PennStateSoft

Coding and Testing Document

Version 1.0

TEAM MEMBERS

Name	Student ID
Eesaa Philips	9 3717 6147
Garrett Adams	9 4659 7647
Huy Tran	9 3572 8072

Revision History

Date	Version	Description	Author
08/13/21	1.0	The initial draft of the coding and testing document for our MSS	Eesaa Philips, Garrett Adams, Huy Tran

Table of Contents

1. Introduction 6
 1.1 Purpose 6
 1.2 Scope 6
  1.2.1 In Scope 6
  1.2.2 Out Scope 6
 1.3 Definitions, Acronyms, and Abbreviations 6
 1.4 References 6

2. Testing Results 7
 2.1 Test Approaches 7
 2.2 Built-in Code Audit 7
 2.3 Code Static Analysis 8
  2.3.1 Client 8
  2.3.2 Server 8
 2.4 Bugs Found 9
  2.4.1 Built in Code Audit 9
  2.4.2 Static Analysis 9
 2.5 Bugs Addressed 9
  2.5.1 Built in Code Audit 9
  2.5.2 Static Analysis 10
 2.6 Manual Testing Bugs/Solutions 10
 2.7 Security Implementations 12
  2.7.1 JSON Web Tokens 12
  2.7.2 Error Handling 12
  2.7.3 Whitelisted Domains 12
  2.7.4 Local Storage 12
  2.7.5 Data Storage 12
  2.7.6 Uniform Classes 12

3. Code Documentation – Client Side 13
 3.1 Module: Authentication 13
  3.1.1 Class: AuthService 14
  3.1.2 Class: RegisterComponent 14
  3.1.3 Class: LoginComponent 15
  3.1.4 Class: LoginVM 16
  3.1.5 Class: RegisterVM 16
  3.1.6 Class: RegisterAdminVM 16
 3.2 Module: Complaint 17
  3.2.1 Class: ComplaintService 17
  3.2.2 Class: ComplaintHistoryComponent 17
  3.2.3 Class: CreateComplaintComponent 17
  3.2.4 Class: ManageComplaintComponent 18
  3.2.5 Class: UserComplaintsComponent 18

3.3 Module: Profile 19
3.3.1 Class: ProfileService 20
3.3.2 Class: ManageProfileComponent 20
3.3.3 Class: ChangePasswordComponent 21
3.3.4 Class: Billing 21
3.3.5 Class: Client 21
3.3.6 Class: User 21
3.3.7 Class: Admin 22
3.4 Module: Reservation 23
3.4.1 Class: ReservationService 24
3.4.2 Class: ReservationModalComponent 24
3.4.3 Class: CalendarViewComponent 25
3.4.4 Class: ManageMeetingComponent 26
3.4.5 Class: Reservation 27
3.5 Module: Room 28
3.5.1 Class: RoomService 29
3.5.2 Class: EditOrCreateRoomComponent 29
3.5.3 Class: RoomsComponent 29
3.5.4 Class: Room 30

4. Code Documentation – Server Side 31
4.1 Module: Authentication 31
4.1.1 Module Class Diagram 31
4.1.2 Class: Authentication Controller 32
4.1.3 Class: RegisterAdministratorVM 33
4.1.4 Class: RegisterClientVM 33
4.1.5 Class: LoginVM 33
4.2 Module: Room 34
4.2.1 Class: RoomController 34
4.2.2 Class: RoomDataService 35
4.2.3 Class: RoomDataAccess 36
4.2.4 Class: Room (Entity) 36
4.3 Module: Complaints 37
4.3.1 Class: ComplaintController 38
4.3.2 Class: ComplaintDataService 38
4.3.3 Class: ComplaintDataAccess 38
4.3.4 Class: Complaint (Entity) 39
4.4 Module: Reservation 40
4.4.1 Class: ReservationController 41
4.4.2 Class: ReservationDataService 41
4.4.3 Class: ReservationDataAccess 42
4.4.4 Class: Room (Entity) 43
4.4.5 Class: Client (Entity) 43
4.4.6 Class: Reservation (Entity) 43
4.5 Module: Profile 44
4.5.1 Class: ProfileController 45

4.5.2 Class: ProfileDataService 45
4.5.3 Class: ProfileDataAccess 46
4.5.4 Class: Administrator (Entity) 47
4.5.5 Class: Client (Entity) 47
4.5.6 Class: Billing (Entity) 47

5. Team Members Log Sheets 48

5.1 Eesaa Philips 48
5.2 Huy Tran 48
5.3 Garrett Adams 48

Coding and Testing Document

1. Introduction

The purpose of this document is to provide all details of the Coding and Testing phase for the Meeting Scheduling System (MSS) and point out parts that have not been implemented nor tested.

1.1 Purpose

The first purpose of this document is to assure that the MSS meets the full requirements and satisfies all use case scenarios but still maintains the highest level of security as specified in both System Design Document and Software Requirements Specifications Document.
The second purpose of this document is to expose all issues and associated risks as well as point out which parts have not been implemented nor tested.
Any changes, updates, or deletions to the system requirements will be documented and tested.

1.2 Scope

1.2.1 In Scope

The MSS Test Plan defines the Unit, Integration, System, Regular Expression (RegEx), and Client Acceptance testing approach. The scope includes the following:

- Testing of all functional, application performance, security, and use case requirements listed in the System Design Document and Software Requirements Specifications Document.

1.2.2 Out Scope

The following are considered out of scope for the Test Plan and MSS:

- Functional requirements testing for systems outside the MSS
- Testing of Business Standard Operating Procedure.

1.3 Definitions, Acronyms, and Abbreviations

MSS: Meeting Scheduling System
RegEx: Regular Expression
XSS: Cross-Site Scripting

1.4 References

codelyzer. npm. (n.d.). https://www.npmjs.com/package/codelyzer.

Mikadumont. (n.d.). *Code analysis using ROSLYN analyzers - Visual Studio (windows)*. Code analysis using Roslyn analyzers - Visual Studio (Windows) | Microsoft Docs. https://docs.microsoft.com/en-us/visualstudio/code-quality/roslyn-analyzers-overview?view=vs-2019.

Meeting Scheduling System Software Requirements Specifications Document Version 1.0.

Meeting Scheduling System - System Design Document Version 1.0.

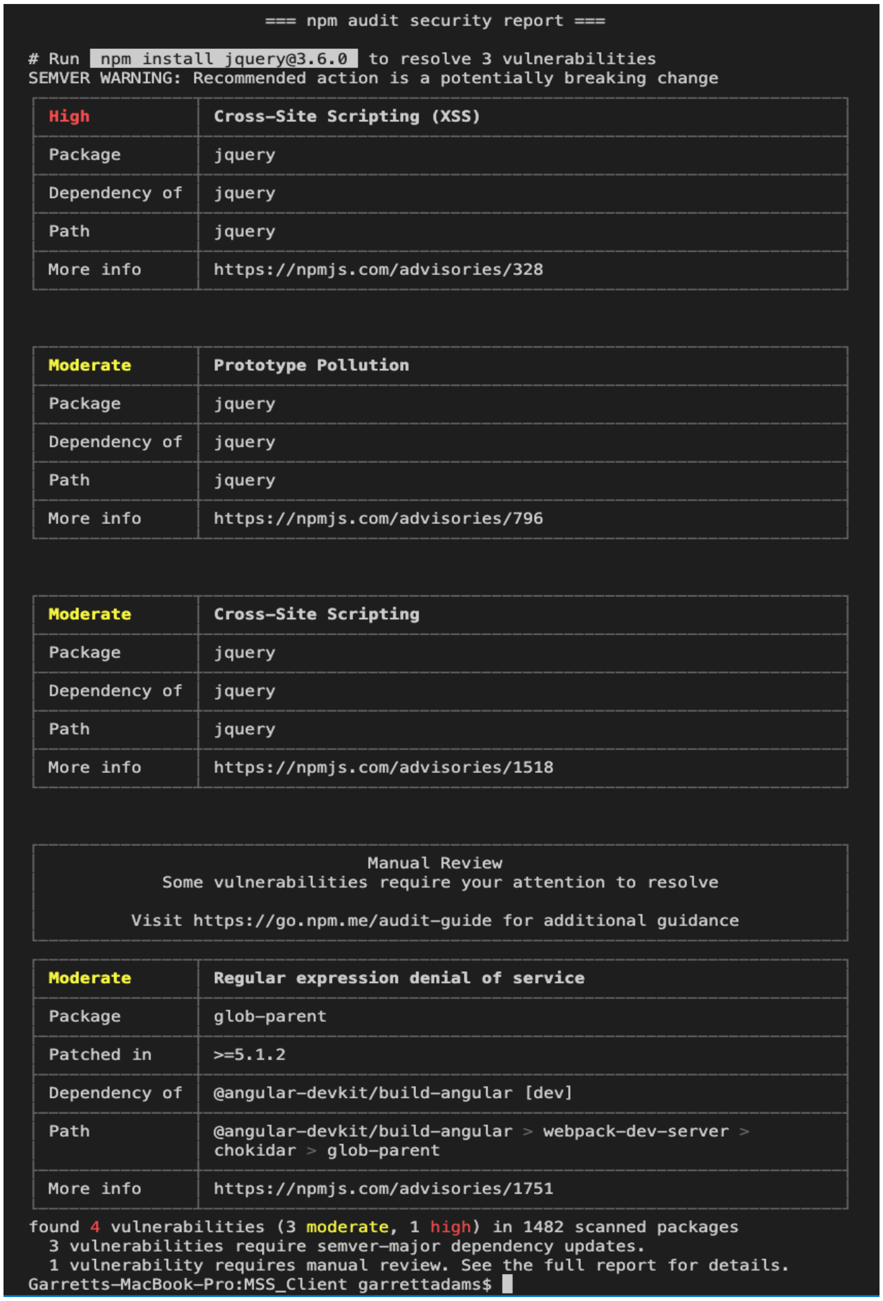
Mikejo5000. (n.d.). *Calculate code metrics - visual studio (windows)*. Calculate code metrics - Visual Studio (Windows) | Microsoft Docs. https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2019.

2. Testing Results

2.1 Test Approaches

During the coding, we utilized built-in tools and static analysis tools to help identify vulnerabilities, and afterward, the complete functionality had been implemented we utilized manual testing to help ensure that none were remaining.

2.2 Built-in Code Audit

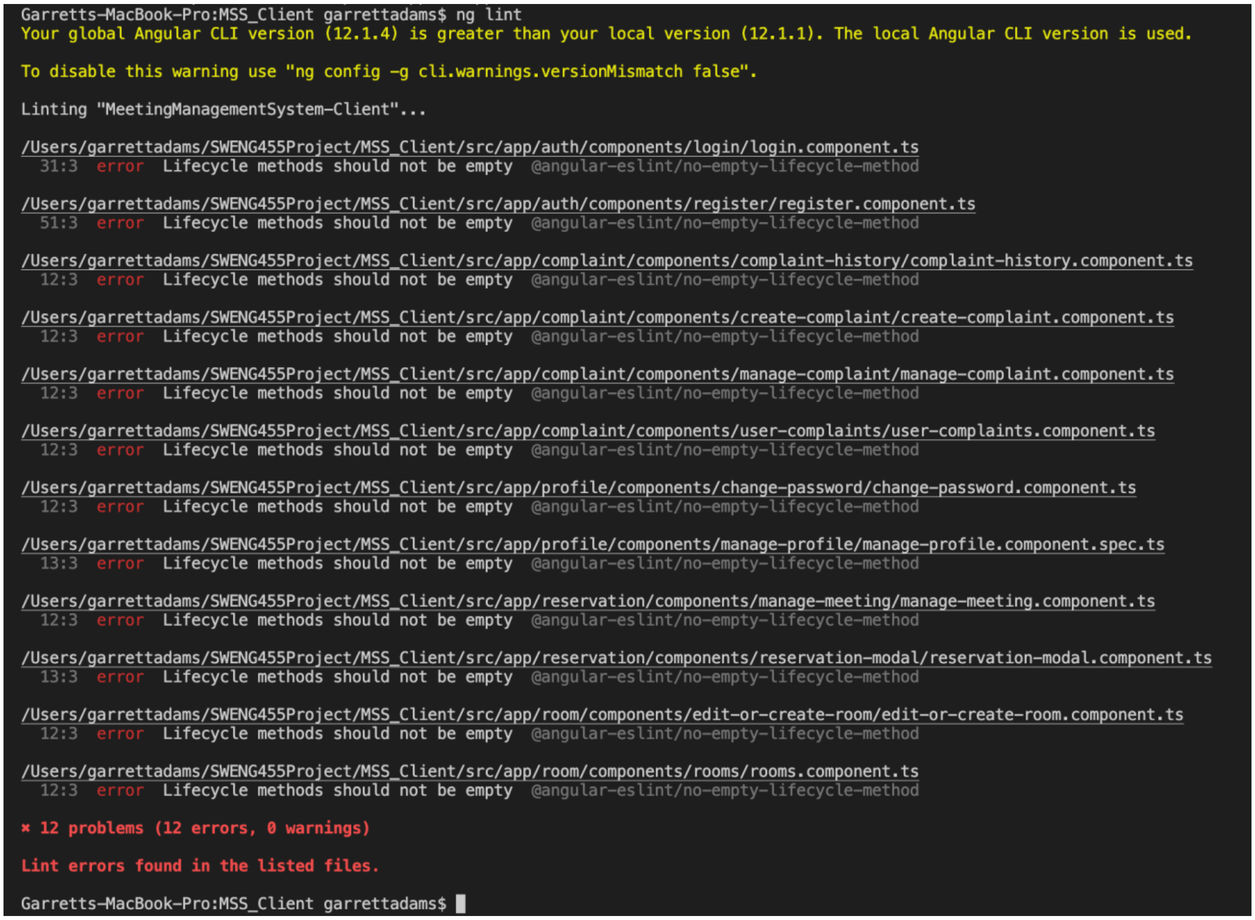


```
                        === npm audit security report ===

# Run  npm install jquery@3.6.0  to resolve 3 vulnerabilities
SEMVER WARNING: Recommended action is a potentially breaking change
┌───────────────┬──────────────────────────────────────────────────────────────┐
│ High          │ Cross-Site Scripting (XSS)                                   │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Package       │ jquery                                                       │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Dependency of │ jquery                                                       │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Path          │ jquery                                                       │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ More info     │ https://npmjs.com/advisories/328                             │
└───────────────┴──────────────────────────────────────────────────────────────┘


┌───────────────┬──────────────────────────────────────────────────────────────┐
│ Moderate      │ Prototype Pollution                                          │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Package       │ jquery                                                       │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Dependency of │ jquery                                                       │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Path          │ jquery                                                       │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ More info     │ https://npmjs.com/advisories/796                             │
└───────────────┴──────────────────────────────────────────────────────────────┘


┌───────────────┬──────────────────────────────────────────────────────────────┐
│ Moderate      │ Cross-Site Scripting                                         │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Package       │ jquery                                                       │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Dependency of │ jquery                                                       │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Path          │ jquery                                                       │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ More info     │ https://npmjs.com/advisories/1518                            │
└───────────────┴──────────────────────────────────────────────────────────────┘


┌──────────────────────────────────────────────────────────────────────────────┐
│                                Manual Review                                 │
│            Some vulnerabilities require your attention to resolve            │
│                                                                              │
│         Visit https://go.npm.me/audit-guide for additional guidance          │
└──────────────────────────────────────────────────────────────────────────────┘
┌───────────────┬──────────────────────────────────────────────────────────────┐
│ Moderate      │ Regular expression denial of service                         │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Package       │ glob-parent                                                  │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Patched in    │ >=5.1.2                                                      │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Dependency of │ @angular-devkit/build-angular [dev]                          │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Path          │ @angular-devkit/build-angular > webpack-dev-server >         │
│               │ chokidar > glob-parent                                       │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ More info     │ https://npmjs.com/advisories/1751                            │
└───────────────┴──────────────────────────────────────────────────────────────┘
found 4 vulnerabilities (3 moderate, 1 high) in 1482 scanned packages
  3 vulnerabilities require semver-major dependency updates.
  1 vulnerability requires manual review. See the full report for details.
Garretts-MacBook-Pro:MSS_Client garrettadams$ 
```

The npm audit command sends a description of the dependencies configured in the project to your default registry and asks for a report of known vulnerabilities.

2.3 Code Static Analysis

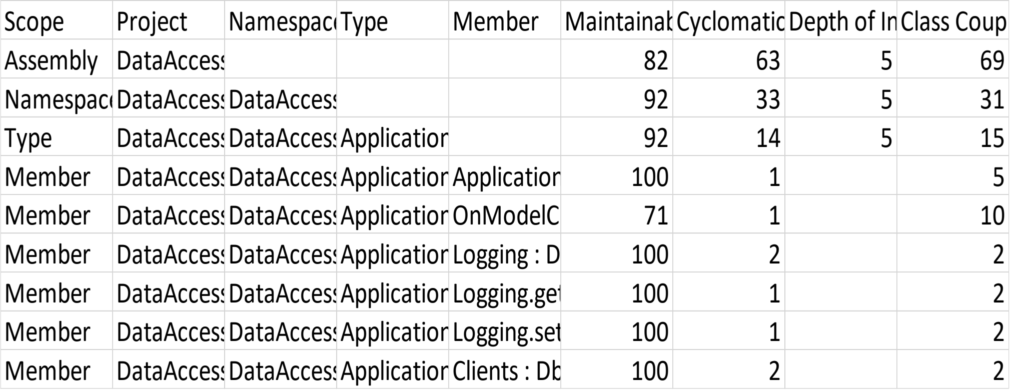
2.3.1 Client



```
Garretts-MacBook-Pro:MSS_Client garrettadams$ ng lint
Your global Angular CLI version (12.1.4) is greater than your local version (12.1.1). The local Angular CLI version is used.

To disable this warning use "ng config -g cli.warnings.versionMismatch false".

Linting "MeetingManagementSystem-Client"...

/Users/garrettadams/SWENG455Project/MSS_Client/src/app/auth/components/login/login.component.ts
  31:3  error  Lifecycle methods should not be empty  @angular-eslint/no-empty-lifecycle-method

/Users/garrettadams/SWENG455Project/MSS_Client/src/app/auth/components/register/register.component.ts
  51:3  error  Lifecycle methods should not be empty  @angular-eslint/no-empty-lifecycle-method

/Users/garrettadams/SWENG455Project/MSS_Client/src/app/complaint/components/complaint-history/complaint-history.component.ts
  12:3  error  Lifecycle methods should not be empty  @angular-eslint/no-empty-lifecycle-method

/Users/garrettadams/SWENG455Project/MSS_Client/src/app/complaint/components/create-complaint/create-complaint.component.ts
  12:3  error  Lifecycle methods should not be empty  @angular-eslint/no-empty-lifecycle-method

/Users/garrettadams/SWENG455Project/MSS_Client/src/app/complaint/components/manage-complaint/manage-complaint.component.ts
  12:3  error  Lifecycle methods should not be empty  @angular-eslint/no-empty-lifecycle-method

/Users/garrettadams/SWENG455Project/MSS_Client/src/app/complaint/components/user-complaints/user-complaints.component.ts
  12:3  error  Lifecycle methods should not be empty  @angular-eslint/no-empty-lifecycle-method

/Users/garrettadams/SWENG455Project/MSS_Client/src/app/profile/components/change-password/change-password.component.ts
  12:3  error  Lifecycle methods should not be empty  @angular-eslint/no-empty-lifecycle-method

/Users/garrettadams/SWENG455Project/MSS_Client/src/app/profile/components/manage-profile/manage-profile.component.spec.ts
  13:3  error  Lifecycle methods should not be empty  @angular-eslint/no-empty-lifecycle-method

/Users/garrettadams/SWENG455Project/MSS_Client/src/app/reservation/components/manage-meeting/manage-meeting.component.ts
  12:3  error  Lifecycle methods should not be empty  @angular-eslint/no-empty-lifecycle-method

/Users/garrettadams/SWENG455Project/MSS_Client/src/app/reservation/components/reservation-modal/reservation-modal.component.ts
  13:3  error  Lifecycle methods should not be empty  @angular-eslint/no-empty-lifecycle-method

/Users/garrettadams/SWENG455Project/MSS_Client/src/app/room/components/edit-or-create-room/edit-or-create-room.component.ts
  12:3  error  Lifecycle methods should not be empty  @angular-eslint/no-empty-lifecycle-method

/Users/garrettadams/SWENG455Project/MSS_Client/src/app/room/components/rooms/rooms.component.ts
  12:3  error  Lifecycle methods should not be empty  @angular-eslint/no-empty-lifecycle-method

✖ 12 problems (12 errors, 0 warnings)

Lint errors found in the listed files.

Garretts-MacBook-Pro:MSS_Client garrettadams$ █
```

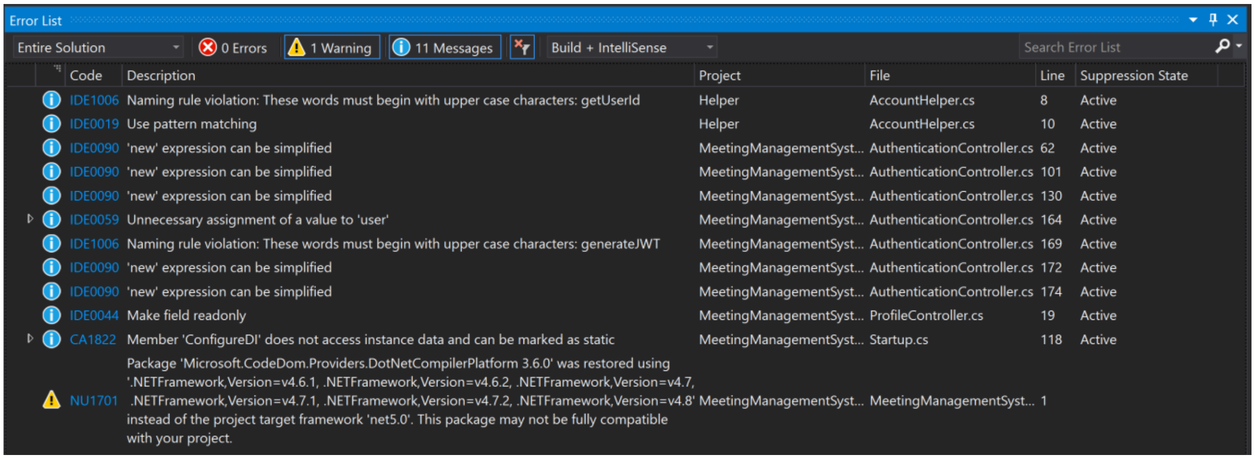
The code static analysis tool used to check the client side was codelyzer. This tool uses a set of tslint rules and checks Angular Typescript and HTML for vulnerabilities.

2.3.2 Server



Scope	Project	Namespac	Type	Member	Maintainab	Cyclomatic	Depth of In	Class Coup
Assembly	DataAcces				82	63	5	69
Namespac	DataAcces	DataAcces			92	33	5	31
Type	DataAcces	DataAcces	Applicatio		92	14	5	15
Member	DataAcces	DataAcces	Applicatio	Applicatio	100	1		5
Member	DataAcces	DataAcces	Applicatio	OnModelC	71	1		10
Member	DataAcces	DataAcces	Applicatio	Logging : D	100	2		2
Member	DataAcces	DataAcces	Applicatio	Logging.get	100	1		2
Member	DataAcces	DataAcces	Applicatio	Logging.set	100	1		2
Member	DataAcces	DataAcces	Applicatio	Clients : Db	100	2		2

The figure above is the Code Metric Result which is the set of measurements providing insightful view into the developing code. The measurements include the complexity and maintainability of the code. To have more specific view, please double-click the figure above. To have more information, please click to this link.



	Code	Description	Project	File	Line	Suppression State
ⓘ	IDE1006	Naming rule violation: These words must begin with upper case characters: getUserId	Helper	AccountHelper.cs	8	Active
ⓘ	IDE0019	Use pattern matching	Helper	AccountHelper.cs	10	Active
ⓘ	IDE0090	'new' expression can be simplified	MeetingManagementSyst...	AuthenticationController.cs	62	Active
ⓘ	IDE0090	'new' expression can be simplified	MeetingManagementSyst...	AuthenticationController.cs	101	Active
ⓘ	IDE0090	'new' expression can be simplified	MeetingManagementSyst...	AuthenticationController.cs	130	Active
ⓘ	IDE0059	Unnecessary assignment of a value to 'user'	MeetingManagementSyst...	AuthenticationController.cs	164	Active
ⓘ	IDE1006	Naming rule violation: These words must begin with upper case characters: generateJWT	MeetingManagementSyst...	AuthenticationController.cs	169	Active
ⓘ	IDE0090	'new' expression can be simplified	MeetingManagementSyst...	AuthenticationController.cs	172	Active
ⓘ	IDE0090	'new' expression can be simplified	MeetingManagementSyst...	AuthenticationController.cs	174	Active
ⓘ	IDE0044	Make field readonly	MeetingManagementSyst...	ProfileController.cs	19	Active
ⓘ	CA1822	Member 'ConfigureDI' does not access instance data and can be marked as static	MeetingManagementSyst...	Startup.cs	118	Active
⚠	NU1701	Package 'Microsoft.CodeDom.Providers.DotNetCompilerPlatform 3.6.0' was restored using '.NETFramework,Version=v4.6.1, .NETFramework,Version=v4.6.2, .NETFramework,Version=v4.7, .NETFramework,Version=v4.7.1, .NETFramework,Version=v4.7.2, .NETFramework,Version=v4.8' instead of the project target framework 'net5.0'. This package may not be fully compatible with your project.	MeetingManagementSyst...	MeetingManagementSyst...	1	

The figure above is the Configuration Analyzation Result which is generated by Roslyn analyzer. The result indicates the spectrum of severity levels. The figure above has shown the final result. To learn more details, please check this link.

2.4 Bugs Found

2.4.1 *Built in Code Audit*

Cross-Site Scripting XSS- This version of jquery interpreted the text responses from the ajax requests incorrectly and there was a risk of it automatically executing the contents in JQuery.globalEval unintentionally.
Prototype Pollution- This version of jquery has a vulnerability where the extend() method allows an attacker to modify the prototype for Object.
Cross-Site Scripting- This version of jquery allows the passing of HTML from untrusted sources to one of jQuery's manipulation methods.
Regular Expression Denial of Service- The version of glob-parent had a regular expression that made it vulnerable to a denial-of-service attack.

2.4.2 *Static Analysis*

Lifecycle methods should not be empty- This error exists because the software detected a method without any contents and supposes this is an error when coding.

There were no bugs found during a static analysis of the server-side code.

2.5 Bugs Addressed

2.5.1 *Built in Code Audit*

Cross-Site Scripting XSS - This was resolved by updating the version of jquery.
Prototype Pollution - This was resolved by updating the version of jquery.
Cross-Site Scripting - This was resolved by updating the version of jquery.
Regular Expression Denial of Service - This was resolved by updating the version of glob-parent.

2.5.2 *Static Analysis*

Lifecycle methods should not be empty- This error was mistakenly called on a constructor that injects a service. Once going back and verifying that these are rightfully empty, we can dismiss them or write an override.

2.6 Manual Testing Bugs/Solutions

- User in different time zone had conflicting reservations when validating the dates

  FIX : save all dates in the database as UTC and display them in each user's respective time zone

- Duplicate participants in meeting would cause multiple items to be deleted when removing the participant

  FIX : check if the participant does not already exist before adding them

- When two reservations were after each other, the system recognized that they are conflicting

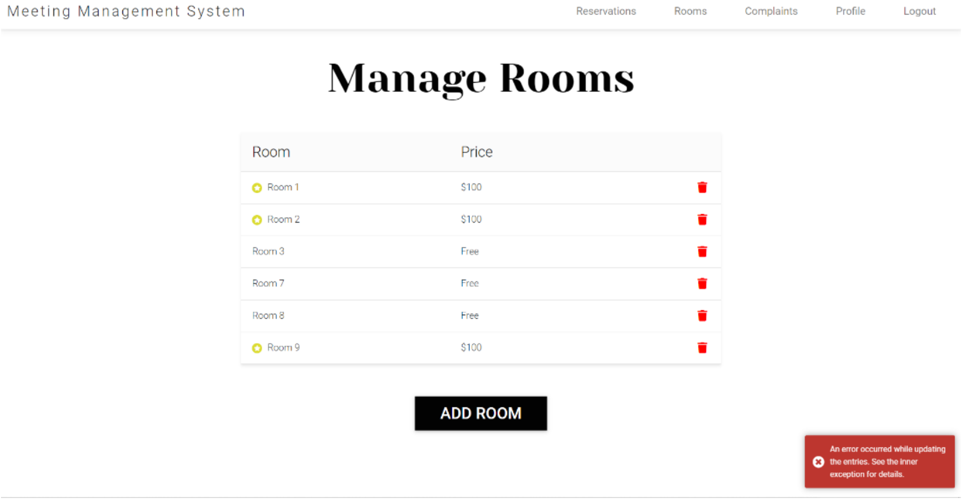
  FIX : check for conflicting meetings using differences in minutes not in hours to avoid rounding

- When the user would click "logout" when they are not logged in, the authentication route guard would enter an infinite loop

  FIX : logout button only works when the user is logged in

- When a room that has previous reservations is deleted, the database throws an exception because of the foreign key. This resulted in a non-readable error to the user.

  FIX : if the room is in use, the dataservice throws an InvalidOperation exception with a specific message which is caught by the controller. The controller then sends the internal message to the client.

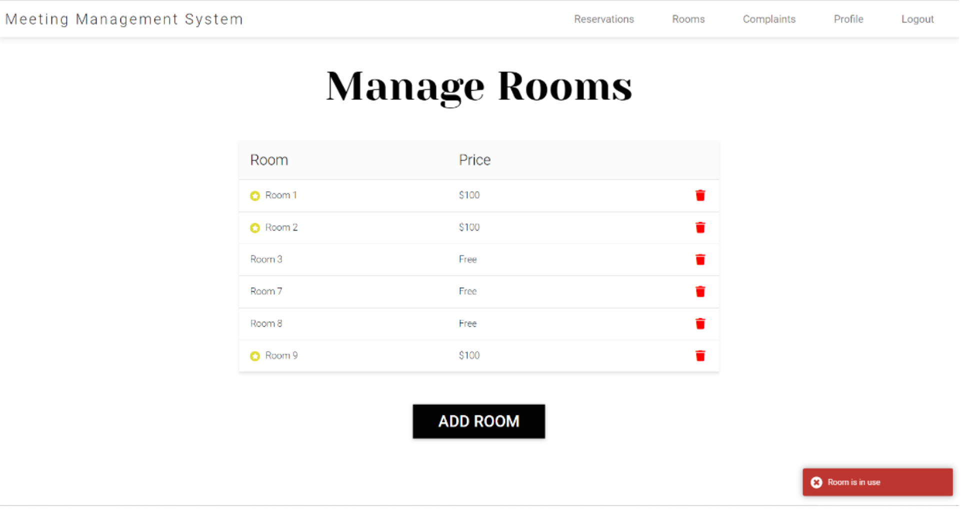


Manage Rooms

Room	Price	
○ Room 1	$100	🗑
○ Room 2	$100	🗑
Room 3	Free	🗑
Room 7	Free	🗑
Room 8	Free	🗑
○ Room 9	$100	🗑

An error occurred while updating the entries. See the inner exception for details.

Before Fixing



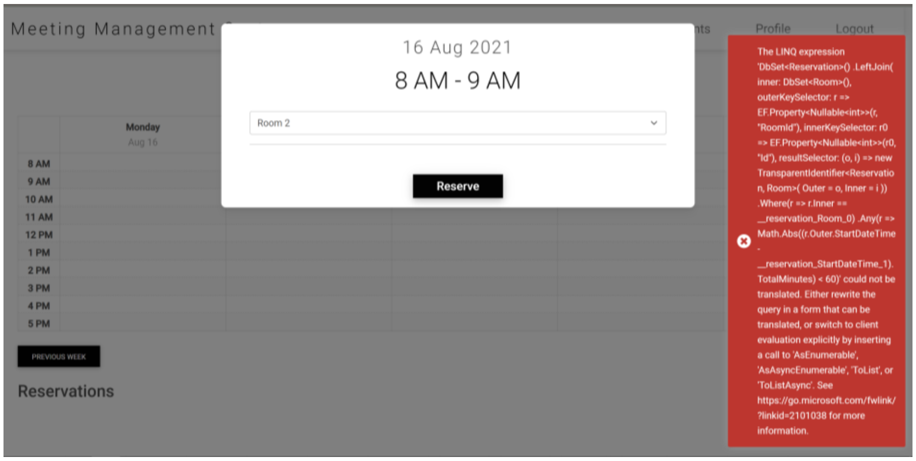
Manage Rooms

Room	Price
Room 1	$100
Room 2	$100
Room 3	Free
Room 7	Free
Room 8	Free
Room 9	$100

Room is in use

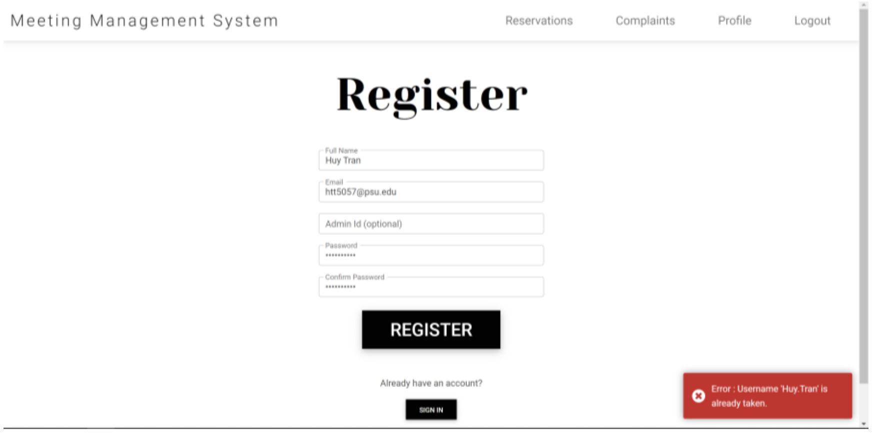
After fixing

- Displays technical error details in the Client View.
  Test approach: Fuzz testing
  Fix: shortens the error message and hides out informative details



```
The LINQ expression 'DbSet<Reservation>() .LeftJoin( inner: DbSet<Room>(), outerKeySelector: r => EF.Property<Nullable<int>>(r, "RoomId"), innerKeySelector: r0 => EF.Property<Nullable<int>>(r0, "Id"), resultSelector: (o, i) => new TransparentIdentifier<Reservation, Room>( Outer = o, Inner = i )) .Where(r => r.Inner == __reservation_Room_0) .Any(r => Math.Abs((r.Outer.StartDateTime - __reservation_StartDateTime_1).TotalMinutes) < 60)' could not be translated. Either rewrite the query in a form that can be translated, or switch to client evaluation explicitly by inserting a call to 'AsEnumerable', 'AsAsyncEnumerable', 'ToList', or 'ToListAsync'. See https://go.microsoft.com/fwlink/?linkid=2101038 for more information.
```

- The system does not allow a person having duplicated name to create an account.
  Test approach: Functional testing
  Fix: Creates and uses FullName property instead of using UserName of IdentityUser



Register

Full Name: Huy Tran

Email: htt5057@psu.edu

Admin Id (optional)

Password: ••••••••••

Confirm Password: ••••••••••

Already have an account?

Error : Username 'Huy.Tran' is already taken.

2.7 Security Implementations

2.7.1 JSON Web Tokens

These long, seemingly random tokens are used to authenticate the user using a hashing algorithm and a secret key. It has an expiry and cannot be used after its lifetime has been reached. All routes require authorization to access the API endpoints so an attacker cannot easily make API calls.

2.7.2 Error Handling

The API catches errors as they are received and provides a simple user-facing error without exposing the technical details of the exception.

2.7.3 Whitelisted Domains

All requests and tokens are verified against a whitelist of domains so attackers cannot run their own clients and call the API's.

2.7.4 Local Storage

No sensitive data is stored in cookies at the client-side since they can be viewed by the user. All authentication (such as checking if the user is an admin or client) happens through the API to prevent privilege-escalation attacks.

2.7.5 Data Storage

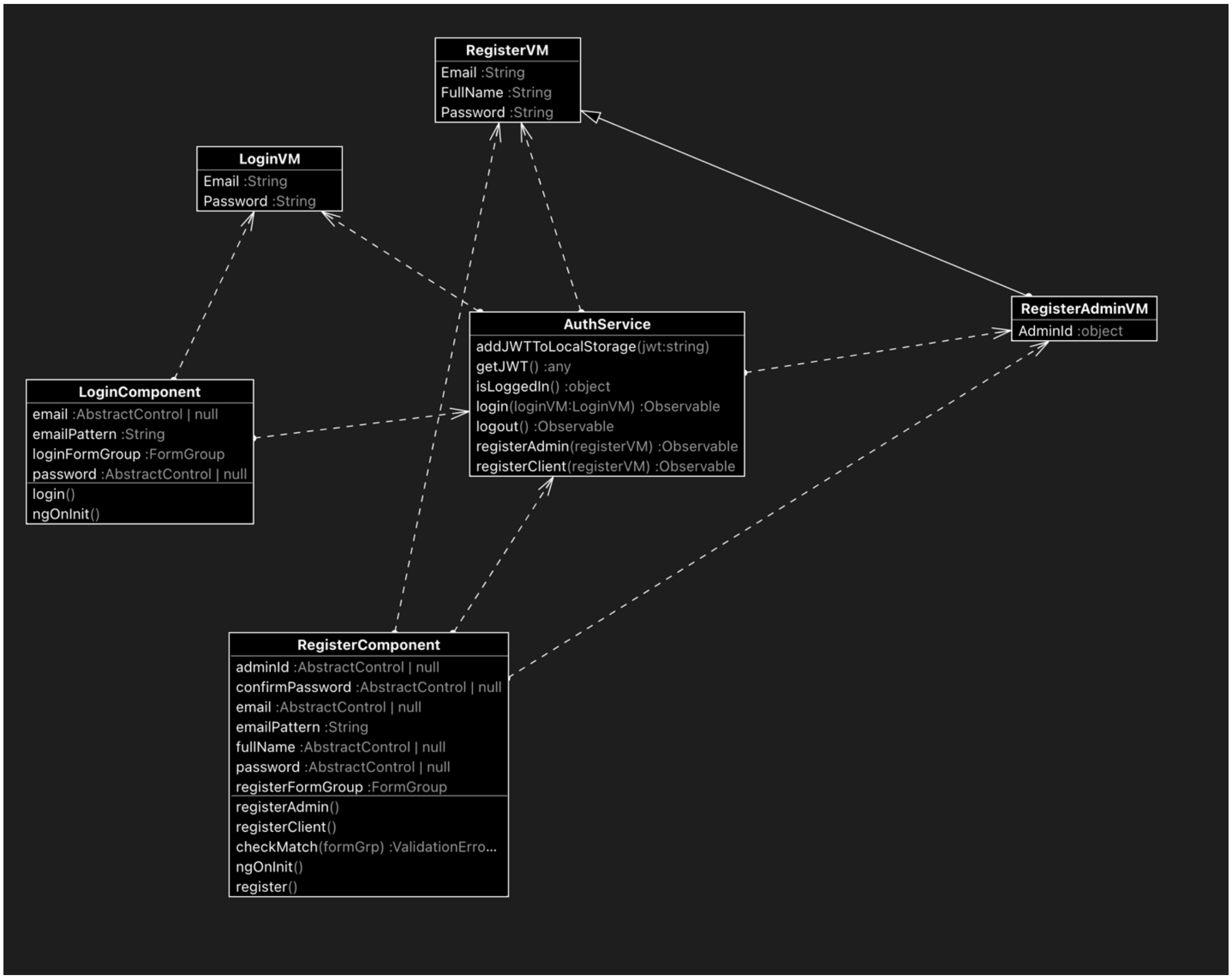
Sensitive information as CVC of the credit card is not stored and is only used transiently when processing the payment. Similarly, the passwords are not stored in the User table. Instead, it is securely handled internally by .NET's UserManager class.

2.7.6 Uniform Classes

The objects sent between the API and Client are defined classes and not JSON objects. This ensures the data is uniform and prevents attackers from adding extra properties or data to an object being sent.

3. Code Documentation – Client Side

3.1 Module: Authentication



RegisterVM
Email :String
FullName :String
Password :String
LoginVM
Email :String
Password :String
AuthService
addJWTToLocalStorage(jwt:string)
getJWT() :any
isLoggedIn() :object
login(loginVM:LoginVM) :Observable
logout() :Observable
registerAdmin(registerVM) :Observable
registerClient(registerVM) :Observable
RegisterAdminVM
AdminId :object
LoginComponent
email :AbstractControl | null
emailPattern :String
loginFormGroup :FormGroup
password :AbstractControl | null
login()
ngOnInit()
RegisterComponent
adminId :AbstractControl | null
confirmPassword :AbstractControl | null
email :AbstractControl | null
emailPattern :String
fullName :AbstractControl | null
password :AbstractControl | null
registerFormGroup :FormGroup
registerAdmin()
registerClient()
checkMatch(formGrp) :ValidationErro...
ngOnInit()
register()

The Authentication Module consists of LoginComponet, LoginVM, RegisterAdminVM, RegisterVM, Register Component, and AuthService.

When a user logs in to the application, the user will need to input their unique email account and password in the Login Page, which uses LoginComponent. The LoginVM reads information from LoginComponent and checks the email pattern by applying RegEx. The LoginComponent returns false if the email is not in the correct pattern or the password field is not filled. Otherwise, the LoginVM transfers data to AuthService where the front-end communicates with the back-end by using HttpClient Post request.

In case of not having an account, the user can only go to Register Page. By default, this page will use the RegisterComponent, which registers a user. The Register Page requires the user's unique email, passwords, and full name. All data will be read by RegisterVM or RegisterAdminVM. The RegisterComponent returns false only when either the fields are not filled, or the email does not match with the company email pattern. Otherwise, the data will be transferred to AuthService.

The AuthService will communicate with the back-end database to check the user’s authorization and update the new account.

3.1.1 Class: AuthService

Description	Where the front-end communicates with the back-end by using HttpClient Post request.
Properties	**N/A**
Methods	**registerAdmin(registerVM): Observable** public method that takes in the registerVM and stores the new administrator account locally **registerClient(registerVM): Observable** public method that takes in the registerVM and stores the new client account locally **login(loginVM:LoginVM): Observable** public method that takes in the loginVM and verifies the user account prior to either letting them into the system or throwing an error **logout(): Observable** public method that logs the user out of the system **isLoggedIn(): object** public method that checks to see whether the user has the appropriate token **getJWT(): any** public method that returns the token **addJWTToLocalStorage(jwt:string)** private method that sets a token

3.1.2 Class: RegisterComponent

Description	Accepts and validates user inputs before the data will be transferred to RegisterVM or RegisterAdminVM
Properties	**confirmPassword: AbstractControl | null** property that stores the password confirmation from the registerFormGroup **adminID: AbstractControl | null** property that stores the id that is assigned to the admin **email: AbstractControl | null** property that stores the email from the registerFormGroup **emailPattern: String** property that assigns the correct format for an email that will be accepted **fullName: AbstractControl | null** property that stores the full name from the registerFormGroup

	password: AbstractControl | null property that stores the password from the registerFormGroup **registerFormGroup: FormGroup** property that stores inputted data from the user
Methods	**checkMatch(formGrp): ValidationError** public method that checks whether “password” and “confirm password” match **registerAdmin()** public method that registers the admin account **registerClient()** public method that registers the client account **ngOnInit()** public method that is a lifecycle hook to indicate that the component is created **register()** public method that registers a new account

3.1.3 Class: LoginComponent

Description	Accepts and validates user inputs before transferring data to LoginVM
Properties	**emailPattern: String** property that assigns the correct format for an email that will be accepted **loginFormGroup: FormGroup** property that stores inputted data from the user **email: AbstractControl | null** property that stores the email from the loginFormGroup **password: AbstractControl | null** property that stores the password from the loginFormGroup

Methods	**login()**
	public method that logs the user into the account
	ngOnInit()
	public method that is a lifecycle hook to indicate that the component is created

3.1.4 Class: LoginVM

Description	Reads user input from LoginComponent
Properties	**Email: String** Property that stores the email the user inputs **Password: String** Property that stores the password the user inputs
Methods	**N/A**

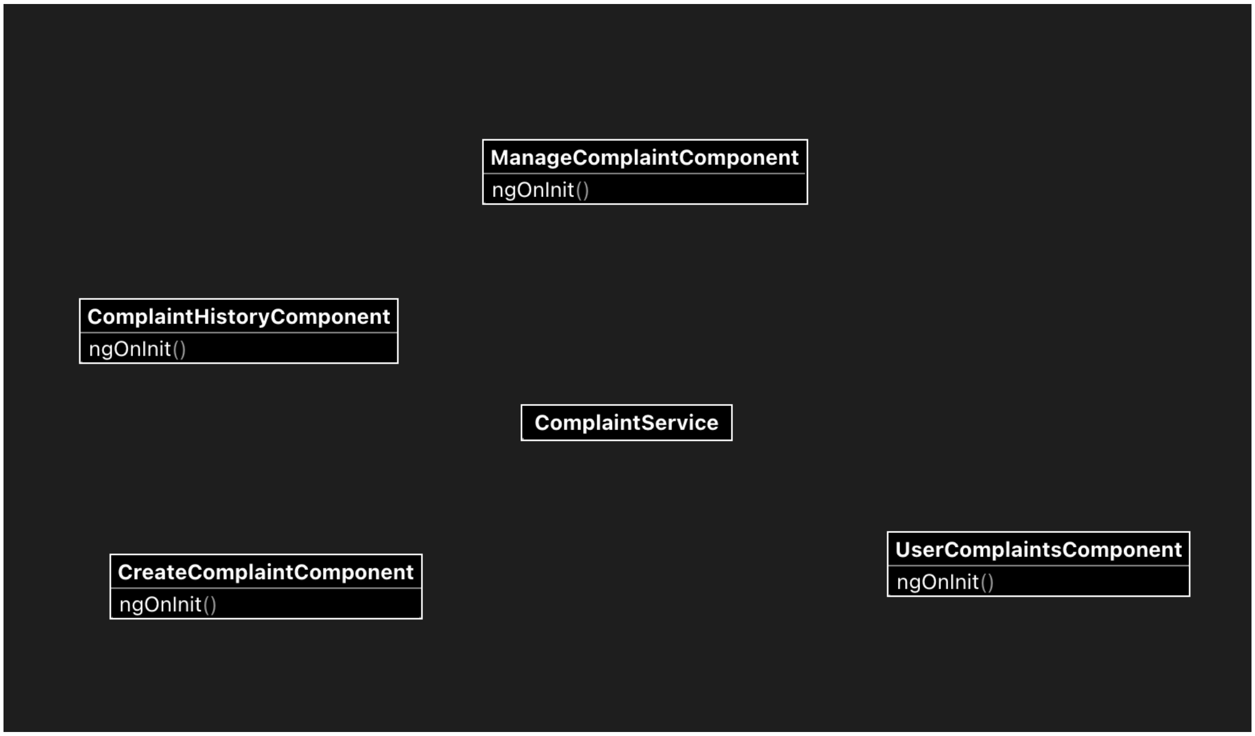
3.1.5 Class: RegisterVM

Description	Reads user inputs from RegisterComponent
Properties	**Email: String** Property that stores the email the user inputs **FullName: String** Property that stores the full name the user inputs **Password: String** Property that stores the password the user inputs
Methods	**N/A**

3.1.6 Class: RegisterAdminVM

Description	Not implemented
Properties	**AdminId: object** Property that stores the admin id the user inputs
Methods	**N/A**

3.2 Module: Complaint



This function has not been fully implemented.

3.2.1 Class: ComplaintService

Description	
Properties	**N/A**
Methods	**N/A**

3.2.2 Class: ComplaintHistoryComponent

Description	
Properties	**N/A**
Methods	**ngOnInit()** public method that is a lifecycle hook to indicate that the component is created

3.2.3 Class: CreateComplaintComponent

Description	
Properties	**N/A**

Methods	**ngOnInit()** public method that is a lifecycle hook to indicate that the component is created

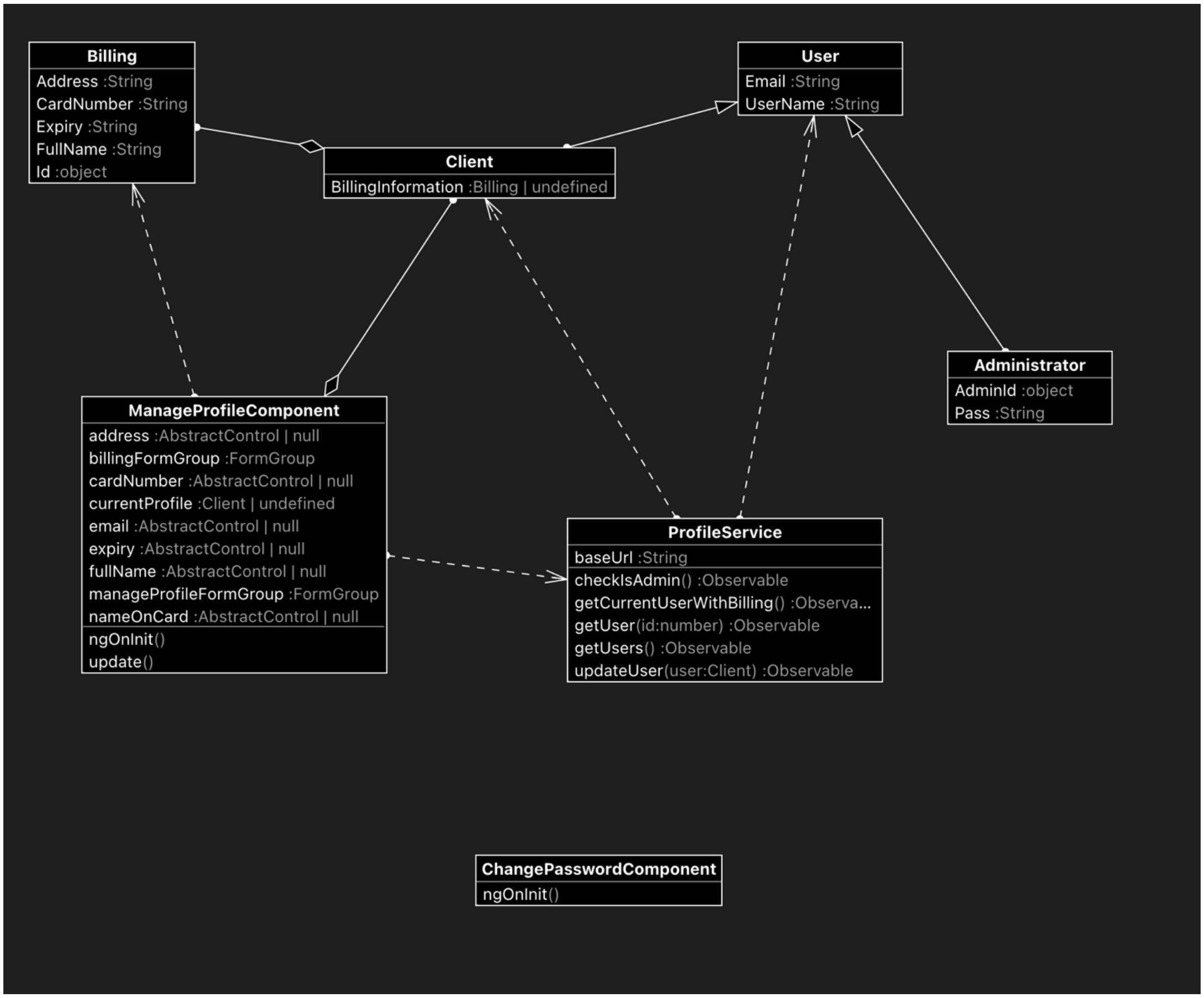
3.2.4 Class: ManageComplaintComponent

Description	
Properties	**N/A**
Methods	**ngOnInit()** public method that is a lifecycle hook to indicate that the component is created

3.2.5 Class: UserComplaintsComponent

Description	
Properties	**N/A**
Methods	**ngOnInit()** public method that is a lifecycle hook to indicate that the component is created

3.3 Module: Profile



Billing
Address :String
CardNumber :String
Expiry :String
FullName :String
Id :object
User
Email :String
UserName :String
Client
BillingInformation :Billing | undefined
Administrator
AdminId :object
Pass :String
ManageProfileComponent
address :AbstractControl | null
billingFormGroup :FormGroup
cardNumber :AbstractControl | null
currentProfile :Client | undefined
email :AbstractControl | null
expiry :AbstractControl | null
fullName :AbstractControl | null
manageProfileFormGroup :FormGroup
nameOnCard :AbstractControl | null
ngOnInit()
update()
ProfileService
baseUrl :String
checkIsAdmin() :Observable
getCurrentUserWithBilling() :Observa...
getUser(id:number) :Observable
getUsers() :Observable
updateUser(user:Client) :Observable
ChangePasswordComponent
ngOnInit()

The Profile Module consists of ManageProfileComponent and ProfileService, plus Billing, Client, User, and Administrator entities.

When a user successfully logs in to the application, the user can go to the ProfilePage, where the user can manage their profile details or change the password. However, the ChangePasswordComponent has not been fully implemented, so the user can only go to the Manage Profile Page. In this page, the user needs to fill all the fields. The input will be checked by the RegEx. If the email, account number (this application only accepts Visa and MasterCard), or expiry does not match with the pattern, the component will return false. Otherwise, the ManageProfileComponent transfers data to ProfileService where the front-end communicates with the back end by using HttpClient Post or Get request.

The ProfileService will communicate with the back-end database to update the user profile.

The Client and Administrator enitites are the extension of the User entity. The Billing entity associates with the Client entity.

3.3.1 Class: ProfileService

Description	Where the front-end communicates with the back end by using HttpClient Post or Get request.
Properties	**baseUrl: String** property that stores the URL that the user is currently on
Methods	**checkisAdmin(): Observable** public method that checks whether the user is an admin **getCurrentUserWithBilling(): Observable** public method that gets a user who must pay a bill **getUser(id:number): Observable** public method that gets a user's id number **getUsers(): Observable** public method that gets a list of all users **updateUser(user:Client): Observable** public method that gets the page to update a user's profile

3.3.2 Class: ManageProfileComponent

Description	Accepts and validates user input before transferring data to ProfileService
Properties	**address: AbstractControl | null** property that stores the users inputted address **billingFormGroup: FormGroup** property that is a formgroup with the billing information of the user **cardNumber: AbstractControl | null** property that stores the users inputted card number **currentProfile: User | undefined** property that determines whether the user has profile **email: AbstractControl | null** property that stores the users inputted email **expiry: AbstractControl | null** property that stores the users inputted card expiration date **fullName: AbstractControl | null** property that stores the users inputted full name **manageProfileFormGroup: FormGroup** property that is a formgroup with the full name and email of the user **nameOnCard: AbstractControl | null** property that stores the users inputted name on card

Methods	**ngOnInit()**
	public method that is a lifecycle hook to indicate that the component is created
	update()
	public method that updates the users profile according to what they have changed

3.3.3 Class: ChangePasswordComponent

Description	Accepts user's new password
Properties	**N/A**
Methods	**ngOnInit()** public method that is a lifecycle hook to indicate that the component is created

3.3.4 Class: Billing

Description	Stores billing information
Properties	**Address: String** property that stores the billing address **CardNumber: String** property that stores the billing card number **Expiry: String** property that stores the billing card expiration date **FullName: String** property that stores the name on the card for billing **Id: object** property that stores the billing id
Methods	**N/A**

3.3.5 Class: Client

Description	Stores client information
Properties	**BillingInformation: Billing | undefined** property that determines whether the user has billing information already
Methods	**N/A**

3.3.6 Class: User

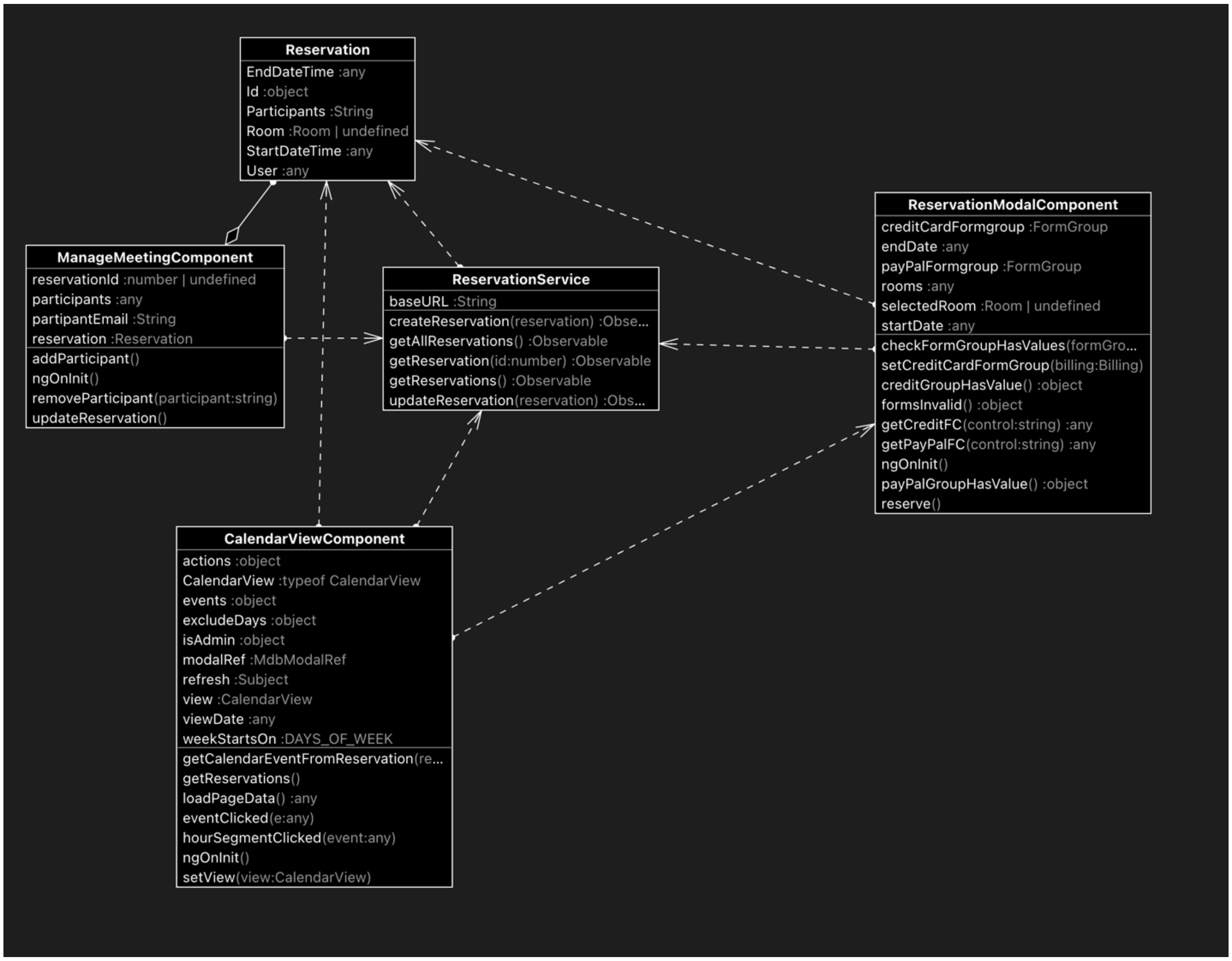
Description	Stores user information

Properties	**Email: String** property that stores the users email **UserName: String** property that stores the users username
Methods	**N/A**

3.3.7 *Class: Admin*

Description	Stores administrator information
Properties	**AdminId: object** property that stores the admins id **Pass: String** property that stores the admins password
Methods	**N/A**

3.4 Module: Reservation


Reservation
EndDateTime :any
Id :object
Participants :String
Room :Room | undefined
StartDateTime :any
User :any
ManageMeetingComponent
reservationId :number | undefined
participants :any
partipantEmail :String
reservation :Reservation
addParticipant()
ngOnInit()
removeParticipant(participant:string)
updateReservation()
ReservationService
baseURL :String
createReservation(reservation) :Obse…
getAllReservations() :Observable
getReservation(id:number) :Observable
getReservations() :Observable
updateReservation(reservation) :Obs…
ReservationModalComponent
creditCardFormgroup :FormGroup
endDate :any
payPalFormgroup :FormGroup
rooms :any
selectedRoom :Room | undefined
startDate :any
checkFormGroupHasValues(formGro…
setCreditCardFormGroup(billing:Billing)
creditGroupHasValue() :object
formsInvalid() :object
getCreditFC(control:string) :any
getPayPalFC(control:string) :any
ngOnInit()
payPalGroupHasValue() :object
reserve()
CalendarViewComponent
actions :object
CalendarView :typeof CalendarView
events :object
excludeDays :object
isAdmin :object
modalRef :MdbModalRef
refresh :Subject
view :CalendarView
viewDate :any
weekStartsOn :DAYS_OF_WEEK
getCalendarEventFromReservation(re…
getReservations()
loadPageData() :any
eventClicked(e:any)
hourSegmentClicked(event:any)
ngOnInit()
setView(view:CalendarView)

The Reservation module consists of CalendarViewComponent, Reservation, ReservationService, ManageMeetingComponent, and ReservationModalComponent.

The user can use the reservation function only if the user successfully logs in the system. After successfully logging in, the user will be automatically redirect to the Calendar View Page. In this page, the user can choose a time frame to book a reservation by clicking on the calendar. After choosing a time window, the system pops up a form where the user can select any available rooms. When the reservation is set, the data will be transferred to the ReservationService the front-end communicates with the back end by using HttpClient Post or Get request.
As same as making a reservation, to update a reservation, the user clicks to the reservation appearing on the calendar, then there is manage form will pop up and allow the user to modify. After finishing update the reservation, the data will be read by the ReservationService.

The ReservationService will communicate with the back-end database to update or create a reservation.

3.4.1 Class: ReservationService

Description	Where the front-end communicates with the back end by using HttpClient Post or Get request.
Properties	**baseURL: String** property that stores the URL that the user is currently on
Methods	**createReservation(reservation): Observable** public method that creates a reservation **getReservation(id: number): Observable** public method that gets a reservation of an inputted id **getAllReservations(): Observable** public method that gets all the reservations **getReservations(): Observable** public method that gets selected reservations updateReservation(reservation): Observable public method that submits an update for the reservations

3.4.2 Class: ReservationModalComponent

Description	Accepts user inputs
Properties	**creditCardFormgroup: FormGroup** property that is a form group of the credit card information **endDate: any** property that stores the end date of the reservation **payPalFormGroup:** property that stores the information of the paypal account **rooms: any** property that stores the information on rooms available **selectedRoom: Room | undefined** property that stores the room that is selected by the user **startDate: any** property that stores the start date of the reservation

Methods	**ngOnInit()**
	public method that is a lifecycle hook to indicate that the component is created
	
	checkFormGroupHasValues(formGroup: FormGroup)
	private method that checks to see if the form group is filed out
	
	setCreditCardFormGroup(billing: Billing)

	private method that sets the inputted information on the creditcard form group **creditGroupHasValue(): object** public method that checks to see if the group is assigned values **formsInvalid(): object** public group that checks the validity of the inputted information **getCreditFC(control: string): any** public method that gets the credit card form control **getPayPalFC(control:string): any** public method that gets the paypal form control **paypalGroupHasValue(): object** public method that checks to see if the paypal group is assigned values **reserve()** public method that makes the reservation

3.4.3 Class: CalendarViewComponent

Description	Displays calendar and accepts user inputs
Properties	**actions: object** property that determines whether there is an action on the calendar **CalendarView: typeof CalendarView** property that determines the type of calendar view **isAdmin(): object** property that stores whether the user is an admin **events: object** property that stores events on the calendar **excludeDays: object** property that stores the days of the week that will be excluded from the view **modalRef: MdbModalRef** property that acts as a template for the calendar **refresh: Subject** property that creates a new subject **view: CalendarView** property that stores a type of calendar view **viewDate: any** property that determines the way the date is viewed

	weekStartsOn: DAYS_OF_WEEK property that determines the day of the week that the week starts on
Methods	**getCalendarEventFromReservation(reservation)** private method that makes an event from the reservation **getReservations()** private method that gets all the reservations **loadPageData(): any** public method that loads the data on the page **hourSegmentClicked(event: any)** public event that selects the segment of an hour when clicked **eventClicked(e:any)** public method that determines whether an event has been clicked **ngOnInit()** public method that is a lifecycle hook to indicate that the component is created **setView(view: CalendarView)** public method that sets the view of the calendar for the user

3.4.4 Class: ManageMeetingComponent

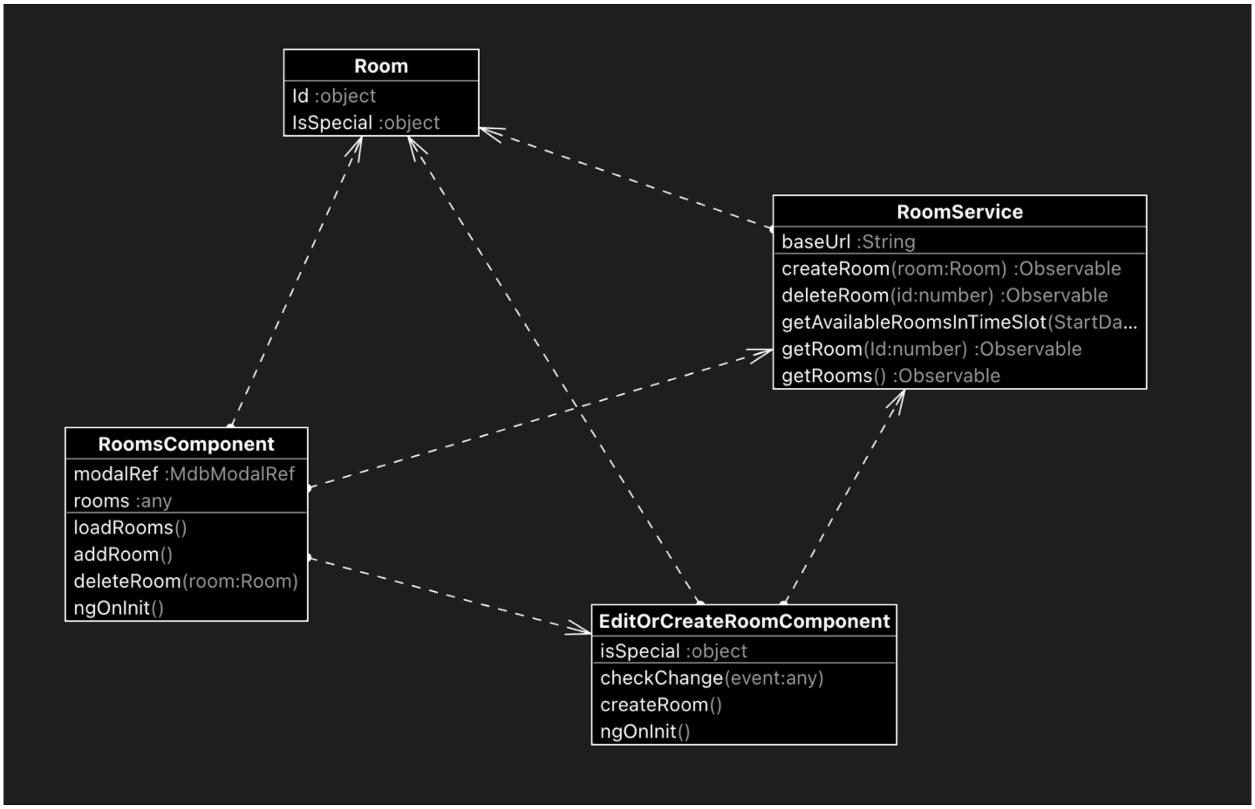
Description	Accepts user input
Properties	**reservationId: number | undefined** property that stores the id of the reservation **particiapants: any** property that lists the participants **participantEmail: String** property that lists the participant’s email **reservation: Reservation** propery that stores the reservation

Methods	**ngOnInit()**
	public method that is a lifecycle hook to indicate that the component is created
	addParticipant()
	public method that adds a participant to the meeting
	removeParticipant(participant: string)
	public method that removes participant from meeting
	updateReservation()
	public method that makes an update to the reservation

3.4.5 Class: Reservation

Description	Stores reservation information
Properties	**EndDateTime: any** property that stores the end time of the reservation **Id: object** property that stores the id of the reservation **Participants: any** property that stores the meeting participants **Room: Room | undefined** property that stores the meeting room **StartDateTime: any** property that stores the start time of the reservation **User: any** property that stores the user who made the reservation
Methods	**N/A**

3.5 Module: Room



The Room module consists of RoomService, EditOrCreateRoomComponent, RoomComponent, and Room entity.

This service is meant to be used by administrators only. The administrators can create or edit a room by using EditOrCreateRoomComponent, and this component will update changes on the database. The RoomService is where the front-end communicates with the backend database.

The RoomService will use HttpClient Post and Get requests to modify and search for rooms.

3.5.1 Class: RoomService

Description	Where the front-end communicates with the backend database.
Properties	**baseUrl: String** property that stores the URL that the user is currently on
Methods	**getAvailableRoomsInTimeSlot(StartDate: Date, EndDate: Date)** public method that chooses rooms that are available during a given time slot **getRoom(Id: number): Observable** public method that gets a room by id **getRooms(): Observable** public method that gets all of the rooms createRoom(room: Room): Observable public method that creates a room deleteRoom(id: number): Observable public method that deletes a room

3.5.2 Class: EditOrCreateRoomComponent

Description	Accepts administrator inputs to edit or create a room
Properties	**isSpecial: object** property that determines whether the room is a special room of a regular room

Methods	**ngOnInit()**
	public method that is a lifecycle hook to indicate that the component is created
	createRoom()
	public method that creates a room
	checkChange()
	public method that checks to see if a room has been changed

3.5.3 Class: RoomsComponent

Description	Accepts administrator inputs
Properties	**modalRef: MdbModalRef** property that stores the modal reference for each room **rooms: any** property that stores the rooms
Methods	**ngOnInit()** public method that is a lifecycle hook to indicate that the component is created

	loadRooms()
	public method that loads the rooms
	addRoom()
	public method that adds a room
	deleteRoom(room: Room)
	public method that deletes a room

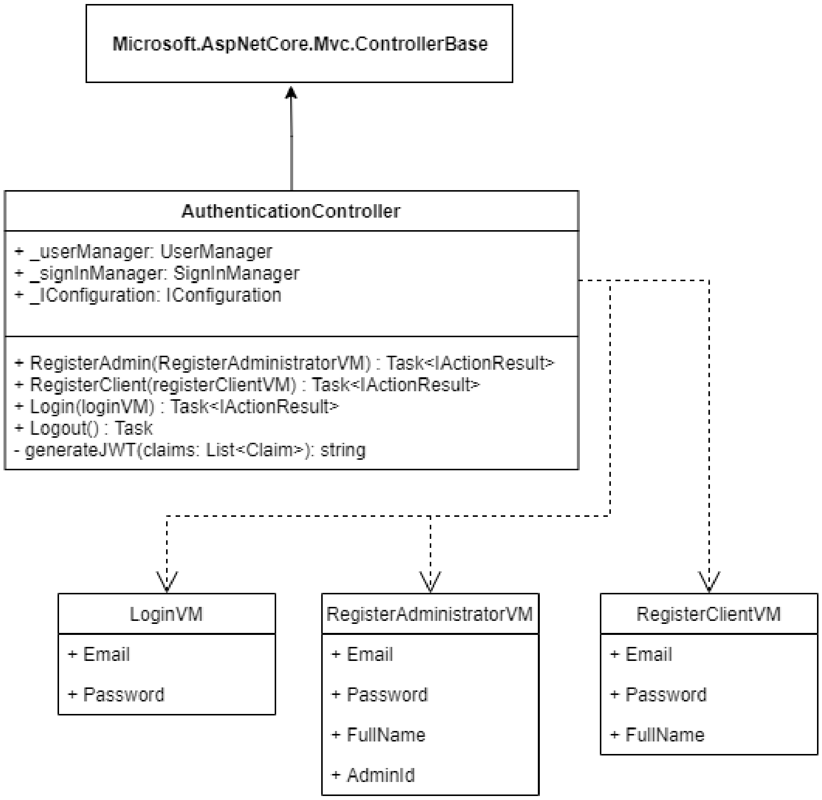
3.5.4 Class: Room

Description	Stores room information
Properties	**Id: object** property that stores the id of a room **isSpecial: object** property that stores whether a room is special
Methods	**N/A**

4. Code Documentation – Server Side

4.1 Module: Authentication

4.1.1 Module Class Diagram



Microsoft.AspNetCore.Mvc.ControllerBase
AuthenticationController
+ _userManager: UserManager
+ _signInManager: SignInManager
+ _IConfiguration: IConfiguration
+ RegisterAdmin(RegisterAdministratorVM) : Task<IActionResult>
+ RegisterClient(registerClientVM) : Task<IActionResult>
+ Login(loginVM) : Task<IActionResult>
+ Logout() : Task
- generateJWT(claims: List<Claim>): string
LoginVM
+ Email
+ Password
RegisterAdministratorVM
+ Email
+ Password
+ FullName
+ AdminId
RegisterClientVM
+ Email
+ Password
+ FullName

The figure above illustrates the relationships and interactions among the Authentication classes.

4.1.2 Class: Authentication Controller

Description	This controller receives API calls from the route “api/Auth”. This class inherits from BaseController which allows the class to check ModelState and do other API functionalities. This controller does not require the user to be authenticated to reach it.
Properties	**UserManager :** part of Microsoft.AspNetCore.Identity is used to retrieve a user, sign them up, manage passwords, tokens, and claims. **SignInManager :** part of Microsoft.AspNetCore.Identity is used to sign the user in or out. **IConfiguration :** passed to the constructor via dependency injection. It is responsible for retrieving values from config files.
Methods	**RegisterAdmin(RegisterAdministratorVM):** HTTP POST method which receives the registerAdminVM object. If the object is not valid or the user was unable to be created, it returns a *BadRequest*. Otherwise, it registers the user admin and signs them in then returns *OK*. **RegisterClient(RegisterClientVM):** HTTP POST method which receives the registerClientVM object. If the object is not valid or the user was unable to be created, it returns a *BadRequest*. Otherwise, it registers the user admin and signs them in then returns *OK*. **Login(LoginVM):** HTTP POST method which receives the loginVM object. If the object is not valid or the password does not match, *BadRequest* is returned. Otherwise, the user is signed in and *OK* is returned. **Logout():** HTTP GET method which logs the user out using the signInManager. **generateJWT(): string** Generates token

4.1.3 Class: RegisterAdministratorVM

Description	Represents a View Model for the registration object of the admin
Properties	**Email** **Password** **FullName :** first name and last name separated with a "." **AdminId :** The id number of this admin. This is not the same as the userId which exists for clients and users. This is specific to admins.
Methods	NONE

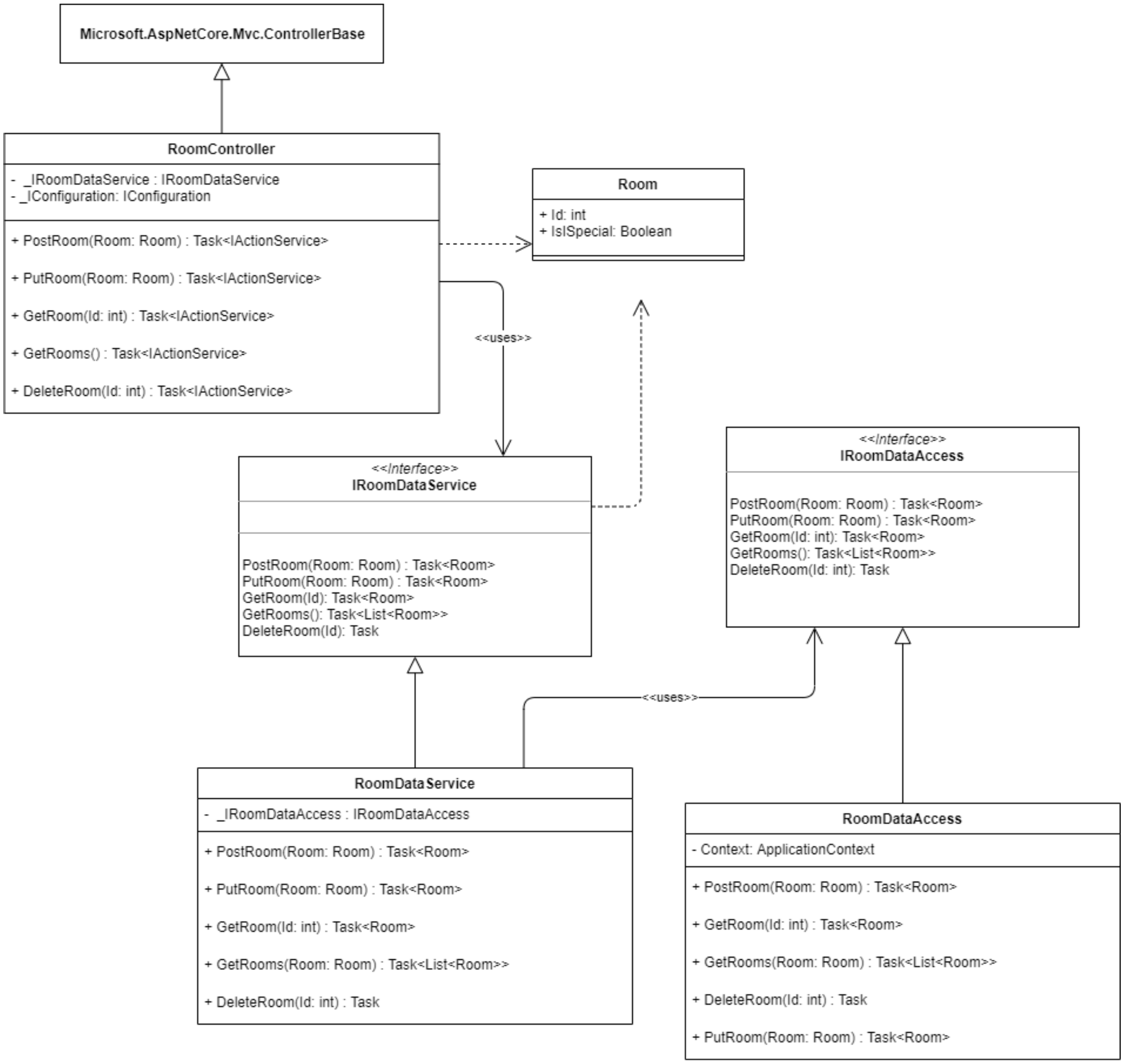
4.1.4 Class: RegisterClientVM

Description	Represents a View Model for the registration object of the client
Properties	**Email** **Password** **FullName :** first name and last name separated with a "."
Methods	NONE

4.1.5 Class: LoginVM

Description	Represents a View Model for the login of both clients and admins
Properties	**Email** **Password**
Methods	NONE

4.2 Module: Room



Microsoft.AspNetCore.Mvc.ControllerBase
RoomController
- _IRoomDataService : IRoomDataService
- _IConfiguration: IConfiguration
+ PostRoom(Room: Room) : Task<IActionService>
+ PutRoom(Room: Room) : Task<IActionService>
+ GetRoom(Id: int) : Task<IActionService>
+ GetRooms() : Task<IActionService>
+ DeleteRoom(Id: int) : Task<IActionService>
Room
+ Id: int
+ IsISpecial: Boolean
<<uses>>
<<Interface>>
IRoomDataService
PostRoom(Room: Room) : Task<Room>
PutRoom(Room: Room) : Task<Room>
GetRoom(Id): Task<Room>
GetRooms(): Task<List<Room>>
DeleteRoom(Id): Task
<<Interface>>
IRoomDataAccess
PostRoom(Room: Room) : Task<Room>
PutRoom(Room: Room) : Task<Room>
GetRoom(Id: int): Task<Room>
GetRooms(): Task<List<Room>>
DeleteRoom(Id: int): Task
<<uses>>
RoomDataService
- _IRoomDataAccess : IRoomDataAccess
+ PostRoom(Room: Room) : Task<Room>
+ PutRoom(Room: Room) : Task<Room>
+ GetRoom(Id: int) : Task<Room>
+ GetRooms(Room: Room) : Task<List<Room>>
+ DeleteRoom(Id: int) : Task
RoomDataAccess
- Context: ApplicationContext
+ PostRoom(Room: Room) : Task<Room>
+ GetRoom(Id: int) : Task<Room>
+ GetRooms(Room: Room) : Task<List<Room>>
+ DeleteRoom(Id: int) : Task
+ PutRoom(Room: Room) : Task<Room>

The figure above illustrates the relationships and interactions among the Room classes.

4.2.1 Class: RoomController

Description	This controller receives API calls from the route “api/Room”. This class inherits from BaseController which allows the class to check ModelState and do other API functionalities. This controller requires authentication to access it
Properties	**IRoomDataService:** a reference to the interface of RoomDataService which is set in the constructor via dependency injection. **IConfiguration:** a reference to the interface which allows access to configuration files. This is set in the constructor via dependency injection

Methods	**GetRoom(id): Task<ActionResult<Room>> :** HTTP GET method which returns a room using its Id. Returns 404 if no room is found with the given Id. **GetRooms(): Task<ActionResult<List<Room>>> :** HTTP GET method which returns all rooms **PostRoom(room: Room): Task<ActionResult<Room>> :** HTTP POST method which creates a new room and returns the inserted room with its Id. **PutRoom(room: Room): Task<ActionResult<Room>> :** HTTP PUT method which updates the room and returns the updated room instance. **DeleteRoom(Id): Task<ActionResult<Room>> :** HTTP DELETE method which deletes the room using its Id.

4.2.2 Class: RoomDataService

Description	This dataservice class is a control class that sits between the DataAccess layer and the API controller. This class is trusted since authentication is required before reaching it. It inherits from IRoomDataService.
Properties	**IRoomDataAccess:** a reference to the RoomDataAccess interface which is set in the constructor via dependency injection.
Methods	**GetRoom(id): Task<Room> :** Gets the room with the given Id. Asynchronously awaits the DataAccess layer to return the room. **GetRooms(): Task<<List<Room>> :** Gets all the rooms. Asynchronously awaits the DataAccess layer to return the room. **PostRoom(room: Room): Task<Room> :** Asynchronously awaits the DataAccess to insert the room. **PutRoom(room: Room): Task<Room> :** Asynchronously gets the room from the DataAccess. If the room is not in use, it will update it via the DataAccess. Otherwise, it will throw an exception **DeleteRoom(Id: int): Task<Room> :** Asynchronously gets the room from the DataAccess. If the room is not in use, it will delete it via the DataAccess. Otherwise, it will throw an exception

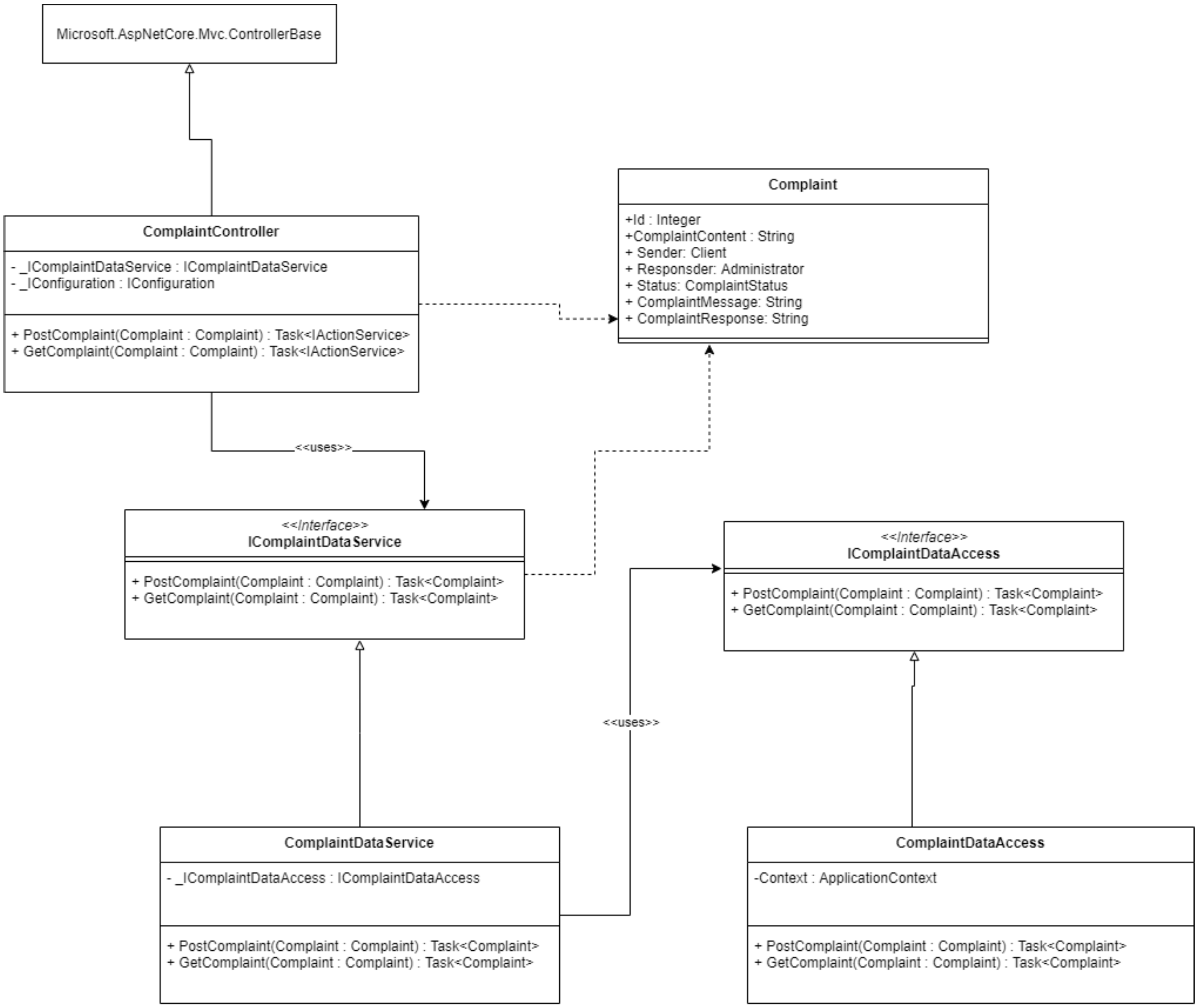
4.2.3 Class: RoomDataAccess

Description	This DataAccess is responsible for communicating with the database (through the context) and performing read and write operations asynchronously. It inherits from IRoomDataAccess interface.
Properties	**Context:** a reference to the application context which can be thought of as the database. This is set in the constructor via dependency injection. (See this link)
Methods	**GetRoom(id): Task<Room> :** Gets the room with the given Id. Asynchronously awaits the context to get the room using a LINQ expression. **GetRooms(): Task<<List<Room>> :** Gets all the rooms. Asynchronously awaits the context to get the rooms using a LINQ expression. **PostRoom(room: Room): Task<Room> :** Asynchronously inserts the room then returns the inserted room. **PutRoom(room: Room): Task<Room> :** Gets the room with the given Id then asynchronously updates it. If the room doesn't exist or it's in use, an exception is thrown. **DeleteRoom(Id: int): Task<Room> :** Gets the room with the given Id then asynchronously deletes it. If the room doesn't exist or it's in use, an exception is thrown.

4.2.4 Class: Room (Entity)

Description	This entity represents a room
Properties	**Id:** a unique integer that is the same as the Room number. Private set, public get **IsSpecial:** boolean to distinguish between regular rooms and special ones. Public set, public get
Methods	**None**

4.3 Module: Complaints



Microsoft.AspNetCore.Mvc.ControllerBase
Complaint
+Id : Integer
+ComplaintContent : String
+ Sender: Client
+ Responsder: Administrator
+ Status: ComplaintStatus
+ ComplaintMessage: String
+ ComplaintResponse: String
ComplaintController
- _IComplaintDataService : IComplaintDataService
- _IConfiguration : IConfiguration
+ PostComplaint(Complaint : Complaint) : Task<IActionService>
+ GetComplaint(Complaint : Complaint) : Task<IActionService>
<<uses>>
<<Interface>>
IComplaintDataService
+ PostComplaint(Complaint : Complaint) : Task<Complaint>
+ GetComplaint(Complaint : Complaint) : Task<Complaint>
<<Interface>>
IComplaintDataAccess
+ PostComplaint(Complaint : Complaint) : Task<Complaint>
+ GetComplaint(Complaint : Complaint) : Task<Complaint>
<<uses>>
ComplaintDataService
- _IComplaintDataAccess : IComplaintDataAccess
+ PostComplaint(Complaint : Complaint) : Task<Complaint>
+ GetComplaint(Complaint : Complaint) : Task<Complaint>
ComplaintDataAccess
-Context : ApplicationContext
+ PostComplaint(Complaint : Complaint) : Task<Complaint>
+ GetComplaint(Complaint : Complaint) : Task<Complaint>

The figure above illustrates the relationships and interactions among Complaint classes.

4.3.1 Class: ComplaintController

Description	This controller receives API calls from the route “api/Complaints”. This class inherits from BaseController which allows the class to check ModelState and do other API functionalities. This controller requires authentication to access it
Properties	**IComplaintsDataService:** a reference to the interface of ComplaintDataService which is set in the constructor via dependency injection. **IConfiguration:** a reference to the interface which allows access to configuration files. This is set in the constructor via dependency injection.
Methods	**GetComplaint(complaint: Complaint): Task<ActionResult<List<Complaint>>> :** HTTP GET method which returns all Complaints **PostComplaint(complaint: Complaint): Task<ActionResult<Complaint>> :** HTTP POST method which creates a new complaint and returns the inserted complaint with its Id.

4.3.2 Class: ComplaintDataService

Description	This dataservice class is a control class that sits between the DataAccess layer and the API controller. This class is trusted since authentication is required before reaching it. It inherits from IComplaintDataService.
Properties	**IComplaintDataAccess:** a reference to the ComplaintDataAccess interface which is set in the constructor via dependency injection.
Methods	**GetComplaint(complaint: Complaint): Task<ActionResult<List<Complaint>>> :** HTTP GET method which returns all Complaints **PostComplaint(complaint: Complaint): Task<ActionResult<Complaint>> :** HTTP POST method which creates a new complaint and returns the inserted complaint with its Id.

4.3.3 Class: ComplaintDataAccess

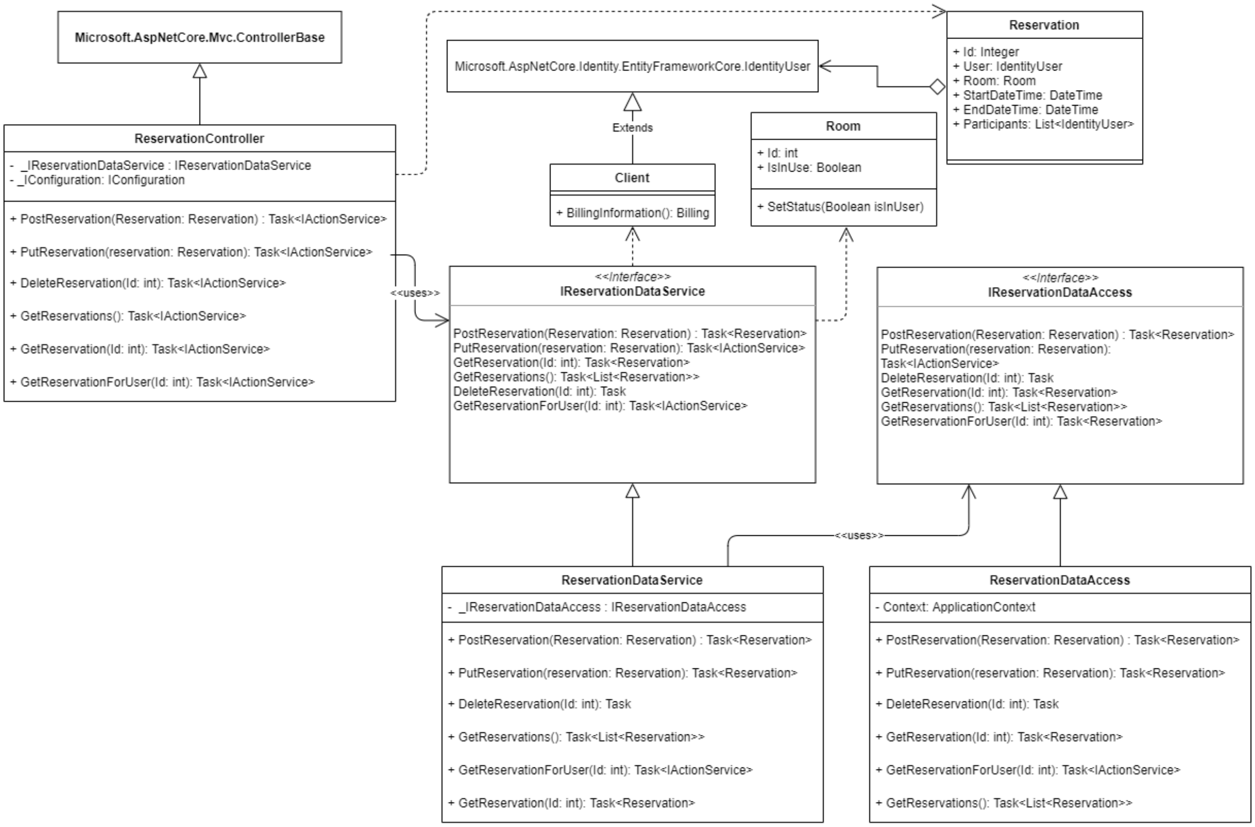
Description	This DataAccess is responsible for communicating with the database (through the context) and performing read and write operations asynchronously. It inherits from IComplaintDataAccess interface.
Properties	**Context:** a reference to the application context which can be thought of as the database. This is set in the constructor via dependency injection. (See this link)
Methods	**GetComplaint(complaint: Complaint): Task<ActionResult<List<Complaint>>> :** HTTP GET method which returns all Complaints

	PostComplaint(complaint: Complaint): Task<ActionResult<Complaint>> : HTTP POST method which creates a new complaint and returns the inserted complaint with its Id.

4.3.4 Class: Complaint (Entity)

Description	This entity represents a complaint
Properties	**Id:** a unique integer that is the same as the complaint number. Private set, public get **Sender: Client** Holds information of sender. Private set, public get **Responder: Administrator** Holds information of responder. Private set, public get **Status: ComplaintStatus** Holds complaint status, which can be pending, open, or close. Private set, public get **ComplaintMessage: String** Holds complaint content. Private set, public get. **ComplaintResponse: String** Holds complaint response. Private set, public get.
Methods	**None**

4.4 Module: Reservation



Microsoft.AspNetCore.Mvc.ControllerBase
ReservationController
- _IReservationDataService : IReservationDataService
- _IConfiguration: IConfiguration
+ PostReservation(Reservation: Reservation) : Task<IActionService>
+ PutReservation(reservation: Reservation): Task<IActionService>
+ DeleteReservation(Id: int): Task<IActionService>
+ GetReservations(): Task<IActionService>
+ GetReservation(Id: int): Task<IActionService>
+ GetReservationForUser(Id: int): Task<IActionService>
<<uses>>
Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUser
Extends
Client
+ BillingInformation(): Billing
Room
+ Id: int
+ IsInUse: Boolean
+ SetStatus(Boolean isInUser)
Reservation
+ Id: Integer
+ User: IdentityUser
+ Room: Room
+ StartDateTime: DateTime
+ EndDateTime: DateTime
+ Participants: List<IdentityUser>
<<Interface>>
IReservationDataService
PostReservation(Reservation: Reservation) : Task<Reservation>
PutReservation(reservation: Reservation): Task<IActionService>
GetReservation(Id: int): Task<Reservation>
GetReservations(): Task<List<Reservation>>
DeleteReservation(Id: int): Task
GetReservationForUser(Id: int): Task<IActionService>
<<Interface>>
IReservationDataAccess
PostReservation(Reservation: Reservation) : Task<Reservation>
PutReservation(reservation: Reservation):
Task<IActionService>
DeleteReservation(Id: int): Task
GetReservation(Id: int): Task<Reservation>
GetReservations(): Task<List<Reservation>>
GetReservationForUser(Id: int): Task<Reservation>
<<uses>>
ReservationDataService
- _IReservationDataAccess : IReservationDataAccess
+ PostReservation(Reservation: Reservation) : Task<Reservation>
+ PutReservation(reservation: Reservation): Task<Reservation>
+ DeleteReservation(Id: int): Task
+ GetReservations(): Task<List<Reservation>>
+ GetReservationForUser(Id: int): Task<IActionService>
+ GetReservation(Id: int): Task<Reservation>
ReservationDataAccess
- Context: ApplicationContext
+ PostReservation(Reservation: Reservation) : Task<Reservation>
+ PutReservation(reservation: Reservation): Task<Reservation>
+ DeleteReservation(Id: int): Task
+ GetReservation(Id: int): Task<Reservation>
+ GetReservationForUser(Id: int): Task<IActionService>
+ GetReservations(): Task<List<Reservation>>

The figure above illustrates the relationships and interactions among the Reservation classes.

4.4.1 Class: ReservationController

Description	This controller receives API calls from the route “api/Reservation”. This class inherits from BaseController which allows the class to check ModelState and do other API functionalities. This controller requires authentication to access it
Properties	**IReservationDataService:** a reference to the interface of ReservationDataService which is set in the constructor via dependency injection. **IConfiguration:** a reference to the interface which allows access to configuration files. This is set in the constructor via dependency injection
Methods	**PostReservation(Reservation: Reservation): Task<ActionResult<Reservation>> :** HTTP POST method which creates a new reservation and returns the inserted reservation with its Id. **PutReservation(reservation: Reservation): Task<ActionResult<Reservation>>:** HTTP PUT method which updates the reservation and returns the updated reservation instance. **DeleteReservation(Id: int): Task<ActionResult<Reservation>>** HTTP DELETE method which deletes a reservation having that id. **GetReservations(): Task<ActionResult<List<Reservation>>>:** HTTP Get method returns all reservations. **GetReservation (Id: int): Task<ActionResult<Reservation>>:** HTTP Get method returns a reservation with the given id. **GetReservationForUser (Id: int): Task<ActionResult<Reservation>>:** HTTP Get method returns a reservation with the given id. This function is meant for client

4.4.2 Class: ReservationDataService

Description	This data service class is a control class that sits between the data access layer and the API controller. This class is trusted since authentication is required before reaching it. It inherits from IReservationDataService.
Properties	**IReservationDataAccess:** a reference to the ReservationDataAccess interface which is set in the constructor via dependency injection.
Methods	**GetReservation(Id: int): Task<Reservation> :** Gets the reservation with the given Id. Asynchronously awaits the data access layer to return the reservation. **GetClients(): Task<<List<Reservation>> :** Gets all reservations. Asynchronously awaits the data access layer to return the list.

	PostReservation(Reservation: Reservation): Task<Reservation> : Asynchronously awaits the DataAccess to insert the reservation. **PutReservation(reservation: Reservation): Task< Reservation>:** Asynchronously update the reservation from the DataAccess. **DeleteReservation(Id: int): Task<Reservation> :** Asynchronously gets the room from the DataAccess then deletes the reservation from the database. **GetReservationForUser (Id: int): Task<Reservation>:** Asynchronously gets the room from the DataAccess then deletes the reservation from the database. This function is meant for client

4.4.3 Class: ReservationDataAccess

Description	This DataAccess is responsible for communicating with the database (through the context) and performing read and write operations asynchronously. It inherits from IReservationDataAccess interface.
Properties	**Context:** a reference to the application context which can be thought of as the database. This is set in the constructor via dependency injection. (See this link)
Methods	**GetReservation(id): Task<Reservation> :** Gets the reservation with the given Id. Asynchronously awaits the context to get the reservation using a LINQ expression. **GetReservations(): Task<<List<Reservation>> :** Gets all reservations. Asynchronously awaits the context to get reservations using a LINQ expression. **PostReservation(Reservation: Reservation): Task<Reservation> :** Asynchronously inserts the reservation then returns the inserted reservation. **PutReservation(reservation: Reservation): Task< Reservation>:** Updates the reservation with the given Id then asynchronously updates it. If the reservation doesn't exist or it's in use, an exception is thrown. **DeleteReservation(Id: int): Task<Reservation> :** Gets the reservation with the given Id then asynchronously deletes it. If the reservation doesn't exist or it's in use, an exception is thrown. **GetReservationForUser (Id: int): Task<Reservation>:** Gets the reservation with the given Id. Asynchronously awaits the context to get the reservation using a LINQ expression. This function is meant for client

4.4.4 Class: Room (Entity)

Description	This entity represents a room
Properties	**Id:** a unique integer that is the same as the Room number. Private set, public get **IsSpecial:** boolean to distinguish between regular rooms and special ones. Public set, public get
Methods	**None**

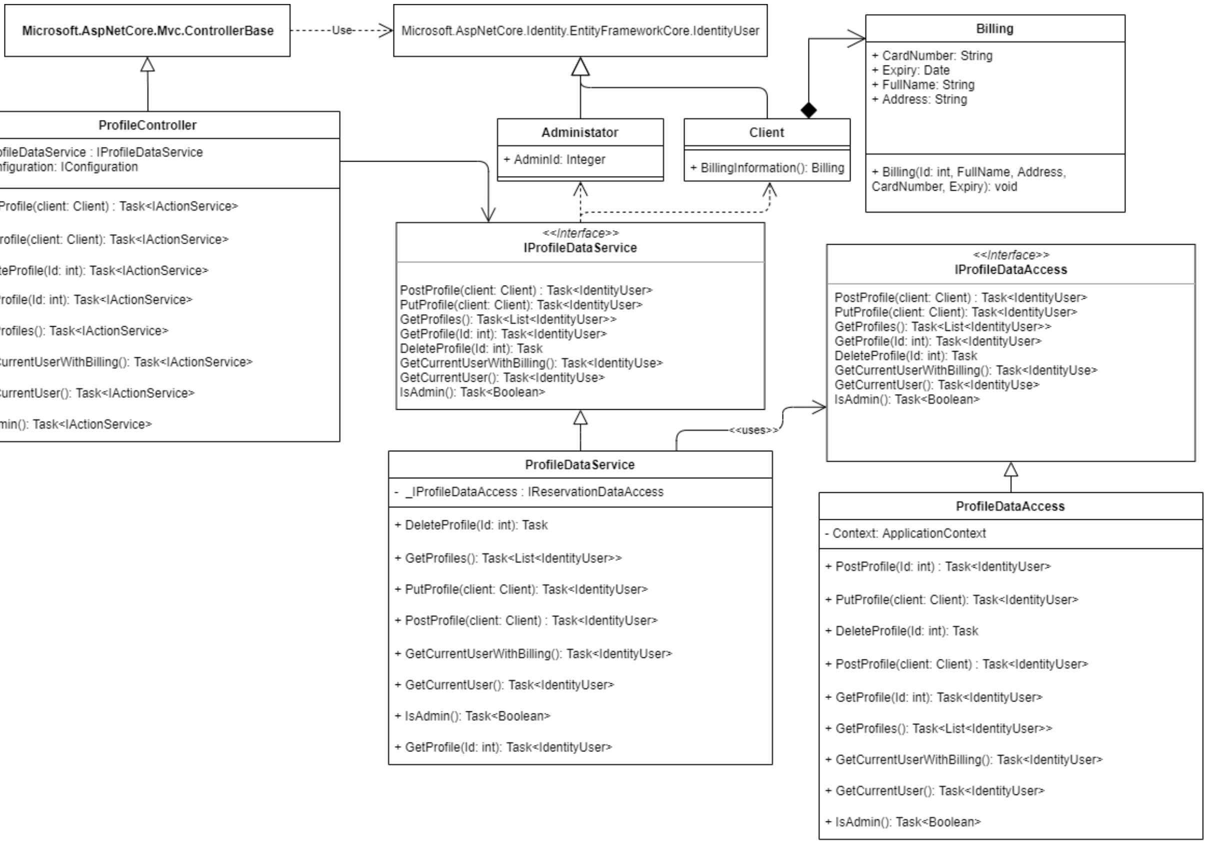
4.4.5 Class: Client (Entity)

Description	This entity represents a client
Properties	**None**
Methods	**BillingInformation(): Billing** Stores the clients billing information

4.4.6 Class: Reservation (Entity)

Description	This entity represents a reservation
Properties	**Id:** a unique integer that is the same as the Room number. Private set, public get **User:** Stores the user who makes the reservation **Room:** Stores the room that the reservation is using **StartDateTime:** Stores the time that the reservation begins **EndDateTime:** Stores the time that the reservation ends **Participants:** A list of the participants of the meeting for this reservation.
Methods	**None**

4.5 Module: Profile



Microsoft.AspNetCore.Mvc.ControllerBase
Use
Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUser
Billing
+ CardNumber: String
+ Expiry: Date
+ FullName: String
+ Address: String
+ Billing(Id: int, FullName, Address, CardNumber, Expiry): void
ProfileController
- _IProfileDataService : IProfileDataService
- _IConfiguration: IConfiguration
+ PostProfile(client: Client) : Task<IActionService>
+ PutProfile(client: Client): Task<IActionService>
+ DeleteProfile(Id: int): Task<IActionService>
+ GetProfile(Id: int): Task<IActionService>
+ GetProfiles(): Task<IActionService>
+ GetCurrentUserWithBilling(): Task<IActionService>
+ GetCurrentUser(): Task<IActionService>
+ IsAdmin(): Task<IActionService>
Administator
+ AdminId: Integer
Client
+ BillingInformation(): Billing
<<Interface>>
IProfileDataService
PostProfile(client: Client) : Task<IdentityUser>
PutProfile(client: Client): Task<IdentityUser>
GetProfiles(): Task<List<IdentityUser>>
GetProfile(Id: int): Task<IdentityUser>
DeleteProfile(Id: int): Task
GetCurrentUserWithBilling(): Task<IdentityUse>
GetCurrentUser(): Task<IdentityUse>
IsAdmin(): Task<Boolean>
<<Interface>>
IProfileDataAccess
PostProfile(client: Client) : Task<IdentityUser>
PutProfile(client: Client): Task<IdentityUser>
GetProfiles(): Task<List<IdentityUser>>
GetProfile(Id: int): Task<IdentityUser>
DeleteProfile(Id: int): Task
GetCurrentUserWithBilling(): Task<IdentityUse>
GetCurrentUser(): Task<IdentityUse>
IsAdmin(): Task<Boolean>
<<uses>>
ProfileDataService
- _IProfileDataAccess : IReservationDataAccess
+ DeleteProfile(Id: int): Task
+ GetProfiles(): Task<List<IdentityUser>>
+ PutProfile(client: Client): Task<IdentityUser>
+ PostProfile(client: Client) : Task<IdentityUser>
+ GetCurrentUserWithBilling(): Task<IdentityUser>
+ GetCurrentUser(): Task<IdentityUser>
+ IsAdmin(): Task<Boolean>
+ GetProfile(Id: int): Task<IdentityUser>
ProfileDataAccess
- Context: ApplicationContext
+ PostProfile(Id: int) : Task<IdentityUser>
+ PutProfile(client: Client): Task<IdentityUser>
+ DeleteProfile(Id: int): Task
+ PostProfile(client: Client) : Task<IdentityUser>
+ GetProfile(Id: int): Task<IdentityUser>
+ GetProfiles(): Task<List<IdentityUser>>
+ GetCurrentUserWithBilling(): Task<IdentityUser>
+ GetCurrentUser(): Task<IdentityUser>
+ IsAdmin(): Task<Boolean>

The figure above illustrates the relationships and interactions among the Profile classes.

4.5.1 Class: ProfileController

Description	This controller receives API calls from the route “api/Profile”. This class inherits from BaseController which allows the class to check ModelState and do other API functionalities. This controller requires authentication to access it
Properties	**IProfileDataService:** a reference to the interface of ProfileDataService which is set in the constructor via dependency injection. **IConfiguration:** a reference to the interface which allows access to configuration files. This is set in the constructor via dependency injection
Methods	**PostProfile(Id: int): Task<ActionResult<Profile>> :** HTTP POST method which creates a new profile of a user having Id and returns the inserted profile with its Id. **PutProfile(user: IdentityUser): Task<ActionResult<Profile>>:** HTTP PUT method which updates the profile and returns the updated profile instance. **GetProfiles(): Task<List<IdentityUser>>** HTTP GET method which returns the list of users. **GetProfile(Id: int): Task<ActionResult<IdentityUser>>** HTTP GET method which returns the user with given id. **DeleteProfile(Id: int): Task<ActionResult<Profile>>** HTTP DELETE method deleting a profile having the indicated Id. **GetCurrentUserWithBilling(): Task<IdentityUse>** Returns the current user with billing information **GetCurrentUser(): Task<IdentityUse>** Returns the current user without billing information **IsAdmin(): Task<Boolean>** Return true if current user is administrator and false if current user is not administrator.

4.5.2 Class: ProfileDataService

Description	This data service class is a control class that sits between the data access layer and the API controller. This class is trusted since authentication is required before reaching it. It inherits from IProfileDataService.
Properties	**IProfileDataAccess:** a reference to the ProfileDataAccess interface which is set in the constructor via dependency injection.
Methods	**GetProfiles(): Task<List<IdentityUser>> :** Gets the list of users. Asynchronously awaits the data access layer to return the profile of all user.

	GetProfile(Id: int): Task<IdentityUser> : Gets the user with given id. Asynchronously awaits the data access layer to return the profile. **PostProfile(client: Client): Task<IdentityUser> :** Asynchronously awaits the DataAccess to insert the profile. **PutProfile(user: Client): Task<IdentityUser>:** Asynchronously updates the profile from the data access. **DeleteProfile(Id: int): Task<Profile>:** Asynchronously deletes the profile having the indicated Id from the data access. If the profile doesn't exist, an exception is thrown. **GetCurrentUserWithBilling(): Task<IdentityUse>** Returns the current user with billing information **GetCurrentUser(): Task<IdentityUse>** Returns the current user without billing information **IsAdmin(): Task<Boolean>** Return true if current user is administrator and false if current user is not administrator.

4.5.3 Class: ProfileDataAccess

Description	This DataAccess is responsible for communicating with the database (through the context) and performing read and write operations asynchronously. It inherits from IProfileDataAccess interface.
Properties	**Context:** a reference to the application context which can be thought of as the database. This is set in the constructor via dependency injection. (See this link)
Methods	**GetProfiles(): Task<List<IdentityUser>> :** Gets the list of users. Asynchronously awaits the context to get the list using a LINQ expression. **GetProfile(Id: int): Task<IdentityUser> :** Gets the user with given Id. Asynchronously awaits the context to get the user using a LINQ expression. **PostProfile(client: Client): Task<IdentityUser> :** Asynchronously awaits the context to insert the user using a LINQ expression. **PutProfile(client: Client): Task<IdentityUser>:** Asynchronously awaits the context to update the user using a LINQ expression. **DeleteProfile(Id: int): Task<>:** Gets the profile with the given Id then asynchronously deletes it. If the profile doesn't exist or it's in use, an exception is thrown.

	GetCurrentUserWithBilling(): Task<IdentityUse> Returns the current user with billing information **GetCurrentUser(): Task<IdentityUse>** Returns the current user without billing information **IsAdmin(): Task<Boolean>** Return true if current user is administrator and false if current user is not administrator.

4.5.4 Class: Administrator (Entity)

Description	This entity represents an administrator
Properties	**AdminId: a** unique integer. Private set, public get
Methods	**None**

4.5.5 Class: Client (Entity)

Description	This entity represents a client
Properties	**None**
Methods	**BillingInformation(): Billing** Gets the billing information for the client

4.5.6 Class: Billing (Entity)

Description	This entity represents a billing
Properties	**CardNumber:** String that stores the card number being used to pay. This accepts MasterCard and Visa only. **Expiry:** String that stores the expiration date on the card being used to pay. **FullName:** String that stores the full name on the card. **Address:** String that stores the billing address of the client
Methods	**Billing(Id: int, FullName, Address, CardNumber, Expiry):** Constructs the billing information

5. Team Members Log Sheets

5.1 Eesaa Philips

Date	task	duration
7/23/21	Create Room Module	6
7/25/21	Create Complaint Module	6
7/30/21	Create Angular Architecture	5
2/8/21	Create Profile Module	6
3/8/21	Auth API	6
5/8/21	Authorization and tokens	6
13/8/21	All client screens' functionality with the API	10
	Total :	45

5.2 Huy Tran

date	task	duration
7/23/2021	Added Reservation module on Server	4 hrs
7/31/2021	Researched Angular, JavaScript, and TypeScript	4 hrs
8/6/2021	Added Profile module on Server	4 hrs
8/9/2021	Added validators on Server side and Client side	8 hrs
8/10/2021	Ran bug tests, fixed bugs on Server side	5 hrs
8/12/2021	Ran bug test, fixed bugs on Client side	5 hrs
8/13/2021	Ran bug tests, fixed bugs, worked on document, fixed UML, and final review	9 hrs
	Total :	39 hrs

5.3 Garrett Adams

date	task	duration
20210728	Researched Angular/Bootstrap	4hrs
20210731	Worked on UI	10hrs
20210803	Worked on UI	10hrs
20210805	Start CT document	5hrs
20210810	Ran security tests/fixed bugs	7hrs

20210812	Updated client-side class diagrams	4hrs
	Total :	40hrs