
University Of Craiova
Faculty Of Sciences
Advanced Techniques for Information Processing
Specialization

Dissertation

Scientific coordinator:

Conf. Univ. Dr. Coşulschi Mirel

Graduate:

Georgescu Ion-Eduard

Craiova

2022

University Of Craiova
Faculty Of Sciences
Advanced Techniques for Information Processing
Specialization

Project and Error Management
Application

Scientific coordinator:

Conf. Univ. Dr. Coşulschi Mirel

Graduate:

Georgescu Ion-Eduard

Craiova

2022

Contents

1	Introduction	3
2	ASP .NET Core	4
2.1	API	5
2.2	Middleware	7
2.3	Model-View-Controller	11
2.4	Routing	13
2.5	NuGet Packages	15
3	Angular	17
3.1	Modules	17
3.2	Angular CLI	19
3.3	Components	20
3.4	Services	23
3.5	Resolver	24
3.6	Angular Routing	25
3.7	TypeScript	27
4	Atlas	29
4.1	Application features	29
4.2	Application architecture	30
4.3	Database design	35
4.4	Application presentation	39
5	Conclusions	47
5.1	Personal contributions	47
5.2	Future improvements	48
	Bibliography	49

Chapter 1: Introduction

The dissertation “Project and Error Management Application” presents the development of the Atlas application which is a web application that allows the management of the projects, errors, requests and tasks, that are designed to be used for organizations, groups of users, and students.

This dissertation highlights the development of the application using the tools offered by ASP .NET Core and the component management benefits offered by the Angular platform.

ASP.NET Core MVC is a web application development platform maintained by Microsoft, which allows the implementation and use of the models, views and controllers.

Angular is a web application development platform that provides developers with a complex set of tools that make it easy to deploy applications across different platforms.

This dissertation is structured in five chapters, as follows:

1. The Introduction chapter represents the introductory part of the paper and the presentation by chapters of its structure.
2. The ASP .NET Core chapter provides details on how to develop APIs and the architecture on which the ASP .NET Core is based and describes the model that contributed to the development of the APIs, named Model-View-Controller.
3. The Angular chapter describes how to develop the user interface. This chapter contains details about the development modules that Angular makes available to developers, these modules being represented by: components, services and routes.
4. The chapter 4 contains the presentation of the application and describes how the concepts presented in the previous chapters have been implemented.
5. The Conclusions chapter contains a summary of the information presented in this paper and a personal opinion on the use of these modes for development of Web applications.

Chapter 2: ASP .NET Core

ASP.NET Core is a web platform created by Microsoft for the development of web applications, APIs, and microservices that uses common models such as MVC and dependency injection¹.

ASP.NET Core is optimized for the development of modern web applications and cloud-based hosting scenarios. The modular design allows applications to depend only on the features they actually use, improving application security and performance while reducing the need for hardware resources.

.NET Core also offers support for multiple platforms, enables the use of microservices and Docker containers, and can improve the stability and the level of performance for applications by lowering the resource requirements.

.NET Core and ASP .NET Core offer more application development advantages than .NET and the main benefits of using .NET Core for server applications are the following [7]:

- ASP.NET Core has a high compilation speed. The .NET code runs much faster than code in interpreted languages such as JavaScript or Ruby. ASP. NET Core is also optimized for multithreading and asynchronous activities.
- The NuGet package manager contains thousands of packages available. For example, there are packages available for JavaScript Object Notation² (JSON) deserialization, database connectors, or PDF generation.
- Manages security to verify data entry and prevent counterfeiting.

ASP.NET Core is optimized for cloud usage because its complexity is very low, has high performance and it allows hosting multiple applications on the same hardware. This advantage decreases the need for more hardware resources and optimizes the use of cloud hosting services "Pay As You Use", a service in which payment is made based on the total time in which the application was active.

¹Dependency injection is a programming technique which allows the use of dependencies of other objects.

²JSON is a syntax used to store data.

A scenario with higher efficiency can be noticed with the ability to serve multiple clients in a given application with the same hardware, further reducing the need to invest in servers and hosting infrastructure. This is the reason why more and more organizations are choosing to host their web applications in the cloud using services such as Microsoft Azure.

ASP.NET Core also fully supports dependency injection, both internally and at the application level. Interfaces can have multiple implementations that can be changed as needed. Dependency injection allows applications to easily interact with those interfaces, which are much simpler than specific implementations, making them easier to extend, maintain and test.

ASP.NET Core provides support for application testing, which facilitates application development and testing using a test server that is created on the local machine and can be used to store applications in the hardware memory. Tests can make requests to this server by running the full stack of the application (including middleware, routing, linking patterns, filters, etc.) and receiving a response in a very short time that is similar to the time obtained on a real server.

2.1 API

An Application-Programming-Interface (API) is an application computing interface that manages interactions between multiple software users. An API defines the types of requests that can be made and how they are resolved. It can also provide a set of extensions so the users can develop existing features in different methods and to varying degrees [9].

Sometimes the term API is used, by extension, to refer to the subset of software entities (code, subcomponents, modules, etc.) that serve to effectively implement the API.

Today, with the rise of Representational State Transfer³ (REST) and Hypertext Transfer Protocol (HTTP) Web services, it is assumed that the term often refers to the APIs of such services when no other context is provided.

In building applications, an API simplifies programming by exposing only the objects or actions that the developers need, while a graphical user interface can provide a button to execute all the steps that perform multiple actions.

An API is usually linked to a software library. The API describes and prescribes

³REST is an architectural style used to create services.

instructions while the library is an implementation of a set of rules. A single API can have multiple implementations in the form of different libraries that share the same programming interface.

An API can also be linked to a software framework. A framework can be based on multiple libraries that implements multiple APIs, but unlike the normal use of an API, access to the integrated behavior of the framework is represented by extending its content to new classes.

Using an API may vary depending on the type of programming language involved. An API for a procedural language, such as Lua, may consist primarily of basic routines for executing code, manipulating data, or handling errors, while an API for an object-oriented language, such as Java, would provide a specification of its classes and class methods.

Separating the API from the client interface by mapping the features and capabilities of a language into an interface allows the use of a library or service written in a different development language. For example, this separation allows the use of ASP. NET Core together with the Angular development platform.

Remote APIs allow developers to manipulate remote resources through specific communication protocols and standards that allow different technologies to work together, regardless of language or platform. For example, the API used to connect to a database allows developers to query many different types of databases with the same set of functions, while the API used to invoke methods allows the invocation of functions that can be performed remotely.

A Web API is a programming interface for an application running on a Web server. Web APIs allows the combination of multiple APIs into new applications known as Mashups. In the social media space, Web APIs have enabled Web communities to facilitate the sharing of content and data between communities and applications. This way, content created in one place can be posted and updated in multiple locations on the web. For example, the Twitter API allows developers to access the Twitter database, and the search API provides ways for developers to interact with data in the database and Twitter trends.

An request to an API is an architectural approach that accesses program interfaces and various sets of application services in order to return the required information to consumers.

In the context of Web application development, an API is usually defined as a set of specifications, such as HTTP request messages, along with the structure definition of

response messages, usually in extensible markup language (XML). or in JSON format.

Designing the security of an API has a significant impact on its use. The principle of hiding information describes the role of programming interfaces that use modular programming to hide the implementation details of modules, so the users do not understand the complexities inside the modules. Thus, designing the security of an API will provide only the tools that a consumer would have expected.

Figure 2.1 shows the accessibility level of a consumer API.

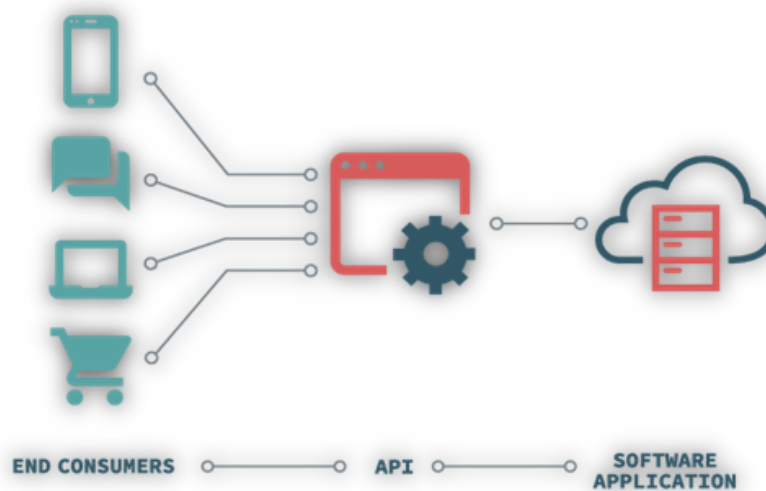


Figure 2.1: The accessibility representation of an API for consumers [1]

2.2 Middleware

Middleware is one of the most important configuration parts for defining how an application behaves and how it responds to user requests.

In ASP.NET Core, middleware is a set of classes written in the C# language that can handle a request or response sent to a server. The main uses for middleware are [9]:

- Manage an HTTP request received by generating an HTTP response.
- Processing a received HTTP request, modifying and forwarding it to another middleware class.
- Processing an HTTP response, modifying and transmitting it to another middleware class or to the ASP.NET Core Web server.

Middleware can be used in a variety of ways in ASP .NET Core applications. For example, a middleware class may write down a user's login information and then pass it on to another middleware class that may use that information.

The most important middleware class in most ASP.NET Core applications is the `MvcMiddleware` class. This class automatically generates all information about the components of an application and maintains their management. One of the most common use of middleware is to resolve the transversal issues of the applications.

Transversal issues are described as the need for one middleware class to call another middleware class to return an answer. These aspects of the application must be managed for each application, regardless of the path specified in the application or the resource requested. These features of middleware include management of the following transversal issues [9]:

- Identification for each request.
- Add standard security for a response.
- Associate a request with the relevant user.
- Setting the language for the current request.

In each of these examples, the middleware will receive a request, modify it and then send the request to the next middleware class. Subsequent middleware can use the details added by the previous middleware to handle the application in a simpler way.

For example, in Figure 2.2, the authentication middleware associates the request with a user.

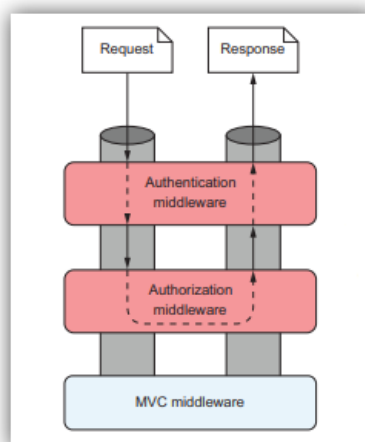


Figure 2.2: The return of a response through the middleware classes [9]

Authorization middleware uses this detail to check if the user has permission to make that application-specific request or not.

If the user has permission, the authorization software middleware will send the request to MvcMiddleware to allow it to generate a response. If the user does not have permission, the authorization middleware may reject the request, generating a direct response. It returns the response to the previous middleware before being sent to MvcMiddleware.

The request is sent in different classes until a middleware class generates a response, at which point the response goes back through the previous classes for the second time, until it returns to the first middleware class. Finally, the last middleware class will send the response back to the ASP.NET Core Web server.

When a request is made, the ASP.NET Core Web server builds an object called HttpContext, which the application uses as a storage space for a single request. The subsequent answer and anything specific to this particular request can be stored in this object.

This could include application properties, application-specific services, data that has been uploaded, or errors that have occurred. The web server will populate the initial HttpContext with details about the initial HTTP request and other configuration details that will be passed on to the rest of the application.

All intermediate programs have access to HttpContext to make a request. For example, this object can be used to determine if the request contains details about the user or which page it tried to access. These details can be used to determine how the application is handled.

Once the application has finished processing the request, it will update the HttpContext with an appropriate response and return it via middleware to the web server. The ASP.NET Core Web server will then turn the representation into an HTTP response that will be sent to the user's browser.

HTTP handler modules and manipulators are also available. An HTTP handler is a process that runs in response to a request and generates a response. HTTP modules handle the transversal issues, such as security, logging or session management.

Logging, error handling, and security are classic concerns that are all required by many different parts of an application. Middleware handles requests, errors, and security for an application. Errors can be managed in a number of different ways, middleware deals with error management by offering different methods of error interpretation.

These are examples of methods for interpreting errors [9]:

- *DeveloperExceptionPageMiddleware* - provides a quick error response with application build details.
- *ExceptionHandlerMiddleware* - provides a generic error page that is easy to use in application development.
- *StatusCodePagesMiddleware* - turns status codes into user-understandable error pages.

By using these methods for interpreting middleware-managed errors, any errors will not expose application security details and will not damage the application. Below is the code sequence for configuring a middleware model.

```
1 public IConfiguration Configuration { get; }
2 public void ConfigureServices(IServiceCollection services){
3     services.AddControllersWithViews();
4 }
5 public void Configure(IApplicationBuilder app, IWebHostEnvironment env){
6     if (env.IsDevelopment()) {
7         app.UseDeveloperExceptionPage();
8     }
9     else{
10        app.UseExceptionHandler("/Home/Error");
11        app.UseHsts();
12    }
13    app.UseHttpsRedirection();
14    app.UseStaticFiles();
15    app.UseRouting();
16    app.UseAuthorization();
17    app.UseEndpoints(endpoints =>{
18        endpoints.MapControllerRoute(
19            name: "default",
20            pattern: "{controller=Home}/{action=Index}/{id?}");
21    });
22 }
```

2.3 Model-View-Controller

ASP.NET Core MVC provides the tools needed to build APIs and Web applications. The term MVC is an acronym for "Model-View-Controller" and is a user interface model that divides the responsibilities of responding to user requests into several parts [9]. In addition to complying with this model, ASP.NET Core MVC allows the implementation of functions in ASP.NET Core applications such as Razor pages.

The MVC model separates the various aspects of the application (input logic, business logic, and UI logic) while providing a free coupling between these elements. The model specifies where the logic should be drawn from the application.

Razor pages are embedded in ASP.NET Core MVC and use the same features for model routing and binding. However, instead of having separate folders and files for controllers, views and templates, Razor pages are placed in a single "/Pages" folder, and their path is based on their relative location in that folder.

ASP.NET Core MVC provides dynamic website development based on templates that provide complete control over markup and use the latest Web standards.

The MVC architecture model separates an application into three main groups of components: models, views and controllers. This model helps to separate tasks and actions.

This separation helps manage the complexity of developing an application, as it allows the implementation of an application sequence without affecting the code of another sequence. For example, a developer can work on the views code without depending on the code of the models and controllers.

Using this interface model, user requests are directed to a controller who is responsible for collaborating with a model to perform actions requested by the user and to retrieve query results. The controller chooses a View to be displayed to the user and gives him all the model data he requests.

Views are responsible for presenting content through the user interface. They use the Razor view engine to embed .NET Core code in HTML code.

The following code sequence shows an example model in ASP .NET Core.

```
1 namespace Paste.Models
2 {
3     public class UserAccount : IdentityUser
```

```
4      {
5          [Column(TypeName = "nvarchar(30)")]
6          public string FirstName { get; set; }
7
8          [Column(TypeName = "nvarchar(30)")]
9          public string LastName { get; set; }
10
11         [Column(TypeName = "nvarchar(30)")]
12         public string VerificationCode { get; set; }
13
14         [Column(TypeName = "smalldatetime")]
15         public DateTime ExpCode { get; set; }
16
17         public virtual ICollection<File> File { get; set; }
18
19         public virtual ICollection<Folder> Folder { get; set; }
20     }
21 }
```

The content in the views is obtained from the models connected to them and any source code in them refers to the presentation of the content. If a complex view from a complex model is required, a `ComponentView`, `ViewModel`, or a view template will be used to simplify the view.

An action is a method that runs in response to a request. A controller is a class that contains a series of action type methods that can be logically grouped [5].

Controllers are the components that interact with users, work with models and finally select a view in which to present it. In an MVC application, the views only display information, the controller handles and responds to user requests and interaction.

In the MVC model, the controller is the initial entry point and is responsible for selecting the types of models to work with and the view to display (from here it controls how the application responds to a request given by the user).

Each request received is compared to the application routing table, and if a suitable route is found, the associated method (belonging to a controller) is used to handle the request. If no matching route is found, an error handler is called (in this case, the method will return a `NotFound` error).

Figure 2.3 shows the management of the MVC system in ASP .NET Core.

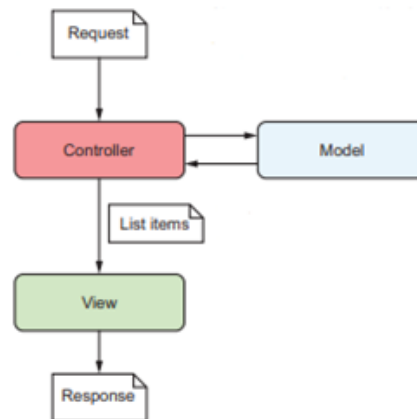


Figure 2.3: The management of a Model-View-Controller system [9]

2.4 Routing

A crucial part of the MVC model in ASP.NET Core is selecting the right method of an action to invoke it and get a response to a given request, this process is called routing.

Routing in ASP.NET Core is the process of mapping an HTTP request to a specific action. Routing is responsible for matching received HTTP requests and sending queries to the application's executable endpoints. The reference points are the application units with the application management executable code. The endpoint matching process can extract values from the request URL and provide those values for request processing [9].

MvcMiddleware can be used to manage more complex application logic. This can handle an appropriate request by invoking a method, known as an action, and using the result to generate a response.

By using MvcMiddleware in a web application, setting up a router makes a map of any HTTP requests to the MVC route handler. It takes a URL and deconstructs it to determine which controller and which action is appropriate.

Figure 2.4 shows how an application is handled by MvcMiddleware.

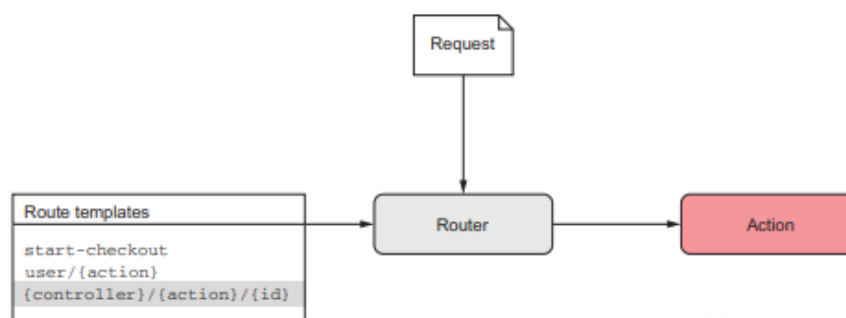


Figure 2.4: The management of a request made for the MvcMiddleware [9]

Each request is sent to the MVC router, which inspects the URL of the request after which the router selects the action to take, based on the received URL and application configuration.

One of the benefits of using a routing system is that it links actions to URLs that are used to execute those methods. This allows for changes to be made to the application URLs by managing the routing system.

Figure 2.5 shows a simple management of the routing system.

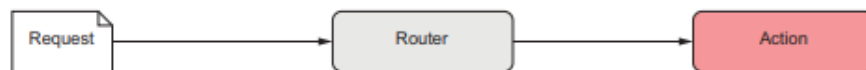


Figure 2.5: The management of a simple routing system [9]

Using URL generation will avoid misdirecting routes for a method. As an example, for the "api/View/3" route, the appropriate action will be selected from the View controller, with parameter 3. If this path is changed to "api/File/3", if the File controller exists, the corresponding action will be performed, if the controller does not exist, the 404 response will be returned, stating that the action was not found.

Many different routing patterns can be defined, each of which can set a different number of URLs that map a variety of actions. Multiple routes can also be defined, all of which indicate the same action from different URLs. Alternatively, a single route can be defined that matches only one URL and matches a single specific action.

The following code sequence shows an example for routing an action.

```
1 [HttpGet]
2 [Route("Content")]
3 public string GetFileContent(int fileId){
4     return _context.File.Where(t => t.FileId == fileId).ToString();
5 }
```

Figure 2.6 shows the entire ASP .NET Core routing system.

When the server receives a request, the URL is split into segments and compared to the first defined route. The controller and action are extracted from the route based on the URL. If a route does not match, it will be checked if there are other routes available and if there are, each route will be checked. If there is no valid route for the request received by the server, then the server will return a 404 response, which means that the action was not found.

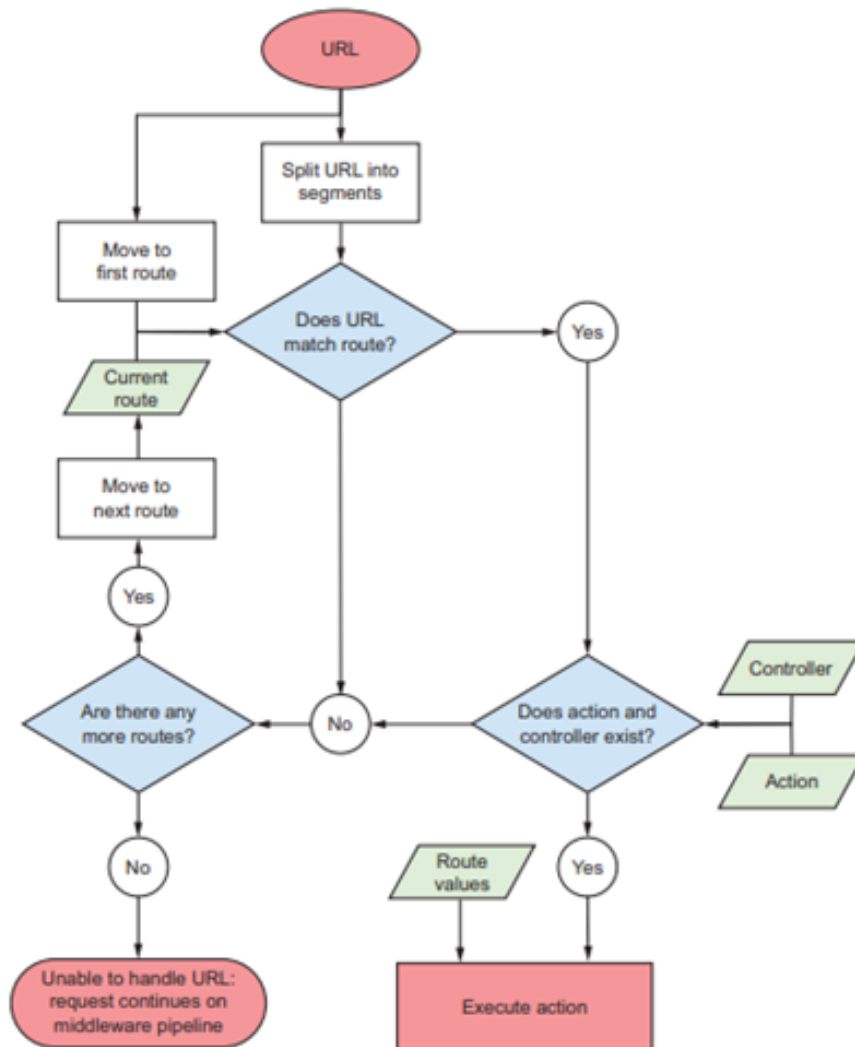


Figure 2.6: The ASP.NET Core routing system [9]

If a valid path is found for the request received by the server, the action is performed using the values extracted from the URL.

2.5 NuGet Packages

NuGet is a package manager designed for the ASP .NET Core development platform, formerly known as NuPack. Since its introduction, NuGet has evolved into a larger system of tools and services [9].

An essential tool for modern development platforms is a mechanism by which developers can share and use code sequences created by other developers. For .NET Core, the code-sharing mechanism is NuGet, which defines how .NET packets are created, hosted, and used.

Developers create packages and store them on a public or private server, consumers

get those packages, add them to their personal projects and then call the functionality of a package in the project.

A compatible package presents assemblies built for at least the .NET development platform. Developers can create platform-specific packages, and to maximize the compatibility of a package, developers target the .NET Standard, which is compatible with .NET and .NET Core projects [5]. This is a method by which both creators and consumers can use code sequences that work for the .NET and .NET Core development platforms.

These are the main benefits of using NuGet packages [5]:

- Provides the nuget.org central repository with support for public and private hosting of packages.
- Provides the tools that developers need to create, publish, and use packages.
- Maintains a reference list of packages used in a project and provides the ability to restore and update packages from that list.

Chapter 3: Angular

Angular is a modern web application platform sponsored and maintained by Google that provides developers with a comprehensive set of tools and capabilities to build large and complex applications.

Angular builds on some of the best server-level development models and uses them to improve the HTML language in the browser that makes it possible to create applications that work for almost any platform (mobile or desktop).

Angular applications are developed using the Model-View-Controller model, creating a set of tools that make developing applications simpler and easier.

Angular has several types of higher level entities. These entities have different roles and specific functions used in creating applications.

These are some of the entities used to develop Angular applications [8]:

- *Modules* - Objects that help to organize dependencies in discrete units.
- *Components* - Elements that will make up most of the structure and logic of an Angular application.
- *Directives* - Objects that change the appearance or behavior of an element, component, or other directives to give them new capabilities.
- *Pipelines* - Functions that format data before it is displayed.
- *Services* - Reusable objects that provide access to data or auxiliary functions.

3.1 Modules

Modules are files created to store related entities for easy reuse and distribution. Angular itself is made up of several modules and any external library that is used contain modules. There are two types of modules, JavaScript modules and Angular modules.

JavaScript modules are language constructs and are a way to separate code into different files that can be uploaded as needed.

Angular modules are logical constructions used to organize similar groups of entities (such as all the things needed to manage routes) and are used by Angular to understand what needs to be loaded as well as what dependencies exist. There should always be a main application module, but there are also additional modules built for Angular applications [10].

Figure 3.1 shows examples of Angular modules and their use.

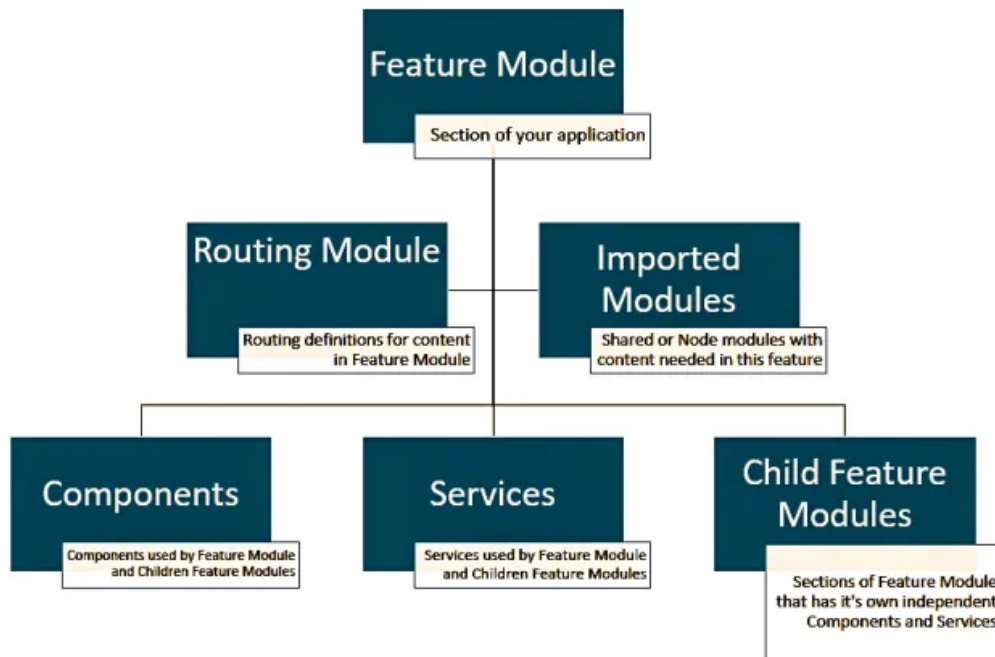


Figure 3.1: Examples of Angular modules [2]

A module is declared by creating a class and declaring it with the `@NgModule` decorator.

The Angular application's main module (AppModule) is generated by the CLI and provides key information for Angular so that it can understand how to display and load the application.

The declaration module contains a list of all the components and directives that the main application module wants to make available to the entire application. The service module also contains a list of all the services made available to the entire application.

The import module contains a list of the other modules on which this module depends. If there are problems with another module not loading, this is the first place to check if it is registered in the Angular application.

To start an application, Angular needs to know which component(s) to display on the screen and look at the component table for this list.

The following code sequence shows an example of using the modules.

```
1 @NgModule({
2   declarations: [AppComponent,UserComponent,],
3   imports: [BrowserModule],
4   providers: [UserService{
5     provide: HTTP_INTERCEPTORS,useClass: AuthInterceptor,
6     useValue: (route: ActivatedRouteSnapshot, state: RouterStateSnapshot) => {},
7     multi: true}],
8 bootstrap: [AppComponent]
9 })
10 export class AppModule { }
```

3.2 Angular CLI

Modern application development usually requires the configuration of many tools to start a project. Angular CLI is a command line interface tool used to initialize, develop, and maintain Angular applications. The tool can be used directly in a command center or indirectly through an interactive user interface, such as Angular Console.

CLI has a number of features that help develop Angular applications. The main features are [10]:

- Generate templates for creating new applications - If it's needed to create a new application using the files of an existing application or create all the files for a new application, CLI will generate a complete application preset as a basic application.
- Generate new parts of the application - It can generate components, services and routes required for the application and will automatically ensure that they are fully connected in the construction process.
- Managing the entire set of build tools - Files need to be processed before being presented to the customer, CLI will process the source files and build them in a version optimized for development and production.
- Hosting a development server - CLI manages the compilation of applications and then starts a development server that listens for requests to localhost with a real-time reloading feature.

Examples of command lines for Angular CLI [8]:

- `npm install -g @angular/cli` - Command line used to install and configure Angular CLI.
- `ng new project-name` - Command line that will create a folder named "project-name" in the project directory. In this folder, the CLI will generate all the components and services needed to develop a new application.
- `ng serve --o` - Command line used to compile the application and present it locally to the user. The server will recompile the application and reload the page when changes are found in any of the files.
- `ng generate` - Command line used for automatic generation of components, routes, services. This command line will automatically manage and configure the newly created files.
- `ng config` - Command line that allows the configurations of files in the Angular application. This command line is used to make changes to the `angular.json` file, which contains the current configuration of the Angular application.
- `ng build` - Command line that builds the current version of the application and will store the files generated after construction in the `dist` directory.
- `ng test` - A command line that will run a test for the application after an initial version of the application has been built. This command line will frequently check for changes to files and will run tests each time a new version of the application is saved.
- `ng update` - Command line that manages the version of the application according to a version selected by the user.
- `ng doc [search]` - A command line that displays the official Angular documentation for keywords given by the user.

3.3 Components

Angular has taken a component-based approach to application development. The goal is to design each component of the application in a standalone manner that limits duplication in different parts of the program.

A component is a way to create custom HTML elements and use a tree of adjacent components to easily make Angular applications [10].

Components are a subset of directives, always associated with a template. Unlike other directives, a single component can be initiated for an element in a template. For a component to be available to another component or application, it must belong to an NgModule. For a component to belong to an NgModule, it must be listed in the NgModule metadata declaration field. In addition to these options, the running behavior of a component can be changed by implementing events.

HTML itself is a language of components. Each element has a certain role and functionality to create more complex functionalities.

The components have several concepts that guide their design and architecture, these are [8]:

- *Encapsulation* - Searching for component logic in one place.
- *Isolation* - Keep internal components hidden from external files.
- *Reusability* - Allow reuse of components with minimal effort.
- *Customizable* - Allows the use of styles to customize the component.
- *Declarative* - Use a component with a simple declaration.
- *Event* - Perform events while using the component.

Applications are essentially combinations of components. These components are based on the basic principles of encapsulation, isolation and reuse, which contain events, customizable styles, and can be declarative.

HTML is a web language that describes the content of a page in a fairly concise set of elements. As a markup language, it is a declarative way of describing content. Custom Elements means we can expand HTML with our own additional elements, adding as much vocabulary as possible.

The official specification for custom element components is intended to allow developers to create new HTML elements that essentially blend naturally and natively into the Document Object Model¹ (DOM). In other words, using a component with custom elements is no different than using other HTML elements.

¹DOM is a programming interface for HTML and XML documents.

Customizable elements give us a declarative way to create a reusable component, which encapsulates the mechanics of the component outside the rest of the application, but can issue events that allow other components to connect to the elements of other components.

The display of a component can be done with an accompanying template. A template is a form of HTML that defines to Angular how the component is displayed.

Templates are usually arranged hierarchically, allowing editing, displaying, and hiding sections or UI pages. The template immediately associated with a component defines the view of that component. The component can also define a view hierarchy, which contains embedded views, hosted by other components.

A view hierarchy may include views of components in the same NgModule, but may also include views of components that are defined in different NgModules.

Figure 3.2 shows a hierarchical organization of the components and templates that are used in the development of Angular Web applications.

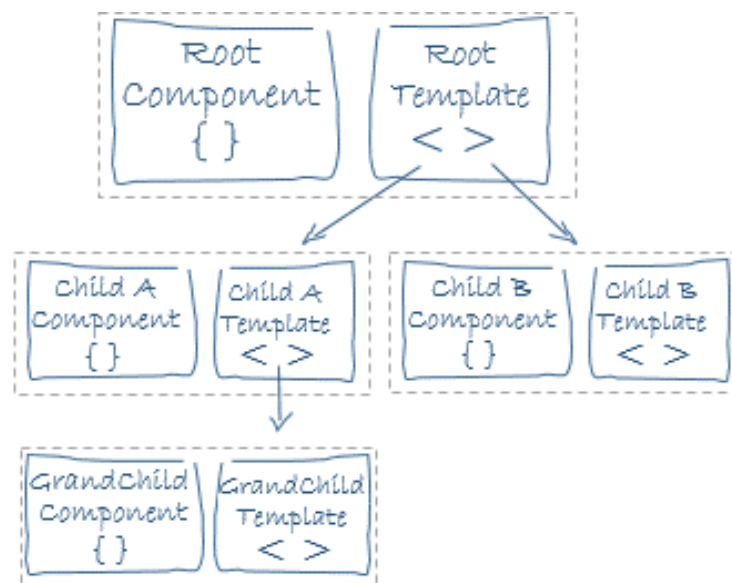


Figure 3.2: The hierarchical organization of the components and templates [4]

A template looks like regular HTML, except that it also contains the Angular syntax of the template, which changes the HTML depending on the application logic, component events, and application data status. A template can use data binding to coordinate application data to transform data before it is displayed.

An example of an Angular template is shown in the following code sequence.

```

1 <div class="box-1 text-center mb-2 mt-2">
2   <div class="btn btn-one" style="width: 200px;"
3     (click)="this.navigateToNew()">
4     <span>NEW FILE</span>

```

```
5     </div>
6 </div>
7 <div class="box-1 text-center mb-2 mt-2">
8     <div class="btn btn-one" (click)="this.navigateToTest()"
9     style="width: 200px;">
10         <span>NAVIGATE</span>
11     </div>
12 </div>
13 <div class="box-1 text-center mb-2 mt-2">
14     <div class="btn btn-one" (click)="this.service.onLogout()"
15     style="width: 200px;">
16         <span>LOG OUT</span>
17     </div>
18 </div>
```

3.4 Services

Services are objects that abstracts code logic for the purpose of reusing the code in multiple places. Using ES2015² modules (ECMAScript 2015), service classes are exported so that any component can import and use them as needed. It could also have functions or even static values, such as a string or a number, with services being a way to share data between different parts of the application.

Another way of presenting services is by an object that contains the same type of data or functions, which any part of the application can import, as needed. Although services often help with data management, they are not limited to a specific location.

The intent of a service is to allow code reuse. A service can be a set of common methods that can be shared. A perfect scenario for using a service is to manage the data and divide it into different components.

Angular distinguishes service components to increase modularity and reuse. By separating the functionality related to visualization of a component from other types of processing, component's classes can be made to work more efficiently with services.

A component may delegate certain service tasks, such as retrieving data from the server, validating user input, or logging in directly to the console. By defining such processing activities in a class of injectable services, these tasks can be made available to any

²ES2015 is a language that manages the scenarios of a web interface.

component. Angular applications can also be made more adaptable by injecting different providers of the same type of service, as appropriate, and under different circumstances [10]. Dependency injection is used to provide new components with services or other dependencies they need.

Components consume services, meaning that a service can be injected into a component, giving the component access to that class of services.

To define a class as a service in Angular, the `@Injectable()` decorator is used to provide metadata that allows Angular to use the class in a component as a dependency.

The services are also ES6 modules (ECMAScript 6) and are designed for portability. Any ES6 class could be used, even if it is not specific to Angular.

CLI provides an easy way to create a service for your application. CLI also handles file management and connecting routes for the new service.

This command is an example of how to generate a service:

```
1 ng generate service services/stock
```

The command will generate the `stock.service.ts` file in the "src/app/services" directory, and the class that represents the file is called `StockService`, which is exported for use of its data and functions in other components. An example of a new service is presented in the following code sequence.

```
1 import { Injectable } from '@angular/core';  
2  
3 @Injectable({  
4   providedIn: 'root'  
5 })  
6 export class StockService {  
7   constructor() { }}
```

To use data and functions in the `StockService` class, it must be imported into the component in which we want to use it. The import can be done with the following line of code:

```
1 import { StockService } from 'src/app/shared/stock.service';
```

3.5 Resolver

An Angular application can make a request for data to an API server, but the response may be delayed, which may prevent the application from displaying the content of the

response, if it did not arrive on time [10]. For Angular applications, this issue can also occur if the contents of the templates are loaded and displayed to the user before receiving the response from the API server.

The resolver is an interface that classes can embed to be a data provider. A data provider class can be used with the route module to resolve requirements to the API server while browsing. The interface defines a solution method that will be invoked at the beginning of the navigation. The route module will then wait for the data to be resolved before the route is finally activated.

In conclusion, it is desired to delay the display of the routed component until all the necessary data has been retrieved.

The following code sequence shows an example of a resolver.

```
1 export class FileResolver implements Resolve<any> {
2     readonly BaseURL = 'http://localhost:53357/api';
3     local = 'http://localhost:4200/';
4     formData: FileModel;
5
6     constructor(
7         private http: HttpClient,
8     ) {}
9     resolve(route: ActivatedRouteSnapshot) {
10         return this.http.get(this.BaseURL + '/File/' + route.params['file'])
11             .pipe(tap(res => {
12                 this.formData = res as FileModel;
13             }));
14     }
```

3.6 Angular Routing

In an application, it is possible to change the content that the user sees by showing or hiding portions of the templates that correspond to certain components. As users perform tasks in the application, they will be redirected to different templates that have been defined.

A very important part of Angular applications is setting the routes that configure the different pages that the application can display [3]. Most applications need some form of routing so that they can display the correct part of the application on time. Angular has a router that works well for Angular architecture by mapping components to routes. The

router allows navigation by interpreting a browser URL as an instruction to change the template.

The router works by declaring a selector and a URL to the intended template, and the contents of that template will be displayed when the selector is called in a template.

In older versions of the CLI, automatic route configuration is not supported, and manual configuration is required in this case. The manual configuration is done by creating a new file in the "src/app/app.routes.ts" directory.

The main purpose of this file is to configure the routes for the application and we start by importing the RouterModule and the Route definition. RouterModule is used to enable the router and supports route configuration when initialized. This file also imports the components that need to have routes, so the components can be properly referenced in the route configuration. Routes are defined as a series of objects that have at least one property, usually a URL path and a component.

Here is the code for an route created as an example to configure routes:

```
1 { path: 'stock', component: StockComponent }
```

This path provides a management URL path (which will be `http://localhost:4200/stock`) and links to the `StockComponent`. This is the most commonly used type of routing that is done with Angular, although there are many ways to configure routes.

The following code sequence shows the route management.

```
1 import { NgModule } from '@angular/core';
2 import { Routes, RouterModule } from '@angular/router';
3 import { StockComponent } from './stock/stock.component';
4
5 const routes: Routes = [
6   {path: 'stock' , component: StockComponent}];
7
8 @NgModule({
9   imports: [RouterModule.forRoot(routes)],
10  exports: [RouterModule]
11 })
12 export class AppRoutingModule { }
```

After managing routes, the application must identify the newly created files. The identification of newly created files must be configured in the `app.module.ts` file, where

the component's classes and route classes created must be imported and declared.

The following code sequence shows the import and declaration of route classes.

```
1 @NgModule ({
2   declarations: [AppComponent, StockComponent],
3   imports: [BrowserModule, AppRoutingModule],
4   providers: [],
5   bootstrap: [AppComponent]
6 })
7 export class AppModule { }
```

3.7 TypeScript

Angular is a modern platform built entirely in TypeScript, so using TypeScript with Angular provides a perfect experience.

TypeScript is an OOP-based programming language created and maintained by Microsoft. It is a JavaScript superset that helps to write high quality code without too much complexity and provides simple application development on the part of client technology using OOP-based programming language.

Provides object-oriented programming functionalities such as classes, interfaces, inheritance, encapsulation, etc. Typescript compiles and creates simple Javascript code which is an advanced and easy way to write Javascript.

Because Typescript is a JavaScript superset, existing JavaScript applications are also valid TypeScript applications. It can be used to develop both client-side and server-side JavaScript applications.

The main advantage of TypeScript is that it can impose restrictions on the value of the variables held [6]. For example, a variable can contain only one number or it can contain a string, but the type of variables is not mentioned so any type of value can be stored in any variable. This feature has given rise to many types of comparison operators, such as "==" for free equality or "===" for strict equality.

Here are some examples of how variables are declared and initialized:

```
1 var bill = 20;
2 var bill: number = 20;
```

These are the main advantages of using TypeScript [10]:

- Add clarity to the code - Variable types are easier to understand because application developers don't have to think much about what type of variable they should use.
- Smarter Editor Support - Using TypeScript with a supported editor provides automatic IntelliSense support for written code. During code writing, the editor suggests known variables or functions and can tell what kind of value is expected.
- Detecting errors before running code - TypeScript will detect syntax errors before running code in the browser, helping to reduce the feedback loop when invalid code is detected.

Chapter 4: Atlas

This chapter describes the web application called Atlas, which was created to make it easier to manage projects for an organization or a group of students by allowing the creation and management of tasks, requests and bugs.

Each file created has a number of metadata associated with it, such as name, description, summary, steps to reproduce, severity, probability, reporter and assignee, all of which provide advanced management of files and their contents.

The application provides more features to users with higher access and the access of a specific user is determined from their rank in the organization.

This application was developed using the ASP .NET Core MVC and Angular development environments, which are development environments based on the structure of the Visual Studio environment and provide customized support for Web application development using the MVC development mode of the API and the component management of Angular.

4.1 Application features

Following the presentation of this paper, the term "file" will be used to refer to tasks, requests or bugs that were created in the database.

The application can be used by a visitor or a user who has a personal account in the application.

A visitor is a person who is not logged in and due to the application security standards, the visitor can only create a personal account in the application.

A user who has an account created in the application, in addition to what any visitor can do, the user can also:

- Log in with the personal details saved in the database.
- View all the files created by the assigned project.
- Add, edit, and delete files or projects.
- Assign users to a specific project.

- View recent changes made to existing files.
- Add comments to existing files.
- Manage personal account details.
- Change application settings for the logged in user.

The features that a user has access to are represented in the following Use Case diagram.

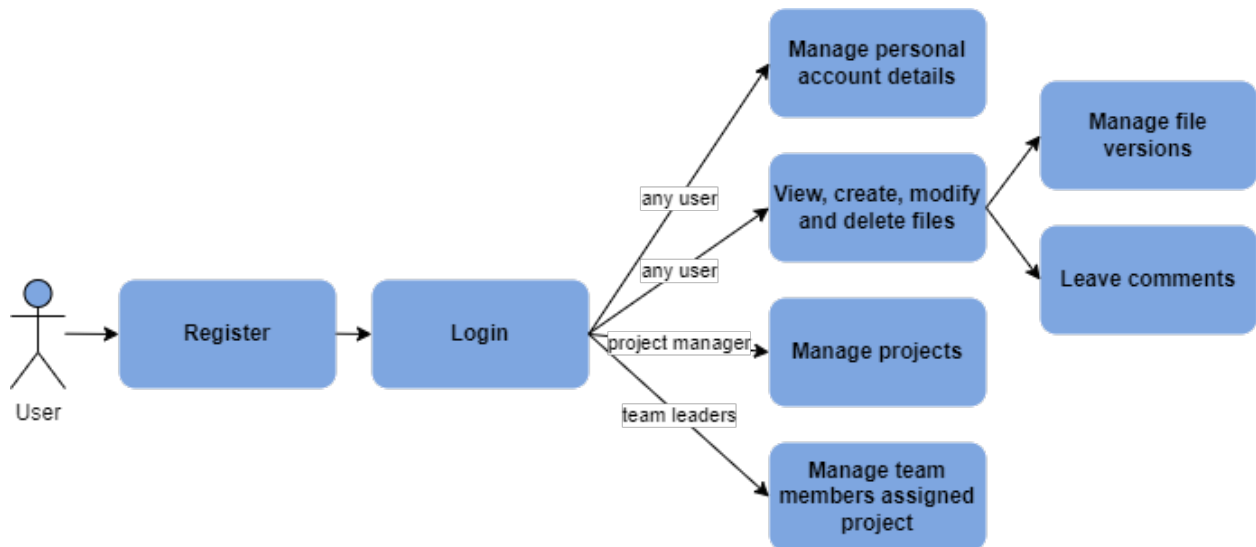


Figure 4.1: Use Case diagram

4.2 Application architecture

ASP .NET Core is used to implement the API of the application by using the tools and classes provided by the MVC model and the NuGet packages provided by .NET Core.

The ASP .NET Core classes used to develop the API are grouped into packages, which are:

- **Controllers** - This package contains all the controller classes that handle the requests received by the server.
- **Models** - It is a package dedicated to the Business type classes of the application, which contains the entities to be saved in the database.
- **Startup** - It is a dedicated package for configuring the services required by the application.

- **Route** - It is a package that allows the management and manipulation of routes in the application.
- **Configuration** - This package manages the processing of configuration information from different file types (JSON, XML, etc.).
- **Entity** - This package contains the interfaces that handle the operations of the application database.
- **MailKit** - It is a package that allows the creation and sending emails to users.
- **JSONWebToken** - This package is dedicated to managing the user authentication system.

Figure 4.2 shows the API structure of the Atlas application.

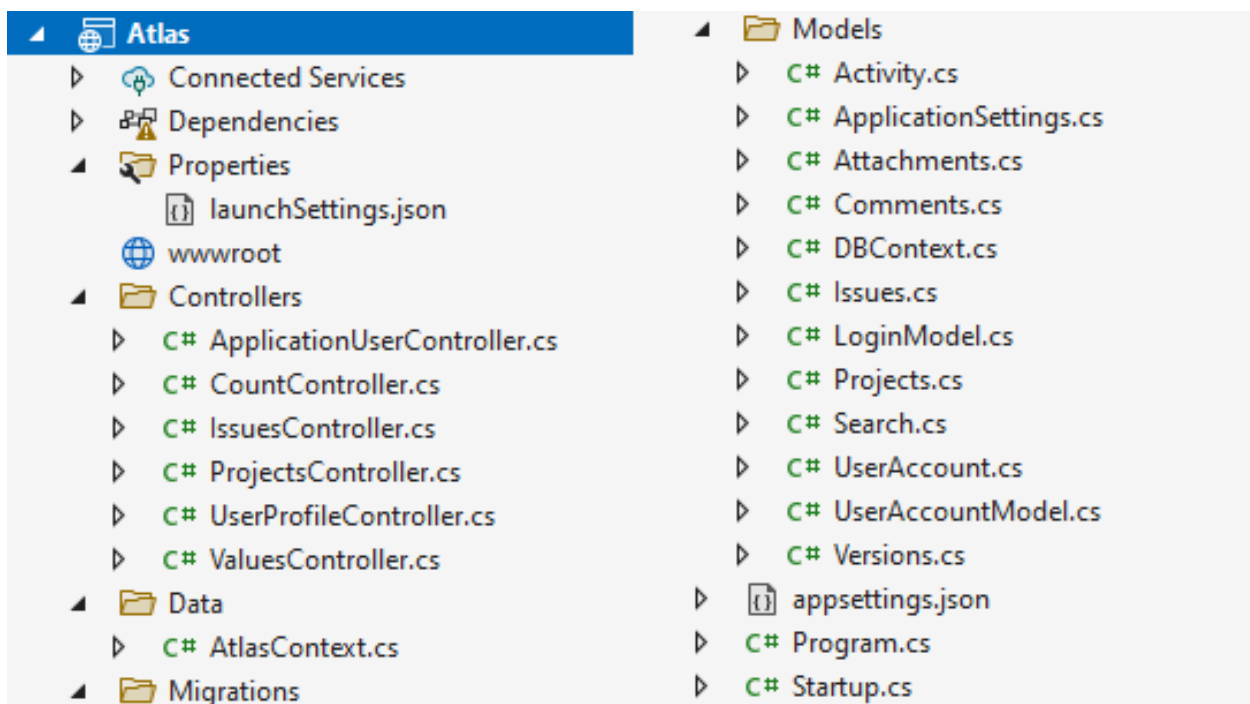


Figure 4.2: Atlas API structure

The most important part of implementing an API is managing the models, these models are available using the MVC package. These are used to collect and store entities sent by the user to be processed by controllers and to return a response or save them in a database. Models are also used to configure database tables.

The following code sequence shows the model needed to create a new account for a user.


```
1 public class UserAccountModel{
2     [MinLength(6)]
3     [RegularExpression("^[a-zA-Z][a-zA-Z0-9._]*[a-zA-Z0-9]$")]
4     public string UserName { get; set; }
5     [MinLength(2)]
6     public string FirstName { get; set; }
7     [MinLength(2)]
8     public string LastName { get; set; }
9     [EmailAddress]
10    [RegularExpression("^[a-zA-Z][a-zA-Z0-9._@]*[a-zA-Z]$")]
11    public string Email { get; set; }
12    [MinLength(10)]
13    [MaxLength(16)]
14    [RegularExpression("[0-9]*$")]
15    public string PhoneNumber { get; set; }
16    [MinLength(6)]
17    public string Password { get; set; }
18    public string AssignedProject { get; set; }
19    [MinLength(2)]
20    public string Rank { get; set; }
21    [MinLength(2)]
22    public string Access { get; set; }
23 }
```

Each variable in the model will have the role of collecting data sent by the user through the controllers and will represent a column for a table of the database. For example, a column will be created in the `AspNetUsers` table with the variable name for the `FirstName` string variable.

The following annotations may be used to establish restrictions on the content of each variable:

- **Required** - Used to determine the need to enter the variable.
- **MinLength** - Used to set the minimum number of characters that the variable must contain.
- **MaxLength** - Used to set the maximum number of characters that the variable should contain.
- **EmailAddress** - Used to apply restrictions to variables that store email addresses.

- **RegularExpression** - Used to set customizable restrictions.

The authentication process is developed using the JWT package. This package allows a user to log in the application using an authentication token called a Bearer Token.

The authentication token is a collection of encrypted data that forms an authentication session. By using this authentication token, it is permitted to maintain the user's authentication session for a number of days that can be set by the developer.

In order for this information to be used, the authentication token must be decrypted using specific algorithms from the JWT package. This authentication token is generated for each user when they want to log in using their personal account.

The following code sequence shows the authentication method using JWT.

```
1 [HttpPost]
2 [Route("Login")]
3
4 //api/ApplicationUser/Login
5 public async Task<IActionResult> Login(LoginModel model){
6     var user = await _userManager.FindByNameAsync(model.UserName);
7     if (user != null && await _userManager.CheckPasswordAsync(user,
8         model.Password)){
9
10         var tokenDescriptor = new SecurityTokenDescriptor{
11             Subject = new ClaimsIdentity(new Claim[] {
12                 new Claim("UserID", user.Id.ToString()),
13                 new Claim("UserName", user.UserName.ToString()),
14                 new Claim("FirstName", user.FirstName.ToString()),
15                 new Claim("LastName", user.LastName.ToString()),
16                 new Claim("AssignedProject", user.AssignedProject
17                     .ToString()),
18                 new Claim("Rank", user.Rank.ToString()),
19                 new Claim("Access", user.Access.ToString())
20             }},
21             Expires = DateTime.UtcNow.AddHours(8),
22             SigningCredentials = new SigningCredentials(
23                 new SymmetricSecurityKey(Encoding.UTF8
24                     .GetBytes(_appSettings.JWT_Secret)),
25                 SecurityAlgorithms.HmacSha256Signature)
26         };
```

```
27     var tokenHandler = new JwtSecurityTokenHandler();
28     var securityToken = tokenHandler.CreateToken(tokenDescriptor);
29     var token = tokenHandler.WriteToken(securityToken);
30     return Ok(new { token });
31 }
32 else{
33     return BadRequest(
34         new { message = "Username or password is incorrect." });
35 }
36 }
```

In the Atlas application, the functions are represented by REST services. These services allow easy manipulation of the resources available in the database.

The request made by the user to the API mentions the path in which the function that processes the data for the user is located.

For example, to display the list of files that the user has, the API will process and send the data to the files when prompted for the "/api/Issues" path using the Get method. This route will perform the function to send the user all the files registered for a specific project in the database. Through these services, the response to the customer can be sent in JSON format.

The following code sequence shows the structure of a JSON response.

```
1 GET /api/Issues?skip=0&take=10
2 HOST: localhost:1531
3 X-Tableau-Auth: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
4 Content-Type: application/json
5 [{
6     "IssueID": "HAL-BUG-1",
7     "Project": "HAL",
8     "IssueType": "BUG",
9     "IssueSummary": "bug_summary",
10    "IssueDescription": "bug_description",
11    "IssueStepsToRepro": "bug_steps_to_reproduce",
12    "IssueSeverity": "1",
13    "IssueProbability": "2",
14    "IssueCreatedDate": "2022-06-20T11:57:00",
15    "IssueModifiedDate": "2022-06-20T08:55:00",
16    "IssueFixVersion": "Build 242",
17    "IssueReporter": "Georgescu Eduard",
```

```
18     "IssueAssignee": "Marian Theodor",  
19     "UserId": null,  
20     "UserAccount": null}  
21  }]
```

Sending a very large set of files can consume a lot of resources for a browser, which can lead to a longer wait for all files to be loaded. This problem can be solved by implementing the management system known as Pagination. Pagination can handle the number of files to be sent to the user and the files to be sent. The following code sequence describes the function for sending files to the user.

```
1  // GET: api/File  
2  [HttpGet]  
3  
4  public async Task<ActionResult<IEnumerable<Issues>>> GetIssuesDetail(  
5      int skip, int take){  
6  
7      string userId = User.Claims.First(c => c.Type == "UserID").Value;  
8      return await _context.Issues  
9          .OrderBy(t => t.IssueSummary).Skip(skip).Take(take).Select(  
10         t => new Issues() {  
11             IssueID = t.IssueID,  
12             IssueType = t.IssueType,  
13             IssueSummary = t.IssueSummary,  
14             IssueDescription = t.IssueDescription,  
15             IssueStepsToRepro = t.IssueStepsToRepro,  
16             IssueSeverity = t.IssueSeverity,  
17             IssueProbability = t.IssueProbability,  
18             IssueCreatedDate = t.IssueCreatedDate,  
19             IssueModifiedDate = t.IssueModifiedDate,  
20             IssueFixVersion = t.IssueFixVersion,  
21             IssueReporter = t.IssueReporter,  
22             IssueAssignee = t.IssueAssignee  
23         }).ToListAsync();  
24     }
```

4.3 Database design

The database diagram and the relationships between the tables are shown in Figure 4.3.

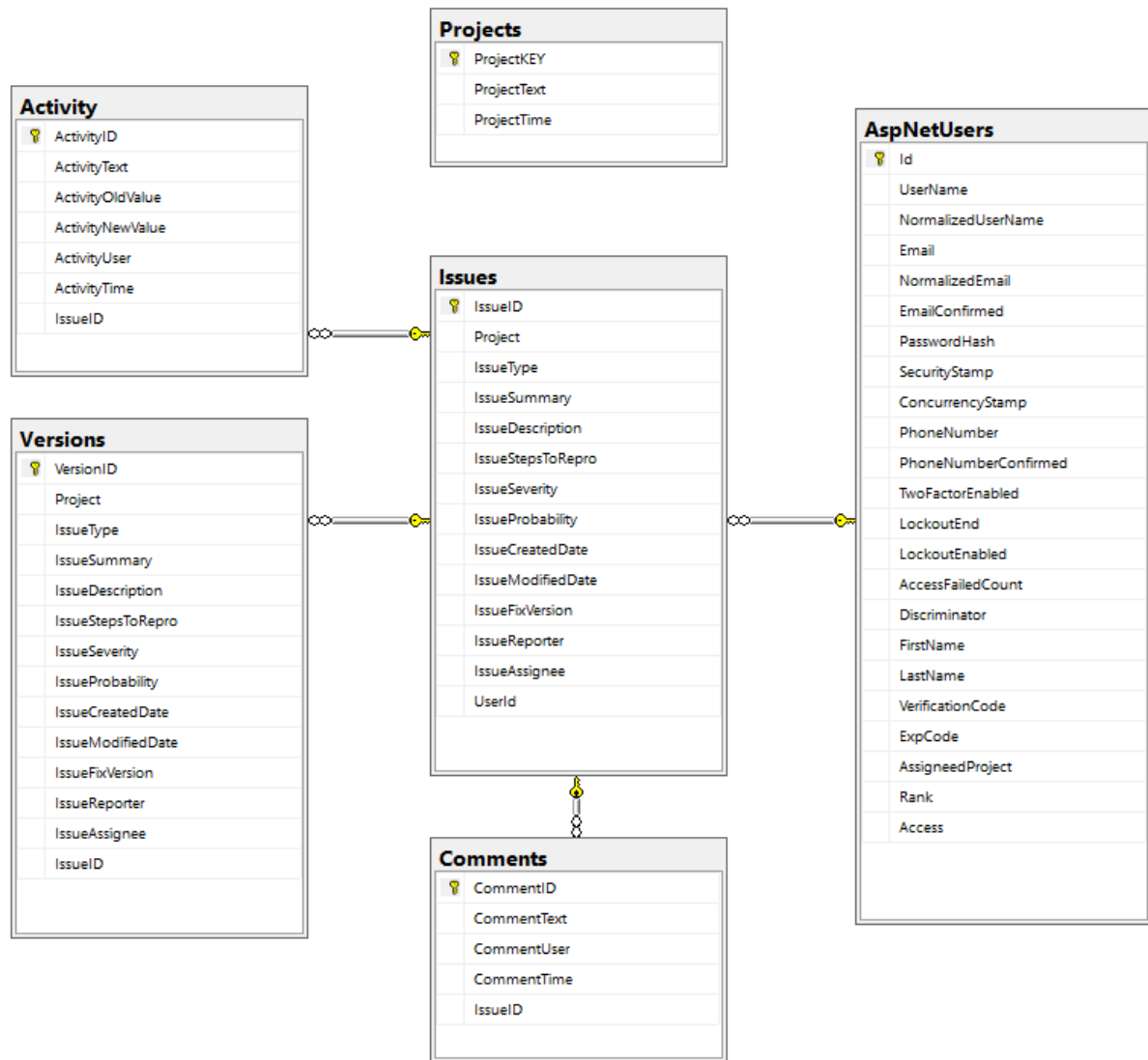


Figure 4.3: Database diagram

The database that was used to develop this application contains the following tables:

- **AspNetUser**
- **Issues**
- **Projects**
- **Activity**
- **Versions**
- **Comments**

The role of the **AspNetUser** table is to manage the information related to the user authentication data. The main columns of this table are the login name chosen by the user,

the first name of the user, the second name of the user, the email, the phone number and the password.

The **Issues** table is used to store the data of a task, request or bug created by the user. The columns of the table are used to store: the project of the file, type of the file, summary, description, steps to reproduce, severity, probability, the time when the file has been created and modified, the version of a build for a fix that has been integrated, the reported and the assignee.

The **Projects** table is used to manage projects and contains three columns: ProjectKEY which is used to identify the project by it's code, ProjectText that specifies the full name of the project and ProjectTime witch is used to track the time passed from the creation of the project.

The **Activity** table is used track the changes made by any user to an existing file. The columns in this table are the ActivityText, ActivityOldValue, ActivityNewValue, ActivityUser, ActivityTime, and an identifier that specifies for which file the mentioned values are available.

The **Versions** table is used to manage previous versions for each file. The columns in this table are similar to the columns of the Issues table as each value needs to be stored in it's specific column and only the IssueID column is required as an identifier that specifies for which file the versions are available.

The **Comments** table is used to store the comments that are added by the users to existing files. The columns in this table are CommentText, CommentUser, CommentTime and an identifier that specifies for which file the comments are available.

The models mentioned in sub-chapter 4.2 will be used to create the database. These models must be configured to allow their association with a database and to achieve this we will use the Microsoft.EntityFrameworkCore package provided by .NET Core through NuGet packages.

This package allows the associations of models with a database and automatically generates tables for user management.

The following code sequence shows the configuration of the database tables according to the application models.

```
1 public class DbContext : IdentityDbContext
2     {
3         public DbContext(DbContextOptions<DbContext> options)
```

```
4      : base(options) { }
5      public DbSet<UserAccount> UserAccount { get; set; }
6      public DbSet<Issues> Issues { get; set; }
7      public DbSet<Activity> Activity { get; set; }
8      public DbSet<Comments> Comments { get; set; }
9      public DbSet<Versions> Versions { get; set; }
10     public DbSet<Projects> Projects { get; set; }
11 }
```

To design the database and use SQL Server Developer it is required to establish a connection to it, the address must be configured for which the database is available. This configuration is done in the Startup.cs file by configuring the services required to design the database. The following code sequence shows the configuration of the database connection.

```
1 public void ConfigureServices(IServiceCollection services)
2 {
3     services.AddMvc().SetCompatibilityVersion(CompatibilityVersion
4         .Version_2_2);
5     services.AddDbContext<DBContext>(options =>
6         options.UseSqlServer(Configuration.
7             GetConnectionString("IdentityConnection")));
8     services.AddIdentityCore<UserAccount>()
9         .AddEntityFrameworkStores<DBContext>();
10    services.Configure<ApplicationSettings>(Configuration
11        .GetSection("ApplicationSettings"));
12 }
```

Information about connecting to the database is taken from the appsettings.json file, which contains various configurations for application functions. The following code sequence shows the settings from the JSON file that are required for connection to the database.

```
1 "AllowedHosts": "*",
2 "ConnectionStrings": {
3     "IdentityConnection": "Server=EDUARD;
4     Database=Atlas; Trusted_Connection=True; MultipleActiveResultSets=True;",
5     "AtlasContext": "Server=(localdb)\\mssqllocaldb;
6     Database=AtlasContext-b7d898bd-2b3b-4f66-b3d9-0dbb491844e5;
7     Trusted_Connection=True;MultipleActiveResultSets=true"
8 }
```

After managing the application models, the "Package Manager Console" command center provided by ASP .NET must be used. The creation of the database is based on files called migrations that generate SQL code for creating and configuring database tables.

The following command line is used in the Package Manager Console to create these migrations.

```
1 Add-Migration "Atlas" -context DbContext
```

This command line will generate the files needed to create and configure database tables according to the models created for the API. If new changes are needed, they can be made by changing migrations or by changing models and creating new migrations.

Figure 4.4 shows the structure of the newly created migrations.

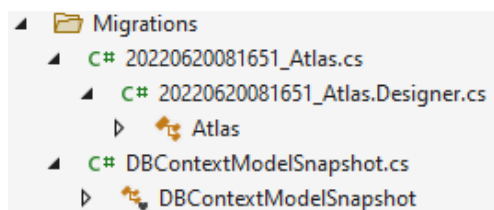


Figure 4.4: The structure of created migrations

After the successful creation, the generated migrations only need to be processed. This can be done by using the following command line in the Package Manager Console.

```
1 Update-Database -context DbContext
```

This command line will process the SQL code and create the database. If changes need to be made to the database, this command will make any necessary changes after generating the migrations.

4.4 Application presentation

Atlas offers more project and errors management rights depending on consumer requirements. In order for a visitor to receive access to a user's rights, they must complete a new account registration form.

Figure 4.5 shows the registration form where the visitor must enter the account creation details.

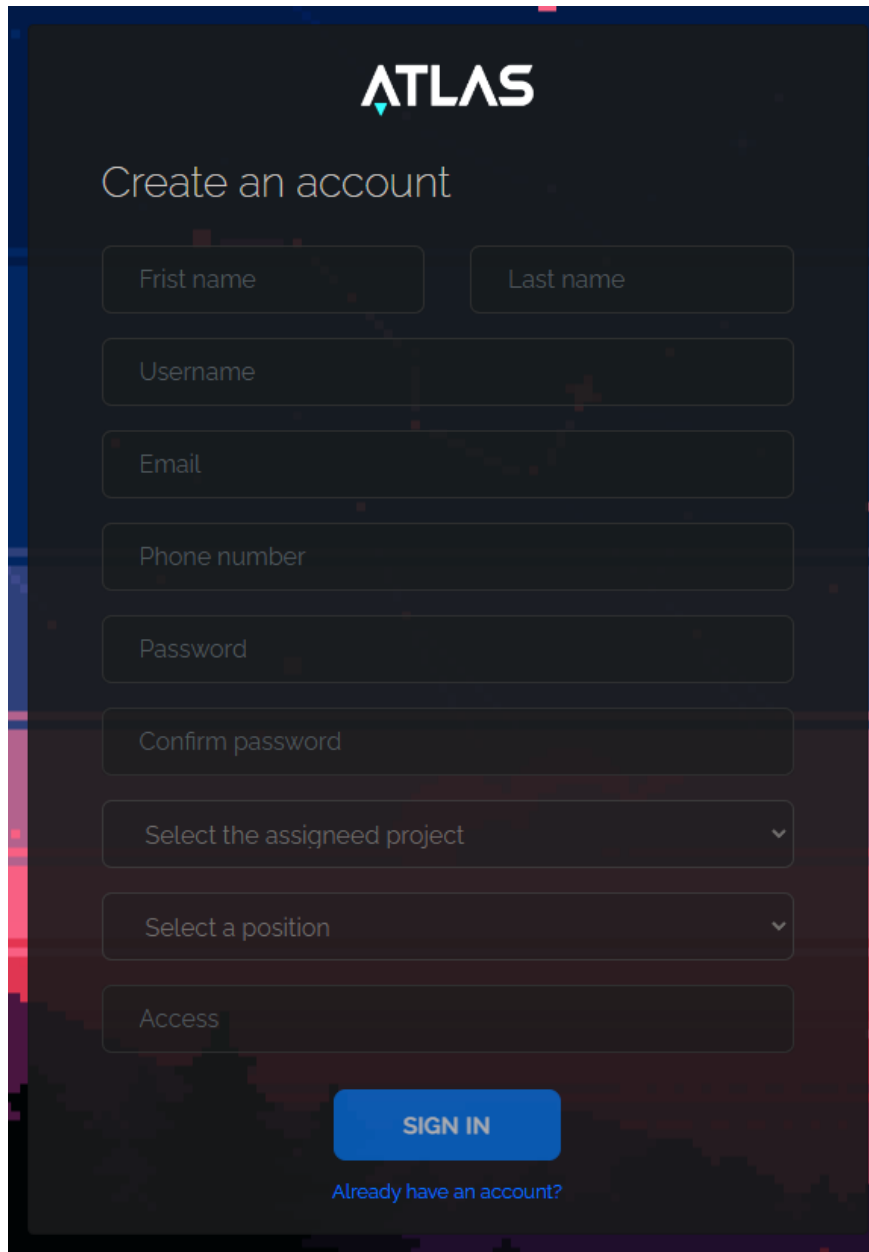
The image shows a user registration form for a system named 'ATLAS'. The form is titled 'Create an account' and is set against a dark background with a subtle world map. It contains several input fields: 'Frist name' (note the typo), 'Last name', 'Username', 'Email', 'Phone number', 'Password', and 'Confirm password'. Below these are two dropdown menus labeled 'Select the assigneed project' (note the typo) and 'Select a position', followed by an 'Access' field. A prominent blue 'SIGN IN' button is located at the bottom of the form, with a link 'Already have an account?' positioned directly beneath it.

Figure 4.5: User's registration form

If the username or email is already registered in the database, an error message is displayed letting the user know that the data they entered has already been used.

Upon successful registration of a visitor, they will receive an email at the address used for registration to inform them that the registration has been completed successfully and they will receive all the rights of a user and will be able to log in to the application.

In order for a user to successfully log in to the application, they must enter the credentials that correspond to the account they created.

Figure 4.6 shows the login form where the user has to enter the account access details.

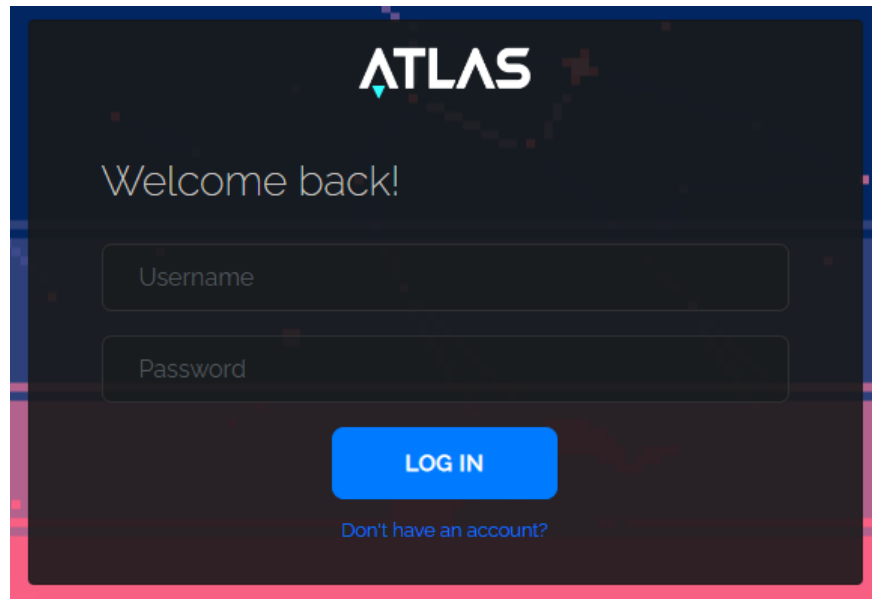


Figure 4.6: User's login form

If the data entered by the user is incorrect, an error message is displayed.

Typescript, HTML and Bootstrap technologies were used to develop this login page, and the login page code is presented in the following code sequence.

```

1 <h3 class="login-heading mb-4" style="color: rgb(209, 205, 199);">
2 Welcome back!</h3>
3 <form #form='ngForm' class="mb-4" autocomplete="off"
4 (submit)="onSubmit(form)">
5 <div class="form-label-group">
6 <input type="text" class="form-control" id="UserName" #UserName="ngModel"
7 name="UserName" [(ngModel)]="formModel.UserName"
8 placeholder="Username" required>
9 <label for="UserName">Username</label></div>
10 <div class="form-label-group">
11 <input type="password" class="form-control" id="Password" #Password="ngModel"
12 name="Password" [(ngModel)]="formModel.Password"
13 placeholder="Password" required>
14 <label for="Password">Password</label>
15 </div>
16 <button type="submit" class="btn btn-lg btn-primary btn-block
17 btn-login text-uppercase font-weight-bold mb-2">Log in</button>
18 <div class="text-center">
19 <a class="small" (click)="this.service.onRegisterRedirect()">
20 Don't have an account?</a>
21 </div>
22 </form>

```

In case of incorrect authentication, the user is notified of possible mistakes that he has made through an alert. This alert was displayed using the ngx-notification package provided by the npm package manager.

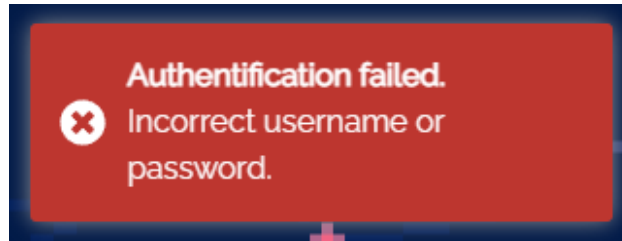


Figure 4.7: Error notification received on a failed log in attempt

After a user successfully logs in, they will be redirected to the main page of the application where they can view a list of all the files that have been created on the assigned project. The page displaying these details is shown in Figure 4.8.

The screenshot shows the main page of the Atlas application. At the top, there is a navigation bar with the 'ATLAS' logo, a '+ New Issue' button, and several menu items: 'Reported by me', 'Assigned to me', 'Versions', 'Projects', and 'Create New Project'. A search bar is also present. The user's name 'Georgescu Eduard' is displayed in the top right corner. Below the navigation bar is a table with the following columns: 'Issue ID', 'Summary', 'Severity', 'Probability', 'Created Date', 'Reporter', and 'Assignee'. The table contains three rows of data. At the bottom of the table, there are navigation buttons: '« Previous', '1', and 'Next »'. The footer of the page states 'Copyright © 2022 All Rights Reserved by Atlas'.

Figure 4.8: Main page of the application

For a more efficient view, at the bottom of the page is shown the total number of pages that separates the files displayed to the user from the files that are existing in the database but are not loaded. The user has two buttons that change the files that are displayed, Previous and Next, which redirect the user to the previous page and the next page, respectively, containing the other files that have not been loaded previously.

This filtering applies the paging function that allows a specific number of files to be

displayed on the page. Such filtering contains the following details: the number of files to be displayed on the page (the default value is ten files per page), the number of the current page, the type of sorting, and the item on which the sort will be performed.

The `getPaginationCount` method sends a request to the API that will return in response the number of pages needed to display all the files.

```
1 getPaginationCount() {  
2     this.http.get<number>(this.BaseURL + '/Count').subscribe(result => {  
3         this.config.totalItems = result;  
4     });  
5 }
```

The `refreshList` method sends a request to the API that will return in a response the files that need to be displayed on the page.

```
1 refreshList() {  
2     this.http.get(this.BaseURL + '/Issues?skip=' + ((this.curentPage - 1)  
3         * this.countPerPage) + '&take=' + this.countPerPage).toPromise()  
4         .then(res => this.list = res as IssuesModel[]);  
5     this.getPaginationCount();  
6 }
```

The `PageChanged` method handles user interaction and page changes. When the user wants to access a new page, this method will send the number of the page that the user wants to view.

```
1 pageChanged(event) {  
2     this.service.config.currentPage = event;  
3     this.service.curentPage = event;  
4     this.service.getPaginationCount();  
5     this.service.refreshList();  
6 }
```

The user can see the details of each file by clicking on the file summary in the Summary column of the table on the page. Upon accessing the link, the user will be redirected to a page that can be viewed in Figure 4.9.

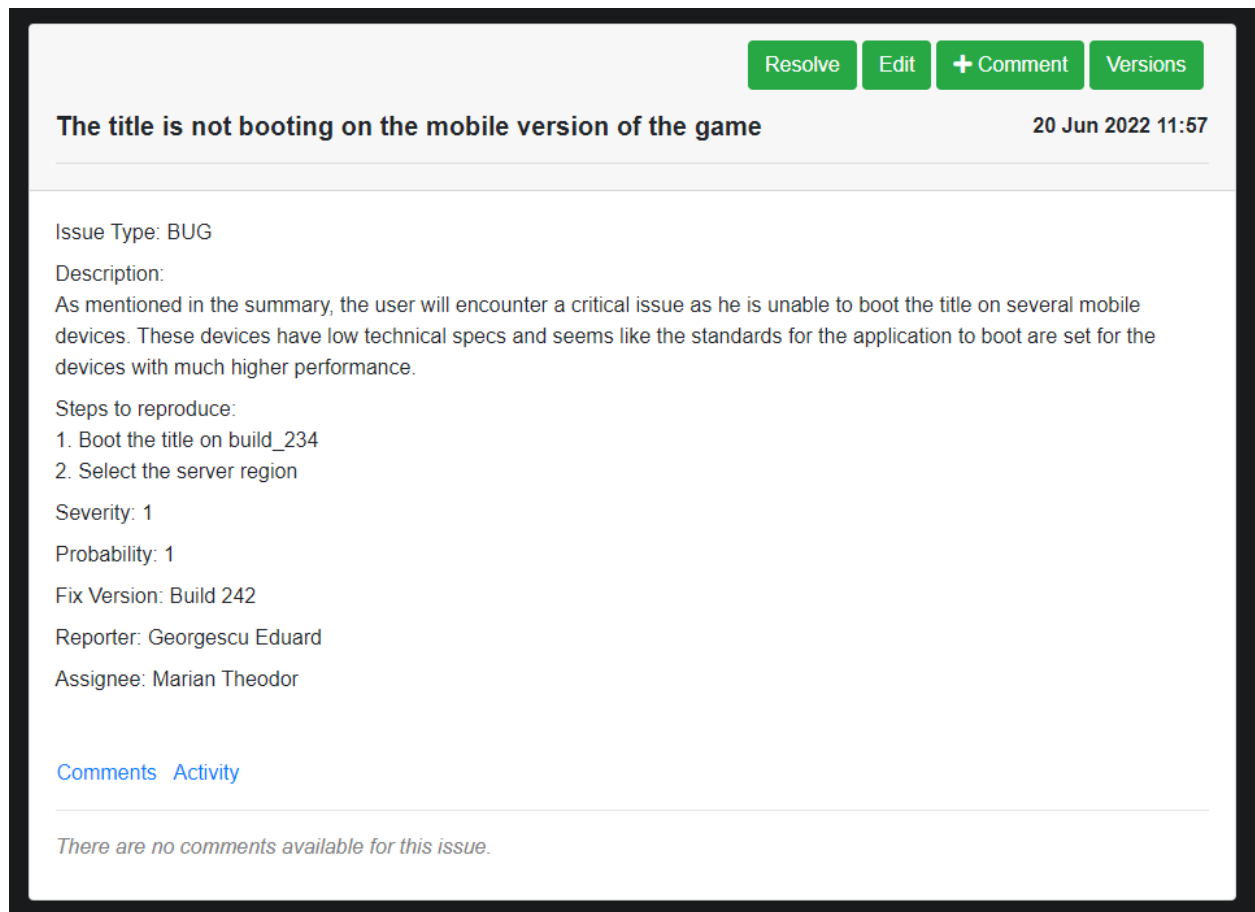


Figure 4.9: The page for viewing the details of a file

This page contains the details of the file selected by the user, which are: summary, the date and time when the file was created, the type of the file, the description of the file, steps to reproduce the issue, the severity and probability of the issue to occur, the version in which the issues will be fixed, the reporter and the assignee of issue.

This page also provides access to methods that can allow to resolve an issue, edit the issue contents, leave a comment for the selected issue, view the comments that have been added to the issue, view the activity log and manage the previous versions of the current issue.

The user can resolve the issue by clicking the **Resolve** button, which will redirect the user to an pop-up that will ask to select the type of resolution for the issue to resolve.

By clicking the **Comment** button a pop-up window will appear asking the user to enter a comment. After leaving a comment to a issue, the comment can be viewed at the bottom of the current page.

The user can access the file modification form by clicking the **Edit** button present next to the summary of the file.

After pressing the mentioned button, the user will be redirected to the file modification

form, the page with the modification form looks like this.



Edit File

Issue Type
BUG

Summary
The title is not booting on the mobile version of the game

Description
As mentioned in the summary, the user will encounter a critical issue as he is unable to boot the title on several mobile devices. These devices have low technical specs and seems like the standards for the application to boot are set for the devices with much higher performance.

Steps to reproduce
1. Boot the title on build_234
2. Select the server region

Severity
1

Probability
1

Figure 4.10: The page that allows editing of file details

This page is similar with the page used to create new files due to a similar form used for the add method.

The user can access the version management page by clicking on the **Versions** button shown next to the summary of the file on the file viewing page. Upon accessing the link, the user will be redirected to the page visible in Figure 4.11.



Version	Modified Date	Modified By
Version 1	20 Jun 2022 17:40	Georgescu Ion-Eduard
Version 2	20 Jun 2022 17:40	Georgescu Ion-Eduard
Version 3	20 Jun 2022 17:40	Georgescu Ion-Eduard
Version 4	20 Jun 2022 17:40	Georgescu Ion-Eduard
Version 5	20 Jun 2022 17:40	Georgescu Ion-Eduard
Version 6	20 Jun 2022 17:40	Georgescu Ion-Eduard
Version 7	20 Jun 2022 17:40	Georgescu Ion-Eduard
Version 8	20 Jun 2022 17:41	Cosmin Potcovaru
Version 9	20 Jun 2022 17:41	Cosmin Potcovaru

Figure 4.11: Versions list page

By pressing that button, the user will be redirected to a page that lists all previous

versions of the file in tabular form, from the first version from which the file was created.

To view the details of a version, the user must click on the name of the desired version in the Version column of the table on the page.

On this page, the user has access to the details and contents of the file for the selected version and how to revert to this version.

The user can return to the selected version by pressing the **Restore** button. Pressing that button will create a new version for the current file details and add it to the file version table, after which the file will be modified by changing the details and content to the version desired to restore to.

In addition to the features presented above, the user has multiple options to filter the displayed issues in tables by accessing the **Reported by me** and **Assigned to me** filters that are presented at the top of the page.

In addition to this two filters, the user also has the option to filter the results of the issues that will be displayed by using a search bar that is present besides the two mentioned filters. The filters and the search bar are presented in the Figure 4.12

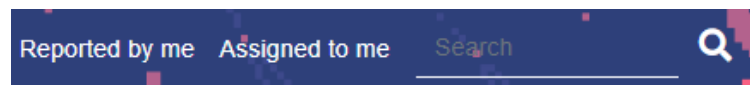


Figure 4.12: Additional filters for search

Chapter 5: Conclusions

The dissertation “Project and Error Management Application” presents the development of the Atlas Web application within the Web network. Atlas is a web application that allows the management of the projects, errors, requests and tasks with an authentication method used for organizations.

This dissertation highlights the development of a web application using the tools offered by ASP .NET Core MVC and the component management benefits offered by the Angular platform.

The use of the ASP .NET Core development platform, in which the Atlas application API was implemented, is still a good choice for developers, 6 years after its introduction, mainly due to the facilities offered for structuring the application models.

Using the Angular development platform, which helped with the implementation of the user interface for the Atlas application, provides a simple understanding and development of web applications, mainly due to the ability to manage application components and configurations.

5.1 Personal contributions

The Atlas application innovates the ability to manage the projects and errors that an organization is working on by providing a set of tools available to the user.

The tools available to the user are:

- Authentication management available for any user
- Management options for the created projects
- Management options for errors, requests and tasks
- Activity and version tracking for errors
- Adding comments to the created errors, requests and tasks

The application can be used for the management of other applications that are in development by an organization or by a group of students and the educational purposes provide a simple understanding of the tools used during development.

All of the above mentions make the Atlas application a complete experience that provides the user with management options for projects and errors in a nicely designed web application.

5.2 Future improvements

As future improvements, the application can be extended by achieving the following objectives:

- Due to the implementation, it is possible to extend the application by adding the possibility to attach files to the created errors.
- Providing a wide range of predefined themes, as well as the possibility to create a personalized theme, by selecting multiple color schemes.
- Implement a verification of the user when trying to login into account.

Bibliography

- [1] Aloï resources. <https://www.aloi.io/wp-content/uploads/2019/09/api-visual.png>. Accessed: 2022-05-13.
- [2] Angular modules. <https://i0.wp.com/www.intertech.com/wp-content/uploads/2018/04/modules.jpg?ssl=1>. Accessed: 2022-05-13.
- [3] In-app navigation: routing to views. <https://angular.io/guide/router>. Accessed: 2022-05-13.
- [4] Introduction to components and templates. <https://angular.io/guide/architecture-components#templates-and-views>. Accessed: 2022-05-13.
- [5] Overview to asp.net core. <https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0>. Accessed: 2022-05-13.
- [6] Typescript inheritance deep dive. <http://www.mukeshkumar.net/articles/typescript/typescript-inheritance-deep-dive>. Accessed: 2022-05-13.
- [7] Nate Barbettini. *The little ASP.NET Core book*. 2018.
- [8] Adam Freeman. *Pro Angular 6 Third Edition*. Apress, 2018.
- [9] Andrew Lock. *ASP .NET Core in Action*. Manning, 2018.
- [10] Jeremy Wilken. *Angular In Action*. Manning, 2018.