.contains('visible'))
    amenitiesList.classList.remove('visible');
  else
    amenitiesList.classList.add('visible');
}
</script>
<script>
let checkList = document.getElementById('departurePeriodsUIList');
checkList.getElementsByClassName('anchor')[0].onclick = function(evt) {
  if (checkList.classList.contains('visible'))
    checkList.classList.remove('visible');
  else
    checkList.classList.add('visible');
}
let checkListDeparturePeriodsReturnFlight = document.getElementById('departurePeriodsReturnUIList');
checkListDeparturePeriodsReturnFlight.getElementsByClassName('anchor')[0].onclick = function(evt) {
  if (checkListDeparturePeriodsReturnFlight.classList.contains('visible'))
    checkListDeparturePeriodsReturnFlight.classList.remove('visible');
  else
    checkListDeparturePeriodsReturnFlight.classList.add('visible');
}
</script>
<script>
let flightCompaniesUICheckList = document.getElementById('flightCompaniesUIList');
flightCompaniesUICheckList.getElementsByClassName('anchor')[0].onclick = function(evt) {
  if (flightCompaniesUICheckList.classList.contains('visible'))
    flightCompaniesUICheckList.classList.remove('visible');
  else
    flightCompaniesUICheckList.classList.add('visible');
}
let flightCompaniesReturnUICheckList = document.getElementById('flightCompaniesReturnUIList');
flightCompaniesReturnUICheckList.getElementsByClassName('anchor')[0].onclick = function(evt) {
  if (flightCompaniesReturnUICheckList.classList.contains('visible'))
    flightCompaniesReturnUICheckList.classList.remove('visible');
  else
    flightCompaniesReturnUICheckList.classList.add('visible');
}
</script>

```

PropertySearch.cshtml

```

<script>
$(☐[name="amenitiesValue"]).on('change', function() {
  var placeType = [];
  $(☐[name="amenitiesValue"]:checked).each(function(){
    placeType.push($(this).attr("value"));
  });
  var selectedAmenity = placeType.join(", ");
  $.ajax({
    type: 'POST',
    url: '/Property/StoreAmenities/' + selectedAmenity,
    success: function (response) {
    },
    error: function (response) {
      // code on failure
    }
  });
});

$(☐[name="propertyTypeValue"]).on('change', function() {
  var propertyType = [];
  $(☐[name="propertyTypeValue"]:checked).each(function(){
    propertyType.push($(this).attr("value"));
  });
  var selectedPropertyType = propertyType.join(", ");
  $.ajax({
    type: 'POST',

```

```
        url: '/Property/StorePropertyType/' + selectedPropertyType,
        success: function (response) {
            },
        error: function (response) {
            // code on failure
        }
    });
});

$('input[name="placeTypeValue"]').on('change', function() {
    var placeType = [];
    $('input:checkbox[name="placeTypeValue"]:checked').each(function(){
        placeType.push($(this).attr("value"));
    });
    var selectedPlaceType = placeType.join(", ");
    $.ajax({
        type: 'POST',
        url: '/Property/StorePlaceType/' + selectedPlaceType,
        success: function (response) {
            },
        error: function (response) {
            // code on failure
        }
    });
});
</script>
```

Annex 4: Database migrator

Migration202206261343Initial.cs

```
using FluentMigrator;

namespace DbMigrator.Migrations
{
    [Migration(202206261343)]
    public class Migration202206261343Initial : AutoReversingMigration
    {
        private const string CityTable = "City";
        private const string CountryTable = "Country";
        public override void Up()
        {
            Create.Table(CountryTable)
                .WithColumn("Id").AsGuid().PrimaryKey()
                .WithColumn("Name").AsString();

            Create.Table(CityTable)
                .WithColumn("Id").AsGuid().PrimaryKey()
                .WithColumn("Name").AsString()
                .WithColumn("CountryId").AsGuid().ForeignKey(CountryTable, "Id");
        }
    }
}
```

Migration202206261513AddCustomerAndPropertyTables.cs

```
using FluentMigrator;

namespace DbMigrator.Migrations
{
    [Migration(202206261513)]
    public class Migration202206261513AddCustomerAndPropertyTables : AutoReversingMigration
    {
        private const string CustomerTable = "Customer";
        private const string PropertyTable = "Property";
        private const string PropertyTypeTable = "PropertyType";
        private const string PlaceTypeTable = "PlaceType";
        private const string RoomAndBedTable = "RoomAndBed";
        private const string AmenitiesPackageTable = "AmenitiesPackage";
        private const string CityTable = "City";
    }
}
```



```

public override void Up()
{
    Create.Table(CustomerTable)
        .WithColumn("Id").AsGuid().PrimaryKey()
        .WithColumn("FirstName").AsString(50)
        .WithColumn("LastName").AsString(50)
        .WithColumn("Login").AsString(50)
        .WithColumn("Password").AsString(50)
        .WithColumn("Email").AsString(50);

    Create.Table(PropertyTypeTable) // auto increment?
        .WithColumn("Id").AsInt16().PrimaryKey().Identity()
        .WithColumn("Type").AsString(15);

    Create.Table(PlaceTypeTable) // auto increment?
        .WithColumn("Id").AsInt16().PrimaryKey().Identity()
        .WithColumn("Type").AsString(15);

    Create.Table(RoomAndBedTable)
        .WithColumn("Id").AsGuid().PrimaryKey()
        .WithColumn("Bedroom").AsInt16()
        .WithColumn("Bed").AsInt16()
        .WithColumn("Bathroom").AsInt16();

    Create.Table(AmenitiesPackageTable)
        .WithColumn("Id").AsGuid().PrimaryKey()
        .WithColumn("WiFi").AsBoolean()
        .WithColumn("Kitchen").AsBoolean()
        .WithColumn("Washer").AsBoolean()
        .WithColumn("Dryer").AsBoolean()
        .WithColumn("AirConditioning").AsBoolean()
        .WithColumn("Heating").AsBoolean()
        .WithColumn("Tv").AsBoolean()
        .WithColumn("Iron").AsBoolean();

    Create.Table(PropertyTable)
        .WithColumn("Id").AsGuid().PrimaryKey()
        .WithColumn("Name").AsString(200)
        .WithColumn("PropertyTypeId").AsInt16().ForeignKey(PropertyTypeTable, "Id")
        .WithColumn("PlaceTypeId").AsInt16().ForeignKey(PlaceTypeTable, "Id")
        .WithColumn("RoomAndBedId").AsGuid().ForeignKey(RoomAndBedTable, "Id")
        .WithColumn("Pet").AsBoolean()
        .WithColumn("AmenitiesPackageId").AsGuid().ForeignKey(AmenitiesPackageTable, "Id")
        .WithColumn("PricePerDay").AsInt16()
        .WithColumn("CityId").AsGuid().ForeignKey(CityTable, "Id");
}
}
}

```

Migration202206281121AddAttractionTables.cs

```

using FluentMigrator;

namespace DbMigrator.Migrations
{
    [Migration(202206281121)]
    public class Migration202206281121AddAttractionTables : AutoReversingMigration
    {
        private const string AttractionTable = "OpenTripMapAttraction";

        public override void Up()
        {
            Create.Table(AttractionTable)
                .WithColumn("Xid").AsString(20).PrimaryKey()
                .WithColumn("Name").AsString(100)
                .WithColumn("Town").AsString(30)
                .WithColumn("State").AsString(50)
                .WithColumn("County").AsString(50)

```

```
.WithColumn("Suburb").AsString(100)
.WithColumn("Country").AsString(30)
.WithColumn("Postcode").AsString(30)
.WithColumn("Pedestrian").AsString(50)
.WithColumn("CountryCode").AsString(20)
.WithColumn("Neighbourhood").AsString(50)
.WithColumn("Rate").AsString(10)
.WithColumn("Osm").AsString(30)
.WithColumn("LonMin").AsDouble()
.WithColumn("LonMax").AsDouble()
.WithColumn("LatMin").AsDouble()
.WithColumn("LatMax").AsDouble()
.WithColumn("Wikidata").AsString(30)
.WithColumn("Kinds").AsString(200)
.WithColumn("Geometry").AsString(15)
.WithColumn("Otm").AsString(50)
.WithColumn("Wikipedia").AsString(600)
.WithColumn("Image").AsString(700)
.WithColumn("Source").AsString(900)
.WithColumn("Height").AsInt16()
.WithColumn("Width").AsInt16()
.WithColumn("Title").AsString(100)
.WithColumn("Text").AsString(10000)
.WithColumn("Html").AsString(10000)
.WithColumn("Lon").AsDouble()
.WithColumn("Lat").AsDouble();
    }
}
}
```

[Migration202206281328AddFlightTables.cs](#)

```
using FluentMigrator;

namespace DbMigrator.Migrations
{
    [Migration(202206281328)]
    public class Migration202206281328AddFlightTables : AutoReversingMigration
    {
        private const string FlightTable = "Flight";
        private const string AirportTable = "Airport";
        private const string FlightPriceTable = "FlightPrice";
        private const string FlightClassTable = "FlightClass";
        private const string FlightCompanyTable = "FlightCompany";
        private const string WeekDaysOfFlightTable = "WeekDaysOfFlight";
        private const string AvailableDepartureTimeTable = "AvailableDepartureTime";
        private const string CityTable = "City";

        public override void Up()
        {
            Create.Table(AirportTable)
                .WithColumn("Id").AsGuid().PrimaryKey()
                .WithColumn("Name").AsString(60)
                .WithColumn("CityId").AsGuid().ForeignKey(CityTable, "Id");

            Create.Table(FlightCompanyTable)
                .WithColumn("Id").AsGuid().PrimaryKey()
                .WithColumn("Name").AsString(30);

            Create.Table(FlightClassTable)
                .WithColumn("Id").AsInt16().PrimaryKey().Identity()
                .WithColumn("Class").AsString(10);

            Create.Table(WeekDaysOfFlightTable)
                .WithColumn("Id").AsGuid().PrimaryKey()
                .WithColumn("DaysList").AsString(100);

            Create.Table(AvailableDepartureTimeTable)
                .WithColumn("Id").AsGuid().PrimaryKey()

```

```

        .WithColumn("DepartureHour").AsString(50);

Create.Table(FlightTable)
    .WithColumn("Id").AsGuid().PrimaryKey()
    .WithColumn("DepartureAirportId").AsGuid().ForeignKey(AirportTable, "Id")
    .WithColumn("ArrivalAirportId").AsGuid().ForeignKey(AirportTable, "Id")
    .WithColumn("Duration").AsInt16()
    .WithColumn("CompanyId").AsGuid().ForeignKey(FlightCompanyTable, "Id")
    .WithColumn("WeekDaysOfFlightId").AsGuid().ForeignKey(WeekDaysOfFlightTable, "Id")
    .WithColumn("AvailableDepartureTimeId").AsGuid().ForeignKey(AvailableDepartureTimeTable, "Id");

Create.Table(FlightPriceTable)
    .WithColumn("Id").AsGuid().PrimaryKey()
    .WithColumn("FlightId").AsGuid().ForeignKey(FlightTable, "Id")
    .WithColumn("ClassId").AsInt16().ForeignKey(FlightClassTable, "Id")
    .WithColumn("Price").AsInt16();
    }
}
}

```

Bootstrap.cs

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using System.IO;

namespace DbMigrator
{
    public static class Bootstrap
    {
        public static string AppSettingsPath { get; set; } = "appsettings.json";

        public static WebApplicationBuilder CreateAppBuilder(string[] args)
            => WebApplication.CreateBuilder(args);

        public static WebApplication BuildApp(WebApplicationBuilder builder)
        {
            var configuration = GetConfiguration();

            builder.WebHost
                .UseConfiguration(configuration)
                .UseUrls(configuration.GetValue<string>("Hostings:Urls"));

            ConfigureServices(builder.Services, configuration);

            var app = builder.Build();

            return app;
        }

        private static void ConfigureServices(IServiceCollection services, IConfiguration configuration)
        {
            AddDb(services, configuration);
        }

        private static void AddDb(IServiceCollection services, IConfiguration configuration)
        {
            var vacationPackageDatabaseOptions =
configuration.GetOptions<VacationPackageDatabaseOptions>(VacationPackageDatabaseOptions.ConfigKey);
            services.AddEntityFramework(vacationPackageDatabaseOptions);
        }

        private static IConfiguration GetConfiguration()
        {
            var builder = new ConfigurationBuilder()
                .SetBasePath(Directory.GetCurrentDirectory())
                .AddJsonFile(AppSettingsPath, false, true)

```

```
        .AddEnvironmentVariables();

        return builder.Build();
    }
}
```

DatabaseHandler.cs

```
using FluentMigrator.Runner;
using Npgsql;
using System.Threading.Tasks;

namespace DbMigrator
{
    public static class DatabaseHandler
    {
        public static async Task CreateAsync(string migrationConnectionString, string catalogName)
        {
            await using var connection = new NpgsqlConnection(migrationConnectionString);
            await connection.OpenAsync();

            if (await DatabaseDoesNotExistAsync(catalogName, connection))
            {
                await using var cmd = new NpgsqlCommand($"create database {catalogName}", connection);
                await cmd.ExecuteNonQueryAsync();
            }
        }

        public static void Update(IMigrationRunner runner, bool rollback = false)
        {
            runner.ListMigrations();

            if (rollback && runner.HasMigrationsToApplyRollback())
            {
                runner.Rollback(1);
                return;
            }

            runner.MigrateUp();
        }

        private static async Task<bool> DatabaseDoesNotExistAsync(string dbName, NpgsqlConnection connection)
        {
            await using var command = new NpgsqlCommand(
                $"select COUNT(1) from pg_catalog.pg_database where datname = '{dbName}';",
                connection);

            return (long)command.ExecuteScalar() == 0;
        }
    }
}
```

DatabaseMigrator.cs

```
using FluentMigrator.Runner;
using Microsoft.Extensions.DependencyInjection;
using System;
using System.Management.Automation;
using System.Reflection;
using System.Threading.Tasks;

namespace DbMigrator
{
    public static class DatabaseMigrator
    {
        public static async Task Update(VacationPackageDatabaseOptions sqlConfig, bool rollback = false)
        {
        }
```

```

        Console.WriteLine("VacationPackage.DbMigrator App has STARTED");

        await DatabaseHandler.CreateAsync(sqlConfig.MigrationConnectionString, sqlConfig.InitialCatalog);

        var serviceProvider = CreateServices(sqlConfig.ConnectionString);

        using var scope = serviceProvider.CreateScope();
        var runner = serviceProvider.GetRequiredService<IMigrationRunner>();

        DatabaseHandler.Update(runner, rollback);

        Console.WriteLine("VacationPackage.DbMigrator App work done");
    }

    private static IServiceProvider CreateServices(string connectionString)
    {
        return new ServiceCollection()
            .AddFluentMigratorCore()
            .ConfigureRunner(rb => rb
                .AddPostgres11_0()
                .WithGlobalConnectionString(connectionString)
                .ScanIn(Assembly.GetExecutingAssembly()).For.Migrations())
            .AddLogging(lb => lb.AddFluentMigratorConsole())
            .BuildServiceProvider(false);
    }
}

```

EntityFrameworkRegistration.cs

```

using DbMigrator.DbContext;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;

namespace DbMigrator
{
    public static class EntityFrameworkRegistration
    {
        public static void AddEntityFramework(this IServiceCollection services, VacationPackageDatabaseOptions config)
        {
            services.AddDbContext<VacationPackageContext>(<dbContextBuilder =>
            {
                dbContextBuilder
                    .UseNpgsql(config.ConnectionString);
            });
        }
    }
}

```

Migration202207011231InsertCountriesCitiesAndAirports.cs

```

using DbMigrator.Models;
using DbPopulator.CsvDataProcessing;
using DbPopulator.CsvDataProcessing.CsvForDatabasePopulating;
using DbPopulator.CsvDataProcessing.CsvModels;
using FluentMigrator;

namespace DbPopulator.Migrations
{
    [Migration(202207011231)]
    public class Migration202207011231InsertCountriesCitiesAndAirports : AutoReversingMigration
    {
        private const string CountryTable = "Country";
        private const string CityTable = "City";
        private const string AirportTable = "Airport";

        public override void Up()

```

```
{
    var airports = ProcessCsvData<AirportsCsvModel>.ReadRecordsFromCsv(CsvLocation.AirportCsvLocation);

    var groupedByCountryAirports = airports.GroupBy(x => x.Country);

    Console.WriteLine("Begin the population of the database.");

    foreach (var country in groupedByCountryAirports)
    {
        var currentCountryId = Guid.NewGuid();
        Insert.IntoTable(CountryTable).Row(new Country()
        {
            Id = currentCountryId,
            Name = country.Key
        });

        var citiesOfCountry = country.ToList();

        var groupedCities = citiesOfCountry.GroupBy(c => c.City);

        foreach(var city in groupedCities)
        {
            var currentCity = new City()
            {
                Id = Guid.NewGuid(),
                Name = city.Key,
                CountryId = currentCountryId
            };

            CommonUsedTablesData.Cities.Add(currentCity);

            Insert.IntoTable(CityTable).Row(currentCity);

            foreach(var airport in city)
            {
                var airportDbModel = new Airport()
                {
                    Id = Guid.NewGuid(),
                    Name = airport.Name,
                    CityId = currentCity.Id
                };

                CommonUsedTablesData.Airports.Add(airportDbModel);
                Insert.IntoTable(AirportTable).Row(airportDbModel);
            }
        }
    }
}
```

Migration202207051250InsertProperties.cs

```
using DbMigrator.Models;
using DbPopulator.CsvDataProcessing;
using DbPopulator.CsvDataProcessing.CsvForDatabasePopulating;
using DbPopulator.CsvDataProcessing.CsvModels;
using DbPopulator.Enums;
using FluentMigrator;
using FluentMigrator.SqlServer;

namespace DbPopulator.Migrations
{
    [Migration(202207051250)]
    public class Migration202207051250InsertProperties : AutoReversingMigration
    {
        private const string PropertyTable = "Property";
        private const string AmenitiesPackageTable = "AmenitiesPackage";
        private const string PlaceTypeTable = "PlaceType";
    }
}
```

```

private const string PropertyTypeTable = "PropertyType";
private const string RoomAndBedTable = "RoomAndBed";

public override void Up()
{
    var properties = ProcessCsvData<PropertyCsvModel>
        .ReadRecordsFromCsv(CsvLocation.CitiesPropertiesCsvLocation)
        .OrderBy(p => p.City)
        .GroupBy(c => c.City);

    MigratePlaceTypesInDatabase();
    MigratePropertyTypesInDatabase();

    foreach (var city in properties)
    {
        var existingCityInDb = CommonUsedTablesData.Cities.FirstOrDefault(x => x.Name == city.Key);

        if (existingCityInDb == default) continue;

        foreach (var property in city)
        {
            var amenitiesPackage = new AmenitiesPackage()
            {
                Id = Guid.NewGuid(),
                AirConditioning = property.AirConditioning,
                Dryer = property.Dryer,
                Heating = property.Heating,
                Iron = property.Iron,
                Kitchen = property.Kitchen,
                Tv = property.Tv,
                Washer = property.Washer,
                WiFi = property.WiFi
            };

            Insert.IntoTable(AmenitiesPackageTable).Row(amenitiesPackage);

            var placeTypeId = GetIdOfPlaceType(property.PlaceType);
            var propertyTypeId = GetIdOfPropertyType(property.PropertyType);

            var roomAndBed = new RoomAndBed()
            {
                Id = Guid.NewGuid(),
                Bathroom = property.Bathroom,
                Bed = property.Bed,
                Bedroom = property.Bedroom
            };

            Insert.IntoTable(RoomAndBedTable).Row(roomAndBed);

            var propertyForDb = new Property()
            {
                Id = Guid.NewGuid(),
                AmenitiesPackageId = amenitiesPackage.Id,
                CityId = existingCityInDb.Id,
                Name = property.Name,
                Pet = property.Pet,
                PlaceTypeId = placeTypeId,
                PricePerDay = property.PricePerDay,
                PropertyTypeId = propertyTypeId,
                RoomAndBedId = roomAndBed.Id
            };

            Insert.IntoTable(PropertyTable).Row(propertyForDb);
        }
    }
}

private void MigratePlaceTypesInDatabase()
{

```

```
        for (short id = 1; id < (short) PlaceTypeId.Default; id++)
        {
            Insert.IntoTable(PlaceTypeTable)
                .WithIdentityInsert()
                .Row(new
                {
                    Type = Enum.GetName(typeof(PlaceTypeId), id)!
                });
        }
    }

    private void MigratePropertyTypesInDatabase()
    {
        for (short id = 1; id < (short) PropertyTypeId.Default; id++)
        {
            Insert.IntoTable(PropertyTypeTable)
                .WithIdentityInsert()
                .Row(new
                {
                    Type = Enum.GetName(typeof(PropertyTypeId), id)!
                });
        }
    }

    private short GetIdOfPlaceType(string placeType)
    {
        return placeType switch
        {
            "EntirePlace" => (short) PlaceTypeId.EntirePlace,
            "PrivateRoom" => (short) PlaceTypeId.PrivateRoom,
            "SharedRoom" => (short) PlaceTypeId.SharedRoom,
            _ => (short) PlaceTypeId.Default
        };
    }

    private short GetIdOfPropertyType(string propertyType)
    {
        return propertyType switch
        {
            "Apartment" => (short) PropertyTypeId.Apartment,
            "GuestHouse" => (short) PropertyTypeId.GuestHouse,
            "House" => (short) PropertyTypeId.House,
            "Hotel" => (short) PropertyTypeId.Hotel,
            _ => (short) PropertyTypeId.Default
        };
    }
}
```

Migration202207090012InsertFlightCompanies.cs

```
using DbMigrator.Models;
using DbPopulator.CsvDataProcessing;
using DbPopulator.CsvDataProcessing.CsvForDatabasePopulating;
using FluentMigrator;

namespace DbPopulator.Migrations;

[Migration(202207090012)]
public class Migration202207090012InsertFlightCompanies : AutoReversingMigration
{
    private const string FlightCompanyTable = "FlightCompany";

    public override void Up()
    {
        var flightCompanies = ProcessCsvData<string>
            .ReadFieldFromCsv("FlightCompanyName", CsvLocation.FlightsCompaniesCsvLocation)
            .OrderBy(x => x).ToList();
    }
}
```



```

foreach (var flightCompany in flightCompanies)
{
    var flightCompanyDbModel = new FlightCompany
    {
        Id = Guid.NewGuid(),
        Name = flightCompany
    };

    CommonUsedTablesData.FlightCompanies.Add(flightCompanyDbModel);
    Insert.IntoTable(FlightCompanyTable).Row(flightCompanyDbModel);
}
}
}

CommonUsedTablesData.cs

using DbMigrator.Models;

namespace DbPopulator
{
    public static class CommonUsedTablesData
    {
        public static List<City> Cities = new();
        public static List<FlightCompany> FlightCompanies = new();
        public static List<Airport> Airports = new();
    }
}

ProcessCsvData.cs

using System.Globalization;
using CsvHelper;

namespace DbPopulator.CsvDataProcessing
{
    public static class ProcessCsvData<T>
    {
        public static List<T> ReadRecordsFromCsv(string csvPath)
        {
            using var reader = new StreamReader(csvPath);
            using var csv = new CsvReader(reader, CultureInfo.InvariantCulture);
            return csv.GetRecords<T>().ToList();
        }

        public static IEnumerable<string> ReadFieldFromCsv(string fieldName, string csvPath)
        {
            using var reader = new StreamReader(csvPath);
            using var csv = new CsvReader(reader, CultureInfo.InvariantCulture);
            var records = new List<string>();
            csv.Read();
            csv.ReadHeader();
            while (csv.Read())
            {
                var data = csv.GetField(fieldName);
                records.Add(data);
            }

            return records;
        }

        public static void WriteRecordsToCsv(IEnumerable<T> recordsToWrite, string csvPath)
        {
            using var writer = new StreamWriter(csvPath);
            using var csv = new CsvWriter(writer, CultureInfo.InvariantCulture);
            csv.WriteRecords(recordsToWrite);
        }
    }
}

```

Annex 5: Application testing

```
#nullable enable
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Net.Http.Json;
using System.Text;
using System.Threading.Tasks;
using Newtonsoft.Json;
using TestWebAPI.BusinessLogic;
using TestWebAPI.CustomerServicesEvaluation;
using TestWebAPI.Enums;
using TestWebAPI.PreferencesPackageRequest;
using TestWebAPI.PreferencesPackageResponse;
using VacationPackageWebApi.Infrastructure.Repositories.Models;

namespace TestWebAPI;

internal static class Program
{
    private const string BaseUrl = "http://localhost:7071/";
    private const string RequestVacationRecommendationUri = BaseUrl + "VacationPackage/RequestVacationRecommendation";
    private const string SaveEvaluationsUri = BaseUrl + "VacationPackage/SaveEvaluations";
    private const string GetFlightCompaniesForDepartureDestinationCity = BaseUrl +
    "Flight/GetFlightCompaniesForDepartureDestinationCity/";
    private const string GetFlightDepartureCities = BaseUrl + "Flight/GetFlightDepartureCities";
    private const string GetFlightArrivalCities = BaseUrl + "Flight/GetFlightArrivalCities/";

    private static readonly Random Random = new();

    private static async Task Main()
    {
        using var client = new HttpClient();

        for (var i = 1; i <= 1000; i++)
        {
            /* 1. Get all departure city flights */
            var responseOfGetFlightDepartureCities = await client.GetAsync(GetFlightDepartureCities);

            var resultOfGetDepartureCities = await responseOfGetFlightDepartureCities.Content.ReadAsStringAsync();
            var jsonGetDepartureCitiesDeserialize =
                JsonConvert.DeserializeObject<List<string>>(resultOfGetDepartureCities);

            /* 2. Get random city for departure */
            var indexOfFlightDepartureCities = Random.Next(jsonGetDepartureCitiesDeserialize!.Count);
            var randomDepartureCityCountry = jsonGetDepartureCitiesDeserialize[indexOfFlightDepartureCities];
            var randomDepartureCity = randomDepartureCityCountry.Split(", ")[0];

            /* 3. Get arrival cities based on departure */
            var responseOfGetArrivalCities = await client.GetAsync(GetFlightArrivalCities + randomDepartureCity);

            var resultOfGetArrivalCities = await responseOfGetArrivalCities.Content.ReadAsStringAsync();
            var jsonGetArrivalCitiesDeserialize =
                JsonConvert.DeserializeObject<List<string>>(resultOfGetArrivalCities);

            /* 4. Get random city for arrival */
            var indexOfFlightArrivalCities = Random.Next(jsonGetArrivalCitiesDeserialize!.Count);
            var randomArrivalCityCountry = jsonGetArrivalCitiesDeserialize[indexOfFlightArrivalCities];
            var randomArrivalCity = randomArrivalCityCountry.Split(", ")[0];

            /* 5. Get flight companies that have flights departure-destination cities */
            var payloadForFlightCompanies = $"{{randomDepartureCity}, {randomArrivalCity}}";
            var responseOfGetFlightCompaniesForDepartureDestinationCity =
                await client.GetAsync(GetFlightCompaniesForDepartureDestinationCity + payloadForFlightCompanies);

            var resultOfGetFlightCompaniesForDepartureDestinationCity =
```

```

    await responseOfGetFlightCompaniesForDepartureDestinationCity.Content.ReadAsStringAsync();
var jsonGetFlightCompaniesForDepartureDestinationCity =
    JsonConvert.DeserializeObject<List<string>>(resultOfGetFlightCompaniesForDepartureDestinationCity);

/* 6. Get flight companies that have flights for return */
var payloadForReturnFlightCompanies = $"{{randomArrivalCity}}, {{randomDepartureCity}}";
var responseOfGetFlightCompaniesForReturn =
    await client.GetAsync(GetFlightCompaniesForDepartureDestinationCity + payloadForReturnFlightCompanies);

var resultOfGetFlightCompaniesForReturn =
    await responseOfGetFlightCompaniesForReturn.Content.ReadAsStringAsync();
var jsonGetFlightCompaniesForReturn =
    JsonConvert.DeserializeObject<List<string>>(resultOfGetFlightCompaniesForReturn);

/* 7. Get departure random flight companies */
var indexOfRandomFlightCompanyDepart =
    Random.Next(jsonGetFlightCompaniesForDepartureDestinationCity!.Count);
var randomFlightCompanyDepart =
    jsonGetFlightCompaniesForDepartureDestinationCity[indexOfRandomFlightCompanyDepart];

var indexOfRandomFlightCompanyDepart1 =
    Random.Next(jsonGetFlightCompaniesForDepartureDestinationCity.Count);
var randomFlightCompanyDepart1 =
    jsonGetFlightCompaniesForDepartureDestinationCity[indexOfRandomFlightCompanyDepart1];

/* 8. Get return random flight companies */
var indexOfRandomFlightCompanyReturn = Random.Next(jsonGetFlightCompaniesForReturn!.Count);
var randomFlightCompanyReturn = jsonGetFlightCompaniesForReturn[indexOfRandomFlightCompanyReturn];

var indexOfRandomFlightCompanyReturn1 = Random.Next(jsonGetFlightCompaniesForReturn.Count);
var randomFlightCompanyReturn1 = jsonGetFlightCompaniesForReturn[indexOfRandomFlightCompanyReturn1];

/* 9. Fulfill preferences */
PreferencesRequest preferencesRequest = new()
{
    DepartureDate = DateTime.UtcNow,
    HolidaysPeriod = (short)Random.Next(3, 8), // random period from 3 to 7 days
    CustomerId = new Guid("4a77f1d2-1175-4065-860a-67ee52d5ea1e"),
    DepartureCityNavigation = new()
    {
        Name = randomDepartureCity
    },
    DestinationCityNavigation = new()
    {
        Name = randomArrivalCity
    },
    PersonsByAgeNavigation = new()
    {
        Adult = 2,
        Children = 1,
        Infant = 2
    },
    CustomerAttractionNavigation = new()
    {
        Architecture = GetRandomBool(),
        Cultural = GetRandomBool(),
        Historical = GetRandomBool(),
        IndustrialFacilities = GetRandomBool(),
        Natural = GetRandomBool(),
        Religion = GetRandomBool(),
        Other = GetRandomBool()
    },
    CustomerPropertyNavigation = new()
    {
        PlaceTypeNavigation = new PlaceTypePreferenceDto()
        {
            EntirePlace = GetRandomBool(),
            PrivateRoom = GetRandomBool(),
            SharedRoom = GetRandomBool()
        }
    }
}

```

```
    },
    AmenitiesNavigation = new()
    {
        AirConditioning = GetRandomBool(),
        Dryer = GetRandomBool(),
        Heating = GetRandomBool(),
        Iron = GetRandomBool(),
        Kitchen = GetRandomBool(),
        Tv = GetRandomBool(),
        Washer = GetRandomBool(),
        WiFi = GetRandomBool()
    },

    RoomsAndBedsNavigation = new()
    {
        // Generate random numbers from 1 to 3
        Bathrooms = (short)Random.Next(1, 4),
        Bedrooms = (short)Random.Next(1, 4),
        Beds = (short)Random.Next(1, 4)
    },

    Pets = GetRandomBool(),
    PropertyTypeNavigation = new()
    {
        Apartment = GetRandomBool(),
        GuestHouse = GetRandomBool(),
        Hotel = GetRandomBool(),
        House = GetRandomBool()
    }
    },
    CustomerFlightNavigation = new FlightDirectionPreferenceDto()
    {
        DepartureNavigation = new FlightPreferenceDto()
        {
            Class = new FlightClassDto()
            {
                Class = (short)Random.Next(1, 4)
            },
            DeparturePeriodPreference = new DeparturePeriodsPreferenceDto()
            {
                Afternoon = GetRandomBool(),
                EarlyMorning = GetRandomBool(),
                Morning = GetRandomBool(),
                Night = GetRandomBool()
            },
            FlightCompaniesNavigationList = new List<FlightCompaniesPreferenceDto>()
            {
                new()
                {
                    Company = new FlightCompanyDto
                    {
                        Name = randomFlightCompanyDepart
                    }
                },
                new()
                {
                    Company = new FlightCompanyDto
                    {
                        Name = randomFlightCompanyDepart1
                    }
                }
            },
            StopsNavigation = new StopsTypePreferenceDto
            {
                Type = (short)Random.Next(1, 4)
            }
        },
        ReturnNavigation = new FlightPreferenceDto
        {
            Class = new FlightClassDto
```

```

    {
        Class = (short)Random.Next(1, 4)
    },
    DeparturePeriodPreference = new DeparturePeriodsPreferenceDto
    {
        Afternoon = GetRandomBool(),
        EarlyMorning = GetRandomBool(),
        Morning = GetRandomBool(),
        Night = GetRandomBool(),
    },
    FlightCompaniesNavigationList = new List<FlightCompaniesPreferenceDto>
    {
        new()
        {
            Company = new FlightCompanyDto
            {
                Name = randomFlightCompanyReturn
            }
        },
        new()
        {
            Company = new FlightCompanyDto
            {
                Name = randomFlightCompanyReturn1
            }
        }
    },
    StopsNavigation = new StopsTypePreferenceDto
    {
        Type = (short) Random.Next(1, 4)
    }
}
};

var json = JsonConvert.SerializeObject(preferencesRequest, Formatting.Indented);
var data = new StringContent(json, Encoding.UTF8, "application/json");

//DateTime T = System.DateTime.UtcNow;
var response = await client.PostAsync(RequestVacationRecommendationUri, data);
//TimeSpan TT = System.DateTime.UtcNow - T; //--> Note the Time Difference

// Console.WriteLine(TT.Milliseconds);
var result = await response.Content.ReadFromJsonAsync<PreferencesResponse?>();

/* 10. Fulfill evaluation of recommendation */
await FormTheRecommendation(result, client);

var departureFlightMatchingRate = FlightRecommendationLogic.CalculateFlightSimilarityRate(preferencesRequest,
result!.FlightRecommendationResponse, "Departure");

var returnFlightMatchingRate = FlightRecommendationLogic.CalculateFlightSimilarityRate(preferencesRequest,
result.FlightRecommendationResponse, "Return");

var propertyMatchingRate = PropertyRecommendationLogic.CalculatePropertySimilarityRate(preferencesRequest,
result.PropertyPreferencesResponse);

var attractionsMatchingRate = AttractionsRecommendationLogic.CalculateAttractionsSimilarityRate(preferencesRequest,
result.AttractionsRecommendationResponse!.AttractionRecommendationList);

var mean = (departureFlightMatchingRate + returnFlightMatchingRate + propertyMatchingRate +
attractionsMatchingRate) / 4;

Console.WriteLine($"{i} {departureFlightMatchingRate} {returnFlightMatchingRate} {propertyMatchingRate}
{attractionsMatchingRate} {mean}");
}
}

private static async Task FormTheRecommendation(PreferencesResponse? result, HttpClient client)

```

```
{
    var attractionEvaluationsList = new List<AttractionEvaluationDto>();

    var attractionRecommendationList =
        result!.AttractionsRecommendationResponse!.AttractionRecommendationList.ToList();
    foreach (var attractionRecommendation in attractionRecommendationList)
    {
        var attractionEvaluation = new AttractionEvaluationDto
        {
            AttractionId = attractionRecommendation.Attraction.Xid,
            AttractionName = attractionRecommendation.Attraction.Name,
            Rate = GetRandomBool()
        };
        attractionEvaluationsList.Add(attractionEvaluation);
    }

    var serviceEvaluation = new ServiceEvaluationDto
    {
        ClientRequestId = (Guid) result.ClientRequestId!,
        AttractionEvaluation = new AllAttractionEvaluationPointDto
        {
            AttractionEvaluations = attractionEvaluationsList
        },
        FlightEvaluation = new FlightDirectionEvaluationDto
        {
            DepartureNavigation = new FlightEvaluationDto
            {
                Class = GetRandomBool(),
                Company = GetRandomBool(),
                FlightDate = GetRandomBool(),
                FlightTime = GetRandomBool(),
                Price = GetRandomBool()
            },
            ReturnNavigation = new FlightEvaluationDto
            {
                Class = GetRandomBool(),
                Company = GetRandomBool(),
                FlightDate = GetRandomBool(),
                FlightTime = GetRandomBool(),
                Price = GetRandomBool()
            }
        },
        PropertyEvaluation = new PropertyEvaluationDto
        {
            Amenities = GetRandomBool(),
            PlaceType = GetRandomBool(),
            PropertyType = GetRandomBool(),
            RoomsAndBeds = GetRandomBool()
        }
    };

    var json1 = JsonConvert.SerializeObject(serviceEvaluation, Formatting.Indented);
    var data1 = new StringContent(json1, Encoding.UTF8, "application/json");

    await client.PostAsync(SaveEvaluationsUri, data1);
}

private static bool GetRandomBool()
{
    return Random.Next(2) == 0;
}
```

Annex 6: Data log from log23-01-04-11-20-14.txt

log23-01-04-11-20-14.txt

Preferences of the user
Customer Id = ff7cf24f-7121-420d-baea-6e5dbc9cb879
Departure Date = 12-01-2023
Holidays Period = 4

Request Timestamp = 1/4/2023 11:20:14 AM

Departure City = Paris

Destination City = Cluj-Napoca

Persons by age:

Adults: 1

Children: 0

Infant: 0

Flight Preferences

Departure Flight Preferences

Flight Companies:

Departure Day Periods:

Class: Economy

Stops:

Direct

Return Flight Preferences

Flight Companies:

Departure Day Periods:

Class: Economy

Stops:

Direct

Property Preferences

Property Type:

Place Type:

Room and Beds:

Bedrooms: 0

Beds: 0

Bathrooms: 0

Amenities:

Attractions Preferences:

Flight recommendations

Departure flight:

Flight Date: 12-01-2023

Departure Time: 02:07

Class:

Economy

More flight information

Departure Country: France

Departure City: Paris

Departure Airport: Paris-Charles de Gaulle Airport

Flight Company: Air France-KLM

Flight is each: Wednesday Thursday Friday Saturday Sunday

Available departure time: 02:07 22:29 16:52

Initial assigned agent name: Agent Bob

Source agent name: Agent Bob

Return flight:

Flight Date: 16-01-2023

Departure Time: 22:49

Class

Economy

More flight information

Departure Country: Romania

Departure City: Cluj-Napoca

Departure Airport: Cluj-Napoca International Airport

Flight Company: Air Serbia

Flight is each: Monday Tuesday Wednesday Friday Saturday Sunday

Available departure time: 22:52 22:49

Initial assigned agent name: Agent Bob

Source agent name: Agent John

Property recommendations

City Cluj-Napoca

Amenities:

Kitchen

Washer

Dryer

Air Conditioning

Iron

Property type: House

Place type: PrivateRoom

Bedrooms: 1

Beds: 1

Bathrooms: 0

Initial assigned agent name: Agent John

Source agent name: Agent Bob

Attractions recommendations

City: Cluj-Napoca

Name: Babos Palace, Cluj-Napoca

Kind pattern: palaces,architecture,historic_architecture,interesting_places

City: Cluj-Napoca

Name: House of Religious Freedom

Kind pattern: historic_architecture,architecture,interesting_places,other_buildings_and_structures

City: Cluj-Napoca

Name: Palace of Justice

Kind pattern: historic_architecture,architecture,interesting_places,other_buildings_and_structures

City: Cluj-Napoca

Name: Institutul Teologic Protestant

Kind pattern: religion,other_temples,interesting_places

Initial assigned agent name: Agent Homer

Source agent name: Agent Homer

User's services evaluations

Flight evaluation

Departure flight evaluation

Class: yes

Price: yes

Proposed Flight Date: yes

Flight Time: yes

Flight Company: yes

Flight Rating: 1

Return flight evaluation

Class: yes

Price: yes

Proposed Flight Date: yes

Flight Time: yes

Flight Company: yes

Flight Rating: 1

Entire Flight Rating: 1

Property evaluation

Property type: no

Place type: no

Rooms and Beds: no

Amenities: no

Rating: 0

Attractions evaluation

Evaluated attraction: Babos Palace, Cluj-Napoca

Liked: yes

Evaluated attraction: House of Religious Freedom

Liked: no
 Evaluated attraction: Palace of Justice
 Liked: yes
 Evaluated attraction: Institutul Teologic Protestant
 Liked: yes

 Final rating: 0.75

Agents updating self expert logic

Name: Agent Bob
 Service type: Flight

Date of request: 2023-01-04
 Difference between today and request date: 1
 Original value: 1
 Actual value: 1

Service type: Attractions

Date of request: 2023-01-04
 Difference between today and request date: 1
 Original value: 0
 Actual value: 0

Name: Agent Homer
 Service type: Property

Date of request: 2023-01-04
 Difference between today and request date: 1
 Original value: 0.75
 Actual value: 0.75

Name: Agent John
 Service type: Flight

Date of request: 2023-01-04
 Difference between today and request date: 1
 Original value: 1
 Actual value: 1

Personal agents services score

	FlightScore	PropertyScore	AttractionsScore
Agent Bob	1	1	0
Agent Homer	0	0	1
Agent John	1	0	0

Personal agent rates

	FlightRating	PropertyRating	AttractionsRating
Agent Bob	1	0	0
Agent Homer	0	0	0.75
Agent John	1	0	0

Annex 7: Data log from AgentsCommunicationLog/communication.txt

Agent Agent Bob got request
 Agent Agent John got request
 Agent Agent Homer got request
 Agent Agent Bob got task type Flight
 Agent Agent John got task type Property
 Agent Agent Homer got task type Attractions
 Agent John send to Agent Bob property_recommendation_request
 Agent Homer send to Coordinator attraction_recommendation_done
 Agent John send to Agent Homer property_recommendation_request
 Agent Bob send to Coordinator departure_flight_recommendation_done

Agent Bob send to Agent Homer return_flight_recommendation_request
 Agent Bob send to Agent John return_flight_recommendation_request
 Agent Bob send to Coordinator property_recommendation_done
 Agent John send to Coordinator return_flight_recommendation_done

Annex 8: Table of recommendations

Departure Flight	Flight Date	12-01-2023
	Departure Time	22:29
	Class	Business
	Departure Country	France
	Departure City	Paris
	Departure Airport	Paris-Charles de Gaulle Airport
	Flight Company	Air France-KLM
	Flight days	Wednesday Thursday Friday Saturday Sunday
	Available departure time	02:07 22:29 16:52
	Initial assigned agent name	Agent Bob
	Source agent name	Agent Bob
Return Flight	Flight Date	16-01-2023
	Departure Time	08:13
	Class	First
	Departure Country	Romania
	Departure City	Cluj-Napoca
	Departure Airport	Cluj-Napoca International Airport
	Flight Company	Ukraine International Airlines
	Flight days	Monday Tuesday Friday
	Available departure time	23:35 08:13
	Initial assigned agent name	Agent Bob
	Source agent name	Agent John
Property	Pet	yes
	Amenities	Kitchen Washer Dryer Air Conditioning Heating TV Iron
	Property type	Hotel
	Place type	PrivateRoom
	Bedrooms	1

	Beds	1
	Bathrooms	1
	Initial assigned agent name	Agent Homer
	Source agent name	Agent Bob
Attractions	Name	Turnul Pompierilor
	Kind pattern	towers,architecture, fortifications,historic, interesting_places, observation_towers, other_fortifications, watchtowers
	Name	Dormition of the Theotokos Orthodox Cathedral
	Kind pattern	religion,cathedrals, churches, interesting_places, eastern_orthodox_churches
	Name	Unitarian Church
	Kind pattern	religion,churches, interesting_places, other_churches
	Name	Cluj-Napoca Neolog Synagogue
	Kind pattern	religion, synagogues, interesting_places
	Name	Reformed Church of the Lower Town, Cluj-Napoca
	Kind pattern	religion, other_temples, interesting_places
	Initial assigned agent name	Agent John
	Source agent name	Agent John

Annex 9: Original log file

Preferences of the user

Customer Id = ff7cf24f-7121-420d-baea-6e5dbc9cb879

Departure Date = 12-01-2023

Holidays Period = 4

Request Timestamp = 1/4/2023 5:01:02 PM

Departure City = Paris

Destination City = Cluj-Napoca

Persons by age:

Adults: 1

Children: 1

Infant: 0

Flight Preferences

Departure Flight Preferences

Flight Companies:

Red Wings Airlines

Departure Day Periods:

Night

Class: Business

Stops:

Direct

Return Flight Preferences

Flight Companies:

Ukraine International Airlines

Departure Day Periods:

Morning

Class: First

Stops:

Direct

Property Preferences

Pets: yes

Property Type:

GuestHouse

Place Type:

PrivateRoom

Room and Beds:

Bedrooms: 2

Beds: 1

Bathrooms: 1

Amenities:

Washer

Air Conditioning

Heating

Attractions Preferences:

Historical

Natural

Flight recommendations

Departure flight:

Flight Date: 12-01-2023

Departure Time: 22:29

Class:

Business

More flight information

Departure Country: France

Departure City: Paris

Departure Airport: Paris-Charles de Gaulle Airport

Flight Company: Air France-KLM

Flight is each: Wednesday Thursday Friday Saturday Sunday

Available departure time: 02:07 22:29 16:52

Initial assigned agent name: Agent Bob

Source agent name: Agent Bob

Return flight:

Flight Date: 16-01-2023

Departure Time: 08:13

Class

First

More flight information

Departure Country: Romania

Departure City: Cluj-Napoca

Departure Airport: Cluj-Napoca International Airport

Flight Company: Ukraine International Airlines

Flight is each: Monday Tuesday Friday

Available departure time: 23:35 08:13

Initial assigned agent name: Agent Bob

Source agent name: Agent John

Property recommendations

City Cluj-Napoca

Pet: yes

Amenities:

Kitchen

Washer

Dryer

Air Conditioning

Heating

TV

Iron

Property type: Hotel

Place type: PrivateRoom

Bedrooms: 1

Beds: 1

Bathrooms: 1

Initial assigned agent name: Agent Homer

Source agent name: Agent Bob

Attractions recommendations

City: Cluj-Napoca

Name: Turnul Pompierilor (1874)

Kind

pattern:

towers,architecture,fortifications,historic,interesting_places,observation_towers,other_fortifications,watchtowers

City: Cluj-Napoca

Name: Dormition of the Theotokos Orthodox Cathedral

Kind pattern: religion,cathedrals,churches,interesting_places,eastern_orthodox_churches

City: Cluj-Napoca

Name: Unitarian Church

Kind pattern: religion,churches,interesting_places,other_churches

City: Cluj-Napoca

Name: Cluj-Napoca Neolog Synagogue

Kind pattern: religion,synagogues,interesting_places

City: Cluj-Napoca

Name: Reformed Church of the Lower Town, Cluj-Napoca

Kind pattern: religion,other_temples,interesting_places

Initial assigned agent name: Agent John

Source agent name: Agent John
