UNIVERISTY OF PIRAEUS - DEPARTMENT OF INFORMATICS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

MSc «INFORMATICS»

ΠΜΣ «ΠΛΗΡΟΦΟΡΙΚΗ»

MSc Thesis

Μεταπτυχιακή Διατριβή

Thesis Title: Τίτλος Διατριβής:	**ACCOMMODATION MANAGEMENT SOFTWARE** ΛΟΓΙΣΜΙΚΟ ΔΙΑΧΕΙΡΙΣΗΣ ΚΑΤΑΛΥΜΑΤΩΝ
Student's name-surname: Ονοματεπώνυμο φοιτητή:	**CHARALAMPOS SIDERIS** ΧΑΡΑΛΑΜΠΟΣ ΣΙΔΕΡΗΣ
Father's name: Πατρώνυμο:	**KONSTANTINOS** ΚΩΝΣΤΑΝΤΙΝΟΣ
Student's ID No: Αριθμός Μητρώου:	ΜΠΠΛ17046
Supervisor: Επιβλέπων:	**EFTHIMIOS ALEPIS, Associate Professor** ΕΥΘΥΜΙΟΣ ΑΛΕΠΗΣ, Αναπληρωτής Καθηγητής

July 2023/ Ιούλιος 2023

3-Member Examination Committee

Τριμελής Εξεταστική Επιτροπή

Efthimios Alepis	**Maria Virvou**	**Constantinos Patsakis**
Associate Professor	**Professor**	**Associate Professor**
Ευθύμιος Αλέπης	Μαρία Βίρβου	Κωνσταντίνος Πατσάκης
Αναπληρωτής Καθηγητής	Καθηγήτρια	Αναπληρωτής Καθηγητής

Table of Contents

Summary 4
Intoduction ..... **Error! Bookmark not defined.**
Purpose of paper 5
Field review 6
User's Manual 7
Villas Management Sytem 7
Table of Contents 7
Introduction 7
Login 7
CRUD for 7
Architecture 7
Dependency Injection 7
MVC Architecture 8
Fully-Driven Web Application 10
Entities 11
Entity Framework 12
Data Annotations 12
Repository Pattern 12
Git Hub for Source Control A 13
Appendices 15
Conclusion 18
References 19

Summary

As the developer tasked with implementing the API for managing Customers, Orders, and Products, I am passionate about creating efficient and robust solutions that meet the needs of the modern software landscape. With a strong foundation in C# and a comprehensive understanding of software design principles, I embarked on the challenge of developing an API that adheres to best practices and incorporates essential features.

Throughout the implementation process, I prioritized the utilization of industry-standard tools and patterns to ensure a scalable and maintainable solution. By employing Entity Framework as the persistence framework and implementing the repository pattern, I established a clear separation between the application logic and data persistence concerns. The inclusion of the unit of work pattern further enhanced data integrity and consistency, guaranteeing reliable operations on multiple entities within a single transaction.

To fulfill the requirements of the project, I integrated CRUD operations for Products, enabling administrators to manage the master data while ensuring the integrity of the system. Additionally, I ensured that the API handles scenarios where multiple Customers may share the same name, reflecting the real-world complexity of customer data management.

Recognizing the significance of documentation, I meticulously documented the API using XML comments, providing comprehensive and easily accessible information about its structure, usage, and available methods. By incorporating unit tests using the NUnit framework, I verified the behavior and functionality of the API, ensuring its correctness, reliability, and adherence to specifications.

Embracing modern software development principles, I implemented Domain-Driven Design (DDD) to focus on the core domain logic and encapsulate business rules within the appropriate entities. The adoption of the Command Query Responsibility Segregation (CQRS) pattern allowed for a clear separation between write and read operations, optimizing performance and scalability.

Throughout the project, I made informed decisions based on assumptions and considerations, striving to fulfill the objectives of the API. I ensured that the API supports necessary validations, such as the maximum length of fields, the requirement for a Customer name, and constraints on price and quantity. I also addressed crucial aspects like the non-deletion of Products and the exclusive availability of the Products controller to administrators.

In conclusion, this paper outlines my journey in designing and implementing an API that manages Customers, Orders, and Products, showcasing my technical skills, attention to detail, and commitment to delivering a high-quality solution. By incorporating industry-standard practices, patterns, and documentation, I have developed an API that meets the project requirements and demonstrates my proficiency as a developer in designing and implementing reliable and scalable systems.

Introduction

In this university paper, we will explore the development of a robust web application built with .NET 7 API and MVC, focusing on creating and managing villas. The application utilizes modern development methodologies, including the Code-First approach with Entity Framework, Repository Pattern, N-Tier Architecture, and Razor Pages for the front end. The paper delves into the functionality of the app, its architectural design, and the various techniques employed to ensure seamless CRUD operations and data validation.

1. Background and Project Scope:
   The project's main objective was to build a feature-rich web application that offers functionality for creating and managing villas. It is aimed at providing users with a streamlined experience to add new villas, generate unique Villa numbers based on villa type, and perform CRUD operations for different villa types. The project utilized .NET 7 for its robustness and the MVC framework to handle the presentation layer.
2. Code-First Approach and Entity Framework:
   To design and manage the database, the Code-First approach was adopted. This approach allows developers to define the data model in code and then automatically generate the

corresponding database schema. Entity Framework, a popular ORM (Object-Relational Mapping) tool, was utilized to facilitate seamless communication between the application and the database. The utilization of migrations enabled efficient database schema updates and versioning.

3. N-Tier Architecture and Repository Pattern:
   The application was designed using the N-Tier architecture, a software design pattern that separates the application into distinct layers, each with a specific responsibility. The three main layers employed in this app were the Presentation Layer (MVC), Business Logic Layer (BLL), and Data Access Layer (DAL). The Repository Pattern was used to abstract the data access logic from the BLL, providing a flexible and maintainable solution for data manipulation.
4. Data Transfer Objects (DTOs):
   To ensure clean and efficient data exchange between the different layers, Data Transfer Objects (DTOs) were implemented. Each CRUD operation was associated with its specific DTO, allowing tailored data models for different business cases. This approach minimized data over-fetching and provided optimized data handling.
5. Villa Creation and Villa Number Generation:
   The application's core functionality revolves around creating villas and generating unique villa numbers based on the villa type. Users are guided through a user-friendly interface in the Razor Pages to input villa details and select the appropriate villa type. Upon submission, the application generates and assigns a unique villa number to each newly created villa.
6. CRUD Operations and Data Validation:
   The application provides full CRUD (Create, Read, Update, Delete) capabilities for villas. Users can easily add, modify, and remove villas as needed. To maintain data integrity and enforce business rules, comprehensive data validation mechanisms were implemented. This ensured that only valid data is processed and stored in the database.

This university paper delved into the development of a powerful web application built with .NET 7 API and MVC. The app's core functionality revolves around villa creation and unique villa number generation based on the villa type. The use of Code-First Approach, Entity Framework, N-Tier Architecture, and the Repository Pattern ensured a robust and scalable solution. The implementation of DTOs optimized data handling, and the Razor Pages offered a user-friendly interface. With a focus on data validation, the application maintains data integrity throughout all CRUD operations. Overall, this project showcases the capabilities of modern web development technologies and their potential to create feature-rich applications that cater to specific business needs.

Purpose of paper

The purpose of this paper is to provide students with a solid understanding of modern web development, without relying on popular platforms such as WordPress or Magento. Instead, the paper focuses on the use of .NET Core and related technologies to build web applications. The paper will likely cover the following topics:

- Introduction to .NET Core and its architecture
- Using the Entity Framework for data access and management
- Implementing the Repository pattern for managing data access
- Creating and consuming web services using .NET Core
- Custom authentication and authorization using .NET Core
- Deployment and hosting of .NET Core web applications

The innovation of this paper is the combination of these technologies, such as using the Entity Framework, .NET Core architecture, Repository pattern and custom authentication and authorization to build web applications. By covering these topics, the paper aims to give students a comprehensive understanding of how these technologies can be used together to build modern web applications and to understand the benefits of using such an architecture.

The paper aims to cover the following topics to give students a solid understanding of modern web development using .NET Core:

- N-Tier Architecture: The paper will cover the principles and benefits of using an N-Tier architecture, which divides an application into layers such as presentation, business, and data access layers.
- Repository patter: The paper will cover the Repository pattern which provide a way to abstract the data access layer from the business logic layer and help to improve the maintainability and testability of the application.
- Dependency Injection.
- TempData – ViewBag - ViewData in .NET Core: The paper will cover how to use TempData, ViewBag, and ViewData in .NET Core to pass data between controllers and views.
- Sweet Alerts-Rich Text Editor and Data Table in .NET Core: The paper will cover how to use the Sweet Alerts, Rich Text Editor and Data Table libraries in .NET Core to create interactive and user-friendly web pages.
- MVC Architecture.
- API Controllers with .NET Core: The paper will cover how to create and consume web services using .NET Core's API controllers.
- Scaffold Identity with (Razor Class Library): The paper will cover how to scaffold the built-in identity system in .NET Core using Razor class library.
- Roles and Authorization in .NET Core: The paper will cover how to implement roles and authorization in .NET Core to control access to resources and functionality.

In summary, the paper will cover several topics related to modern web development using .NET Core, including architecture, patterns, data access, UI libraries, web services, authentication and authorization.

Field review

My application's functionality for managing villas and generating unique villa numbers can be compared to real-world apps like Airbnb, HomeAway, Vrbo and Booking. These platforms are widely recognized as popular vacation rental marketplaces, and they incorporate similar features and technologies as my app.

- Airbnb:
  Airbnb, founded in 2008, is one of the most well-known and successful vacation rental platforms globally. It connects travelers with unique accommodations offered by hosts in over 220 countries and regions. Property owners can list various types of properties, including villas, apartments, houses, and even castles, providing travelers with a wide range of options to choose from. Airbnb's user-friendly interface, extensive property listings, and secure payment system have made it a go-to platform for vacation rentals.
- Vrbo (Vacation Rentals by Owner):
  Vrbo, founded in 1995 and now part of Expedia Group, is another popular vacation rental platform. It caters to vacation homeowners and property managers who list properties like villas, cabins, and beach houses. Vrbo provides a platform for travelers seeking unique and personalized vacation experiences.
- Booking.com:
  While Booking.com is primarily recognized as a hotel booking platform, it has expanded its services to include alternative accommodations such as vacation rentals, apartments, and villas. ravelers can browse through a diverse range of accommodations to find suitable vacation rental options.

In conclusion, real-world apps like Airbnb, Vrbo, and Booking.com have revolutionized the vacation rental and property management industry, offering travelers an array of unique accommodations and providing property owners with efficient platforms to list and manage their properties. These platforms employ modern web technologies, user-friendly interfaces, and secure payment systems, making them trusted choices for millions of travelers worldwide.

User's Manual

Villas Management System

In this section you will find the user’s manual guideline.

Table of Contents

1. Introduction
2. Login
3. CRUD for
    - Villas
    - Villas Number

Introduction

The application for villas Management is designed to facilitate the creation, retrieval, updating, and deletion of Villas, Villas Numbers, Register and Login. This user's manual aims to guide you through the process of utilizing the API effectively. However, you can see the available villas as well as the

Login

The Application need not authentication to access its functionality. If the user has not registered in the application, he/she cannot perform the actions of the application. However, he/she can see the available villas as well as the villas numbers.

CRUD for

The user can do all the operations (CRUD) select, delete, and update for villas and villas number.

Also, she/he can see all the villas and villas number.

Architecture

The architecture of the project was designed with a focus on domain logic and draws inspiration from the principles of Domain-Driven Design (DDD). By following DDD principles, the project aims to align closely with the business domain and ensure a clear separation of concerns. Utilizing the following concepts, we achieve a serviceable, maintainable, and robust codebase.

Dependency Injection

Dependency injection (DI) is a software design pattern that allows a class to receive its dependencies from an external source, rather than creating them directly. This helps to decouple the class from its dependencies, making the code more flexible and easier to test.

There are three main components in dependency injection:

- The service: This is the class that has a dependency.
- The dependency: This is the class that the service relies on.
- The injector: This is the component responsible for providing the service with its dependency.

In practice, dependency injection is often implemented using a dependency injection framework, which provides a way to configure the dependencies and automatically inject them into the service

when it is created. This can be done through the use of constructor injection, setter injection or interface injection.

Dependency injection can help to improve the maintainability and testability of code by making it easier to swap out dependencies for testing or to update them. It also makes the code more modular and reusable.

It also makes it easier to understand the dependencies of a class and how they are used and allows for better separation of concerns.

Advantages of Dependency Injection:

- Loose coupling: Dependency injection promotes loose coupling between objects and classes, making it easier to change or replace components without affecting the rest of the system.
- Testability: By injecting dependencies into a class, it becomes easier to write unit tests for that class, as you can easily substitute mock objects for the real dependencies.
- Flexibility: Dependency injection makes it easy to switch between different implementations of a particular dependency, making it easy to change the behavior of a class at runtime.
- Reusability: Classes that use dependency injection are more reusable, as they are not tightly coupled to a specific implementation of a dependency.
- Maintainability: Dependency injection helps to make code more maintainable by making it easier to understand how a class interacts with its dependencies and how they are used in the larger system.

Disadvantages of Dependency Injection

One of the main disadvantages of dependency injection is the increased complexity of the code. Injecting dependencies into a class requires additional boilerplate code, such as constructors or property setters, which can make the code more difficult to understand and maintain. Additionally, using dependency injection frameworks can add an additional layer of abstraction, making it harder to understand how the dependencies are being used and how they relate to the rest of the system.

Another disadvantage is the performance overhead. Depending on the implementation of the dependency injection framework, it can add some overhead to the application's performance. This can be more pronounced in applications that have a large number of dependencies or in scenarios where the dependencies are frequently created and destroyed.

Another disadvantage is that it can introduce additional dependencies to the system. When using a dependency injection framework, it can introduce additional dependencies to the system, making it harder to understand how the application is put together and how to deploy it.

Finally, dependency injection can also make it more difficult to debug an application. When using dependency injection, it can be harder to understand what is happening at runtime because the dependencies are being created and managed by the framework, rather than being created directly in the code.

Overall, while dependency injection can have many benefits, it also has some potential drawbacks. It's important to consider these drawbacks when deciding whether to use dependency injection in your application and to choose the right framework that meets your needs.

MVC Architecture

The Model-View-Controller (MVC) pattern is a widely used architectural pattern for building web applications. It is a design pattern that separates the application's concerns into three main components: the Model, the View, and the Controller.

The Model represents the data and the business logic of the application. It is responsible for managing the data and performing any data-related operations.

The View represents the user interface of the application. It is responsible for displaying the data to the user and handling user input.

The Controller is the intermediary between the Model and the View. It receives user input, updates the Model and updates the View to reflect the changes in the Model.

How MVC is used in web app development:

In web app development, MVC is used to separate the concerns of the application into different components, making the code more organized and easier to maintain. MVC allows for a separation of concerns, which means that the developer can focus on a specific area of the application without having to worry about the other areas. This allows for a more efficient development process and makes it easier to make changes to the application without affecting the entire codebase.

Advantages

- Separation of concerns: The Model, View, and Controller components of the MVC pattern are separated, which allows developers to focus on specific areas of the application without having to worry about the other areas. This makes the code more organized, easier to maintain and understand.
- Code Reusability: MVC allows for code reusability by separating the concerns of the application, components can be reused across different areas of the application, for example, the same view component can be used across different controllers, this can reduce the amount of code needed and make it easier to maintain.
- Modularity: MVC allows for a modular approach to building web applications. This means that different parts of the application can be developed and tested separately before being integrated together. This makes it easier to make changes to the application without affecting the entire codebase.
- Testing: MVC allows for easier testing of different parts of the application separately, this makes it easier to identify and fix bugs, also it makes it easier to implement automated testing.
- Efficient Development: MVC allows for a more efficient development process by separating the concerns of the application, this allows developers to work on different areas of the application simultaneously, which can speed up the development process.
- Popularity: MVC is widely used in web development, this means that there are a lot of resources

Disadvantages

- Complexity: MVC can add complexity to the application, especially for small projects. The separation of concerns can make the application harder to understand and navigate for developers who are not familiar with the pattern.
- Boilerplate code: MVC can lead to a lot of boilerplate code, particularly in the controller and view components. This can make the codebase harder to maintain and understand.
- Flow of the application: It can be harder to understand the flow of the application if the codebase is not well structured. This can make it harder to debug issues and make changes to the application.
- Debugging: It can be harder to debug issues if the codebase is not well organized, this is because the flow of the application is harder to understand.
- Not always suitable: MVC may not always be the best choice for certain types of web applications. For example, if the application is a simple CRUD application, MVC may be overkill and a simpler pattern such as MVVM may be more suitable.
- Learning curve: MVC can have a steeper learning curve for developers who are not familiar with the pattern. It takes time for developers to understand the concepts and best practices of MVC, it may not be suitable for small projects where time is of the essence.

MVC is a widely used architectural pattern for building web applications, but it also has its own disadvantages. It can add complexity to the application, lead to a lot of boilerplate code, make it harder

to understand the flow of the application and make it harder to debug issues. It may not always be the best choice for certain types of web applications, it also has a steeper learning curve for developers who are not familiar with the pattern.

URL pattern in MVC (Routing)

In the Model-View-Controller (MVC) pattern, URL routing is the process of mapping URLs to specific actions or controllers in an application.

In ASP.NET Core MVC, routing is configured using the routing middleware, which maps incoming URLs to specific controllers and actions. The routing middleware is typically added to the pipeline in the Startup.cs file.

URL routing in ASP.NET Core MVC is defined using routing templates. A routing template is a string that defines the pattern of a URL and can include placeholders for variable values. For example, a routing template for a URL that includes an ID parameter might look like this: /customers/{id}

When a request is received, the routing middleware compares the incoming URL to the routing templates that have been defined in the application. The first template that matches the URL is used to determine which controller and action should handle the request.

In the routing template above, the {id} is a placeholder for a variable, and the value of this variable is passed to the action as a parameter.

You can also define default values for some of the placeholders and also constrain the values that can be passed as part of the URL.

In addition, you can also define routes with attribute routing, where you can specify the route directly on the action, this allows for more fine-grained control over the routing.

Routing in MVC plays a critical role in the architecture of the application, it allows for a clean separation of concerns, and also enables search engine optimization by having a meaningful URL structure.

Fully-Driven Web Application

Building a fully driven web application involves several steps, which can vary depending on the specific requirements and complexity of the project. However, a general outline of the steps involved in building a web application would be:

- Planning and Analysis: This step involves gathering the requirements for the application and creating a plan for the project. This includes identifying the target audience, the goals of the application, and the features that need to be implemented.
- Designing: This step involves creating the visual design and layout of the application, including the user interface, navigation, and overall look and feel. This step also includes wireframing and prototyping of the application to create a visual representation of the final product.
- Architecture: This step involves designing the architecture of the application, which includes determining the technology stack, choosing the appropriate frameworks, and designing the database schema.
- Development: This step involves writing the code for the application, including the front-end and back-end code, as well as any necessary integration with third-party services or APIs.
- Testing: This step involves testing the application to ensure that it is functioning correctly and that all the features are working as expected. This includes unit testing, integration testing, and end-to-end testing.
- Deployment: This step involves deploying the application to a web server or hosting platform. This step also includes configuring the server, setting up security measures, and ensuring that the application is accessible to the target audience.

- Maintenance: This step involves maintaining and updating the application to fix bugs, add new features, and ensure that the application is secure and up to date.

Building a web application is a process that involves several steps including Planning and Analysis, Designing, Architecture, Development, Testing, Deployment, and Maintenance. Each step is important to ensure that the final product is functional, secure, and easy to use.

Building a fully driven web application using .NET Core involves using a variety of techniques and tools to ensure that the final product is functional, secure, and easy to use. Some of the common techniques and tools used in building a web application in .NET Core include:

- .NET Core Framework: .NET Core is a cross-platform, open-source framework that provides a structure for building web applications. It can be used to build web applications using C# and the .NET framework.
- ASP.NET Core: ASP.NET Core is a web framework for building web applications using .NET Core. It provides features such as routing, model binding, and dependency injection to help developers build web applications quickly and efficiently.
- Entity Framework Core: Entity Framework Core is an ORM framework that allows developers to interact with databases using object-oriented code, rather than writing raw SQL queries. It provides a way to map between the database schema and the classes in the application, thus allowing developers to work with entities (objects) that represent the data in the database.
- Authentication and Authorization: .NET Core includes built-in support for authentication and authorization using technologies such as OAuth and OpenID Connect.
- Visual Studio: Visual Studio is a popular development environment for .NET Core applications, it provides a rich set of tools for debugging, testing, and deploying web applications.
- Git: Git is a distributed version control system that allows developers to track changes to their code and collaborate with others on the same project.
- Docker: Docker is a platform that allows developers to package and deploy their applications in containers, this allows them to run the same application on different environments, this also allows to test and deploy the application in a reliable and consistent way.

Building a web application in .NET Core involves using a variety of techniques and tools, such as the .NET Core framework, ASP.NET Core, Entity Framework Core, Razor Class Library, built-in authentication and authorization, Visual Studio, Git and Docker, these tools and techniques help to ensure that the final product is functional, secure, and easy to use.

Entities

Entities are plain old C# classes that represent the data in a web application. In the context of web app development, entities are used to represent the data that is stored in a database and interact with the data access layer to retrieve and persist data. The entities are typically defined with properties that correspond to the columns in the database table, and the Entity Framework automatically generates the SQL statements for CRUD (Create, Read, Update, Delete) operations based on the configuration.

Entities play a crucial role in web app development by providing a way to separate the data access logic from the business logic and the presentation logic. By using entities, developers can work with objects that represent the data in the database, rather than writing raw SQL queries. This allows them to focus on the business logic of the application, while the Entity Framework takes care of the data access logic.

Entities also provide a way to enforce data validation and data integrity by using the properties of the entities. This allows developers to define validation rules on the entities, such as required fields, maximum length, and data type validation, which can help to prevent errors and inconsistencies in the data.

Entities also enable the use of Object-Relational Mapping (ORM) frameworks like the Entity Framework, which provide several features such as Code First, Database First, LINQ and Lazy Loading, these features help developers to work with databases more efficiently.

Entities play a crucial role in web app development by providing a way to separate the data access logic from the business logic, enforce data validation and data integrity and by enabling the use of ORM frameworks. Entities are central to maintain the separation of concerns and to structure the code in a maintainable and extensible way, which are essential for web app development.

Entity Framework

The Entity Framework is an Object-Relational Mapping (ORM) framework that allows developers to interact with databases using object-oriented code, rather than writing raw SQL queries. It provides a way to map between the database schema and the classes in the application, thus allowing developers to work with entities (objects) that represent the data in the database.

An Entity is a plain old C# class that represents a table or a view in the database. It has properties that correspond to the columns in the table, and the Entity Framework generates the SQL statements for CRUD (Create, Read, Update, Delete) operations automatically, based on the configuration.

The Entity Framework provides several features to help developers work with databases more efficiently:

1) Object-Relational Mapping (ORM): The Entity Framework allows developers to map between the database schema and the classes in the application. This means that developers can work with entities (objects) that represent the data in the database, rather than writing raw SQL queries.

2) Code First: The Entity Framework allows developers to create a database based on the class definitions, this is known as Code First approach, this eliminates the need to create the database schema first and then write the code to match the schema, this approach allows developers to focus on the domain model rather than the database structure.

3) Database First: The Entity Framework allows developers to generate the classes based on the existing database, this is known as Database First approach, this eliminates the need for developers to create the class structure manually, it allows developers to start working with the existing database structure.

4) LINQ: The Entity Framework allows developers to use LINQ (Language Integrated Query) to query data from the database using a syntax that is like SQL. LINQ allows developers to write type-safe, expressive, and efficient queries.

5) Lazy Loading: The Entity Framework allows developers to load related data from the database only when it is needed. This can help to improve the performance of the application and reduce the amount of data that needs to be loaded from the database.

In summary, the Entity Framework is a powerful ORM framework that allows developers to interact with databases using object-oriented code, it provides several features such as Object-Relational Mapping, Code First, Database First, LINQ and Lazy Loading, these features help developers to work with databases more efficiently.

Data Annotations

Data Annotations are used in conjunction with Entity Framework to define metadata and validation rules for the domain entities. These annotations allow for declarative configuration of the entities, ensuring data integrity and enforcing business rules.

Repository Pattern

The repository pattern is a design pattern that is used to abstract the data access logic in a web application. It is a way to separate the data access logic from the business logic and the presentation logic. The repository pattern provides a way to interact with the data storage (usually a database) in a consistent and unified way.

The repository pattern defines a set of methods that can be used to retrieve, add, update, and delete data. These methods are implemented in a separate class, called a repository, which is responsible for

interacting with the data storage. The repository class provides a simple and consistent interface for the application to interact with data storage.

The repository pattern can be used with different types of data storage, including databases, file systems, and web services. It can also be used with different types of data access frameworks, such as the Entity Framework, NHibernate, and ADO.NET.

Advantages of using the repository pattern:

- Separation of concerns: The repository pattern separates the data access logic from the business logic and the presentation logic. This makes the code more organized and easier to maintain.
- Code reusability: The repository pattern allows for code reusability by providing a consistent interface for the application to interact with the data storage.
- Testability: The repository pattern makes it easier to test the data access logic by isolating it from the rest of the application.
- Flexibility: The repository pattern can be used with different types of data storage and data access frameworks, which makes it more flexible.
- Single Responsibility Principle: The repository pattern follows the Single Responsibility Principle (SRP) by having a single class responsible for interacting with the data storage.

Disadvantages of using the repository pattern:

- Over-abstraction: The repository pattern can lead to over-abstraction of the data access logic, which can make the code more complex and harder to understand.
- Increased complexity: The repository pattern can increase the complexity of the application by adding an extra layer of abstraction. This can make the code more difficult to understand and maintain.
- Performance issues: Using the repository pattern can lead to performance issues if not implemented correctly, for example, if the repository retrieves more data than necessary or makes more database calls than necessary.
- Increased codebase: The repository pattern can increase the codebase by adding an extra layer of abstraction. This can make the codebase harder to maintain and understand.
- Not always suitable: The repository pattern may not always be the best choice for certain types of applications. For example, if the application has a simple data access layer, the repository pattern may be overkill.
- Leaky Abstraction: In some cases, the repository pattern can hide the underlying database structure and makes it difficult to know how the data is being stored and retrieved.

Git Hub for Source Control A

Source control is a system that allows developers to track and manage changes made to the source code of a software project. It is an essential tool for software development, as it enables teams to collaborate on a codebase, maintain a history of changes, and revert to previous versions if necessary.

One of the main benefits of source control is the ability to collaborate on a codebase. When multiple developers are working on a project, it can be difficult to keep track of who made what changes and when. Source control systems like Git allow multiple developers to work on the same codebase simultaneously and keep track of the changes that each developer makes. This allows teams to collaborate effectively and avoid conflicts when merging changes.

Source control also provides a way to maintain a history of changes made to the codebase. This allows developers to understand how the codebase has evolved over time and to revert to previous versions if necessary. This is particularly useful when debugging or troubleshooting issues, as developers can easily review the changes that were made and identify the cause of a problem.

Another important benefit of source control is the ability to branch and merge code. This allows developers to work on new features or bug fixes in isolation, without affecting the main codebase. Once the changes have been tested and verified, they can be merged back into the main branch, ensuring that the codebase remains stable and functional.

In summary, source control is an essential tool for software development teams, as it enables effective collaboration, maintainability, and traceability of the codebase. Git is one of the most popular and widely used source control systems, providing a robust and flexible platform for managing code changes.

How Git Hub is used for source control

GitHub is a web-based platform that uses the Git source control system to manage code repositories. It provides a centralized location where developers can store, share, and collaborate on code.

When using GitHub for source control, developers can create a new repository for their project, which serves as the central location for storing and managing the source code. Each developer can then clone the repository to their local machine, allowing them to work on the code locally and make changes.

As changes are made, developers can commit their changes to the local repository and push them to the remote repository on GitHub. This allows other members of the team to see and review the changes that have been made. GitHub provides a web-based interface for reviewing and managing commits, which makes it easy for developers to collaborate on the codebase.

GitHub also provides a system for creating branches and merging code. This allows developers to work on new features or bug fixes in isolation, without affecting the main codebase. Once the changes have been tested and verified, they can be merged back into the main branch, ensuring that the codebase remains stable and functional.

GitHub also offers a few other features, such as pull requests and code review, that make it easy for teams to collaborate on code and maintain a high-quality codebase. Additionally, it provides a useful tool for tracking issues, milestones, and project management.

In summary, GitHub is a web-based platform that uses the Git source control system to manage code repositories, making it easy for developers to store, share, and collaborate on code. It provides a centralized location for managing code and additional features that make it easy for teams to collaborate and maintain a high-quality codebase.

Best practices

Here are some best practices for using GitHub for source control:

- Use branches for different features or bug fixes: Create a separate branch for each new feature or bug fix that you work on. This allows you to work on the code in isolation, without affecting the main codebase. Once you have completed your changes, you can merge the branch back into the main codebase.
- Use Pull Requests to review code changes: Use pull requests to review and approve code changes before they are merged into the main codebase. This allows other members of the team to review the changes and provide feedback, ensuring that the codebase remains stable and functional.
- Use descriptive commit messages: Use descriptive commit messages that clearly explain the changes that have been made. This makes it easier for other members of the team to understand the changes and why they were made.
- Keep your branches up to date: Make sure to keep your branches up to date with the main codebase by regularly pulling in changes from the main branch. This ensures that your code changes are based on the most recent version of the codebase.
- Use GitHub issues to track bugs and features: Use GitHub issues to track bugs, features, and other tasks. This allows you to easily keep track of what needs to be done and who is working on it.
- Collaborate with other members of the team: Collaborate with other members of the team to maintain the codebase. Share knowledge, review code changes and work together to improve the quality of the codebase.
- Keep your codebase organized: keep the codebase organized and clean. This makes it easier to understand and maintain, and helps to keep the codebase in good shape.

- Regularly backup your codebase: Regularly backup your codebase to ensure that your code is safe and can be recovered if necessary.

By following these best practices, you can ensure that your team is able to collaborate effectively and maintain a high-quality codebase using GitHub.

Appendices

```
5 references
public class Repository<T> : IRepository<T> where T : class
{
    private readonly ApplicationDbContext _db;
    internal DbSet<T> dbSet;
    2 references
    public Repository(ApplicationDbContext db)
    {
        _db = db;
        //_db.VillaNumbers.Include(u => u.Villa).ToList();
        this.dbSet=_db.Set<T>();
    }
```

Figure 1 Dependency Injection Example

```
namespace MagicVilla_VillaAPI.Repository.IRepostiory
{
    3 references
    public interface IRepository<T> where T : class
    {
        4 references
        Task<List<T>> GetAllAsync(Expression<Func<T, bool>>? filter = null, string? includeProperties = null,
            int pageSize = 0, int pageNumber = 1);
        10 references
        Task<T> GetAsync(Expression<Func<T, bool>> filter = null, bool tracked = true, string?
          includeProperties = null);
        3 references
        Task CreateAsync(T entity);
        3 references
        Task RemoveAsync(T entity);
        3 references
        Task SaveAsync();
    }
}
```

Figure 2 IRepository Example

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    0 references
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    3 references
    public DbSet<ApplicationUser> ApplicationUsers { get; set; }

    0 references
    public DbSet<LocalUser> LocalUsers { get; set; }
    1 reference
    public DbSet<Villa> Villas { get; set; }
    1 reference
    public DbSet<VillaNumber> VillaNumbers { get; set; }
```

Figure 3 Application DB Context

```
[HttpGet("{id:int}", Name = "GetVilla")]
[ProducesResponseType(StatusCodes.Status403Forbidden)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
//[ProducesResponseType(200, Type =typeof(VillaDTO))]
  [ResponseCache(Location =ResponseCacheLocation.None,NoStore =true)]
0 references
public async Task<ActionResult<APIResponse>> GetVilla(int id)
{
    try
    {
        if (id == 0)
        {
            _response.StatusCode = HttpStatusCode.BadRequest;
            return BadRequest(_response);
        }
        var villa = await _dbVilla.GetAsync(u => u.Id == id);
        if (villa == null)
        {
            _response.StatusCode = HttpStatusCode.NotFound;
            return NotFound(_response);
        }
        _response.Result = _mapper.Map<VillaDTO>(villa);
        _response.StatusCode = HttpStatusCode.OK;
        return Ok(_response);
    }
    catch (Exception ex)
    {
        _response.IsSuccess = false;
        _response.ErrorMessages
             = new List<string>() { ex.ToString() };
    }
    return _response;
}
```

Figure 4 Villa API Controller

Figure 5 AppSettings Json

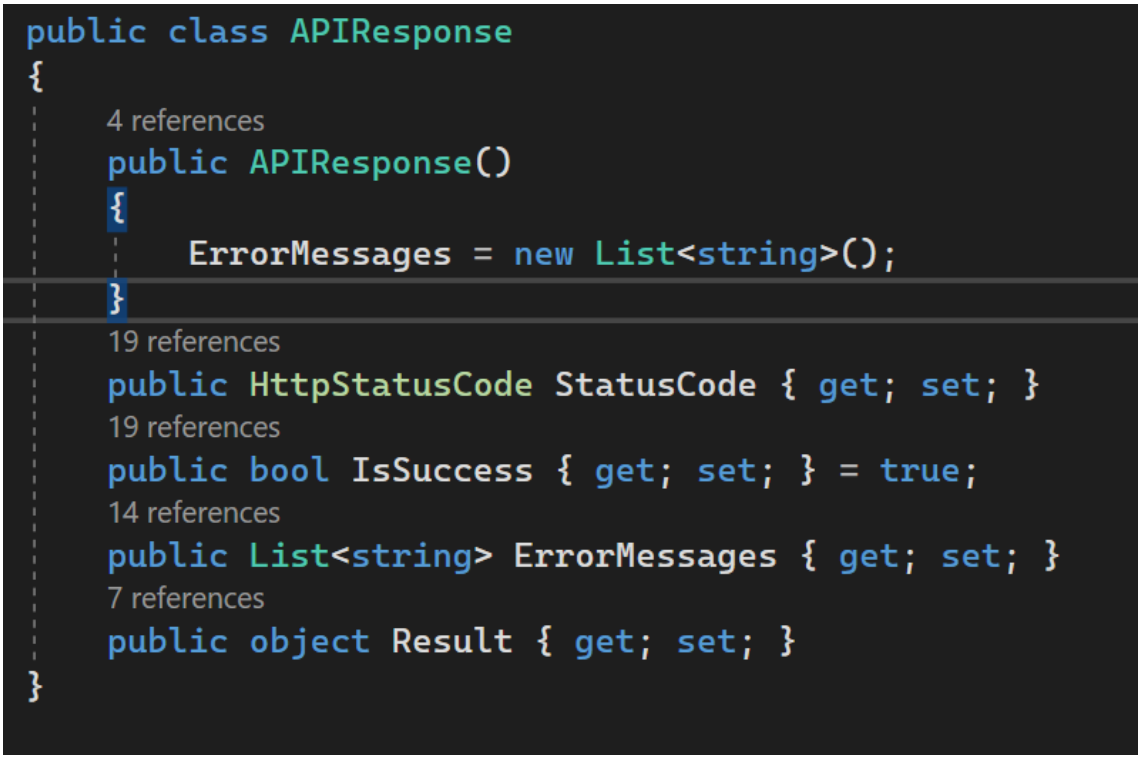

Figure 6 APIResponse Model

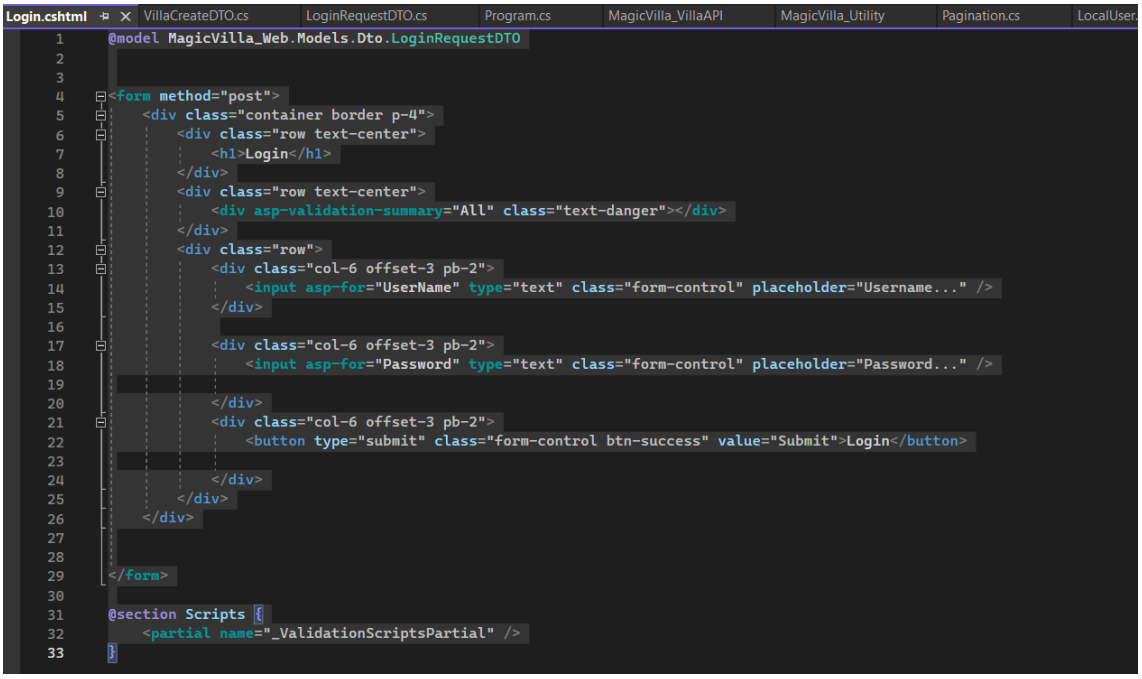

Figure 7 Login.cshtml file

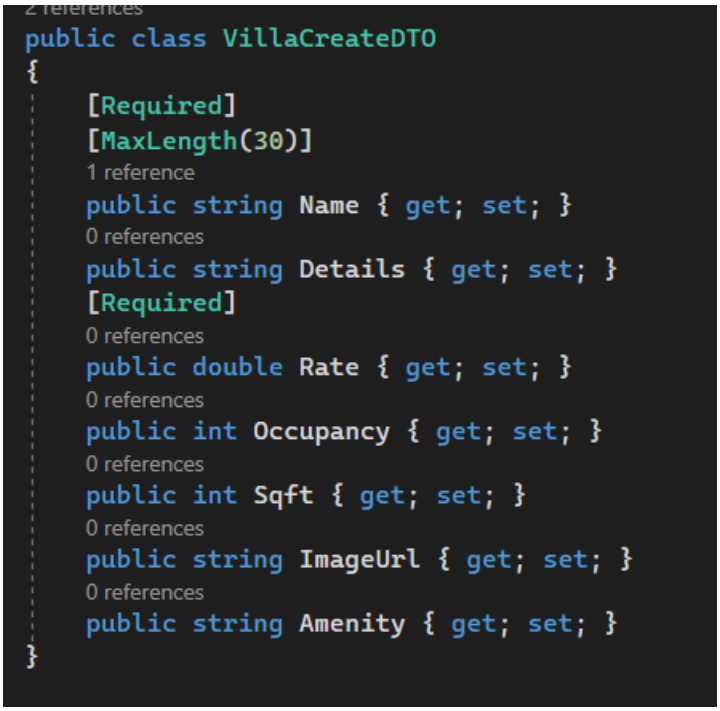

Figure 8 Villa Create DTO

Conclusion

In conclusion, the online villa management system application is a well-designed and implemented web application that utilizes various technologies and design patterns to meet the requirements of a modern e-commerce application. The use of n-tier architecture and repository pattern helped to create a clear separation of concerns, making the application easy to maintain and extend.

The application is built on the Microsoft .NET platform, using ASP.NET Core framework and Entity Framework, which allows for cross-platform compatibility, and a high level of performance and scalability.

The application also utilizes Azure services such as Azure App Service, Azure SQL, and Azure Cosmos DB which allowed for better performance, scalability, and security. The use of Azure Deploy allowed for easy deployment of the application and management of the resources.

The use of .NET Core Identity for authentication and authorization provides a secure mechanism for managing users and roles. The application also utilizes Razor Pages for UI, which is a simple and efficient way to build web pages.

Overall, the application is a great example of how modern technologies and design patterns can be used to build a high-performance, scalable, and secure e-commerce application.

References

1. Pro .NET Design Pattern Framework 4.5, by Steven John Metsker and Matthew B Jones.
2. C# 6 for Programmers (6th Edition), by Paul Deitel and Harvey Deitel.
3. Professional C# 7 and .NET Core 2, by Christian Nagel et al..
4. C# 7 and .NET Core 2.0 – Modern Cross-Platform Development, Third Edition, by Mark J. Price.
5. Pro C# 7: With .NET and .NET Core, Eighth Edition, by Andrew Troelsen
6. CLR via C# (Developer Reference), Fifth Edition, by Jeffrey Richter
7. Programming in C# Exam Ref 70-483: Second Edition, by Wouter de Kort
8. Effective C#: 50 Specific Ways to Improve Your C# (Effective Software. Development Series), Sixth Edition, by Bill Wagner
9. Murach's Beginning Visual C# 2019, Fifth Edition ,by Anne Boehm & Joel Murach
10. Programming Microsoft ASP.NET MVC, Dino Esposito (Microsoft Press)