CSC3131 Building Systems for People: Dev-ops cycle for an E-clinic web application built with ASP.NET, React, and MongoDB

Word count: 3220 words

GitHub link: https://github.com/rolyli/EClinic

Contents

1 Introduction

EClinic is a clinic management web application, built for patients and medical service providers such as GP clinics. Using EClinic, users can make appointments for a specific time and doctor. I developed EClinic using React, ASP.Net, MongoDB, as well as behind-the-scenes tools such as Docker, Kubernetes, xUnit, GitHub Workflows, etc. In this report, I will discuss in each section the reason for choosing the tool, implementation details, as well as evaluation for the future.

2 Frontend (React)

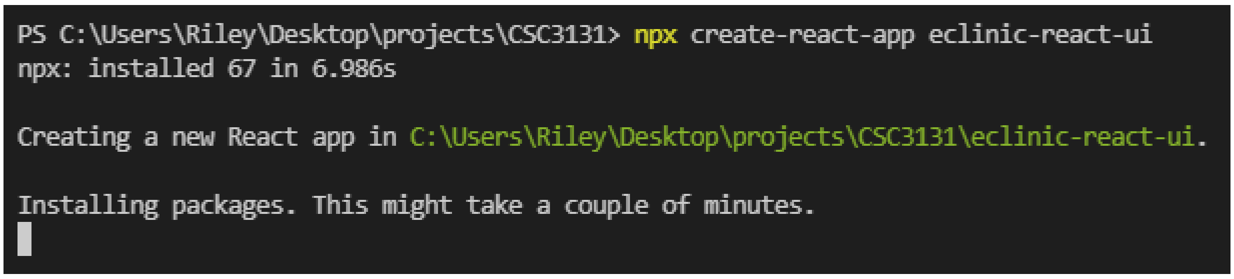
2.1 Reason for choice

React is a front-end framework for building user interfaces. The following are the reasons I chose React:

1) React is component based, and use of components reduce repeated code. There will be reused components in my EClinic, such as the message input which will be reused for sending new messages as well for replying to existing messages [1].
2) Documentation is good. Used in services such as Twitter and Facebook, React is the most popular front-end framework [2].

2.2 Implementation Details

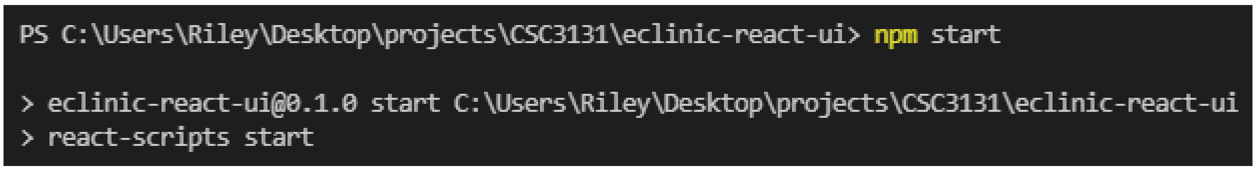
2.2.1 Initial Setup



```
PS C:\Users\Riley\Desktop\projects\CSC3131> npx create-react-app eclinic-react-ui
npx: installed 67 in 6.986s

Creating a new React app in C:\Users\Riley\Desktop\projects\CSC3131\eclinic-react-ui.

Installing packages. This might take a couple of minutes.
```

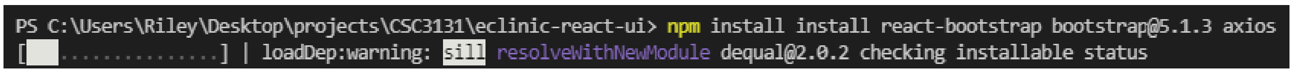
Figure 1: Using create-react-app, an integrated tool for setting up a React development environment

At this point, the backend was already developed. I created a skeleton structure for React project using *create-react-app* [3].



```
PS C:\Users\Riley\Desktop\projects\CSC3131\eclinic-react-ui> npm start

> eclinic-react-ui@0.1.0 start C:\Users\Riley\Desktop\projects\CSC3131\eclinic-react-ui
> react-scripts start
```

Figure 2: Running the development environment using npm start



```
PS C:\Users\Riley\Desktop\projects\CSC3131\eclinic-react-ui> npm install install react-bootstrap bootstrap@5.1.3 axios
[██..............] | loadDep:warning: sill resolveWithNewModule dequal@2.0.2 checking installable status
```

Figure 3: Installing required packages from NPM

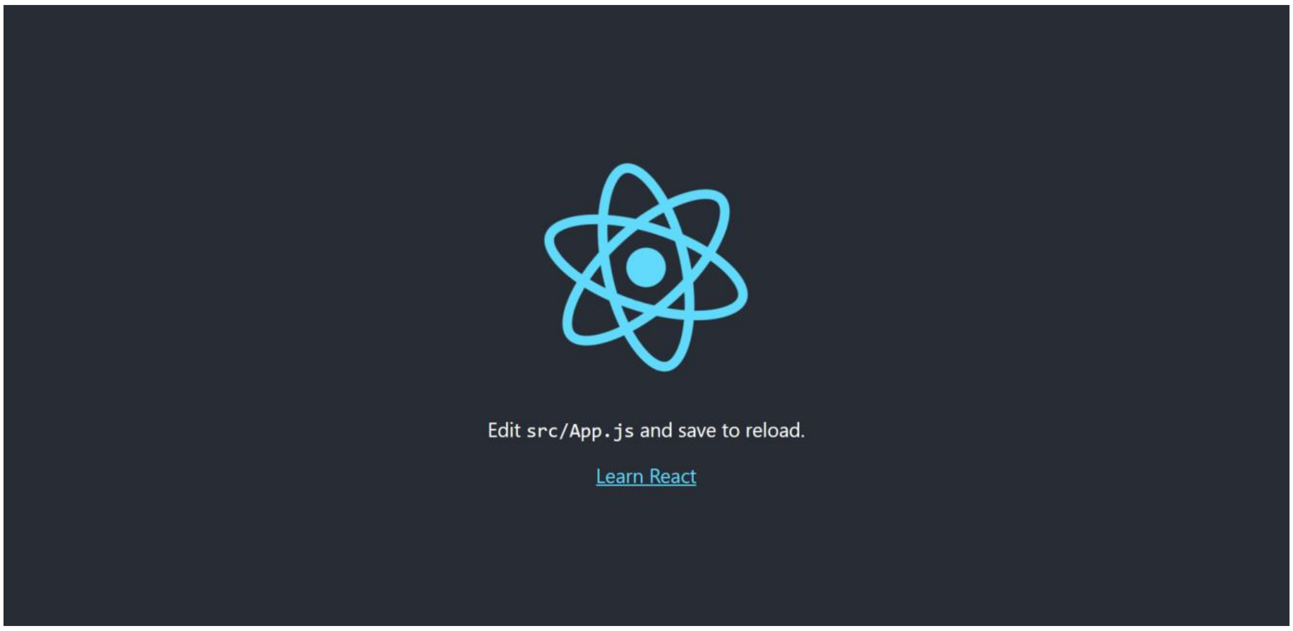
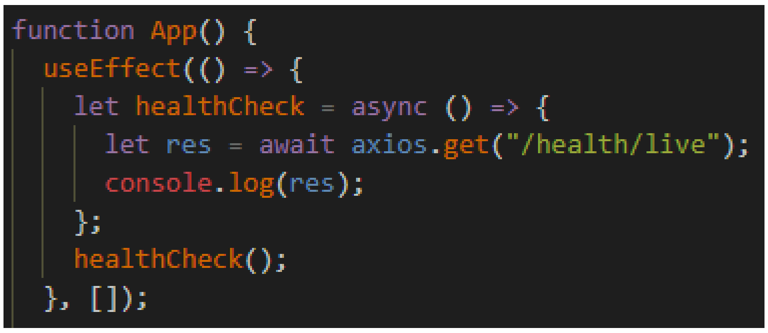


Figure 4: Default page after running create-react-app

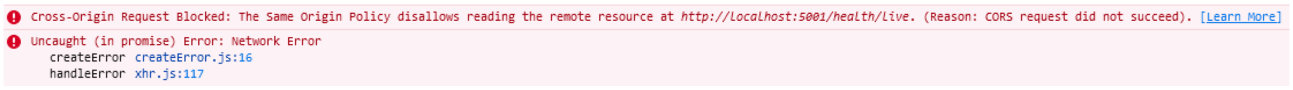
After running *npm start* and installing required packages such as Bootstrap, I was greeted with the default screen shown in Figure 4.

2.2.2 Fixing CORS issue



```
function App() {
  useEffect(() => {
    let healthCheck = async () => {
      let res = await axios.get("/health/live");
      console.log(res);
    };
    healthCheck();
  }, []);
```

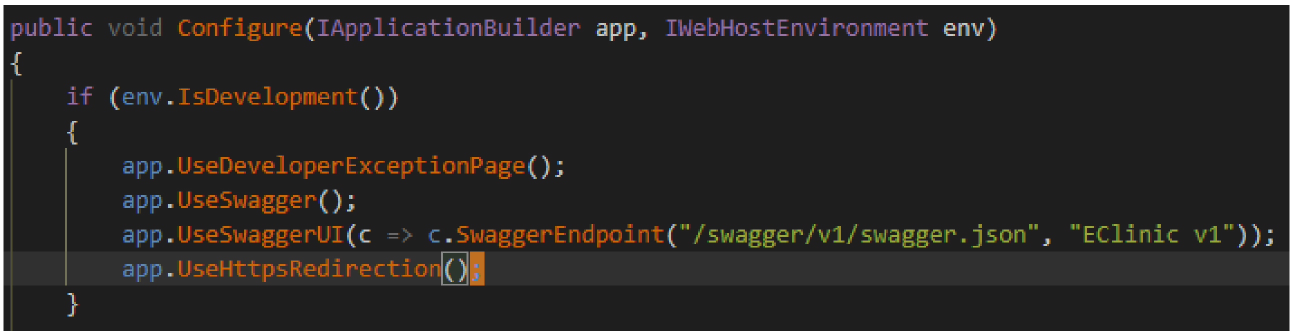
Figure 5: React hook for health check to check if the backend is live



```
Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at http://localhost:5001/health/live. (Reason: CORS request did not succeed). [Learn More]
Uncaught (in promise) Error: Network Error
    createError createError.js:16
    handleError xhr.js:117
```

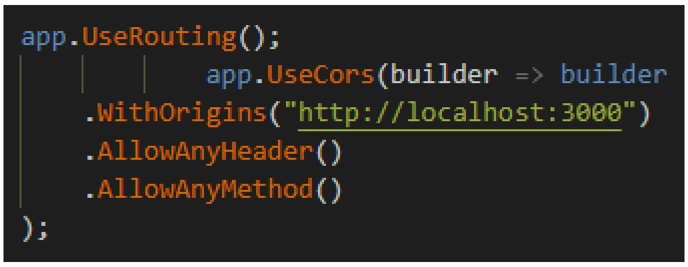
Figure 6: Health checks failed due to CORS error

In order to check that the connection to the backend was working, I used a useEffect React hook that will send a request to /health/live API endpoint when the page is loaded. However, this resulted in a CORS error (Figure 6).



```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseSwagger();
        app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "EClinic v1"));
        app.UseHttpsRedirection();
    }
```

Figure 7: Disabled Https Redirection to fix CORS issue

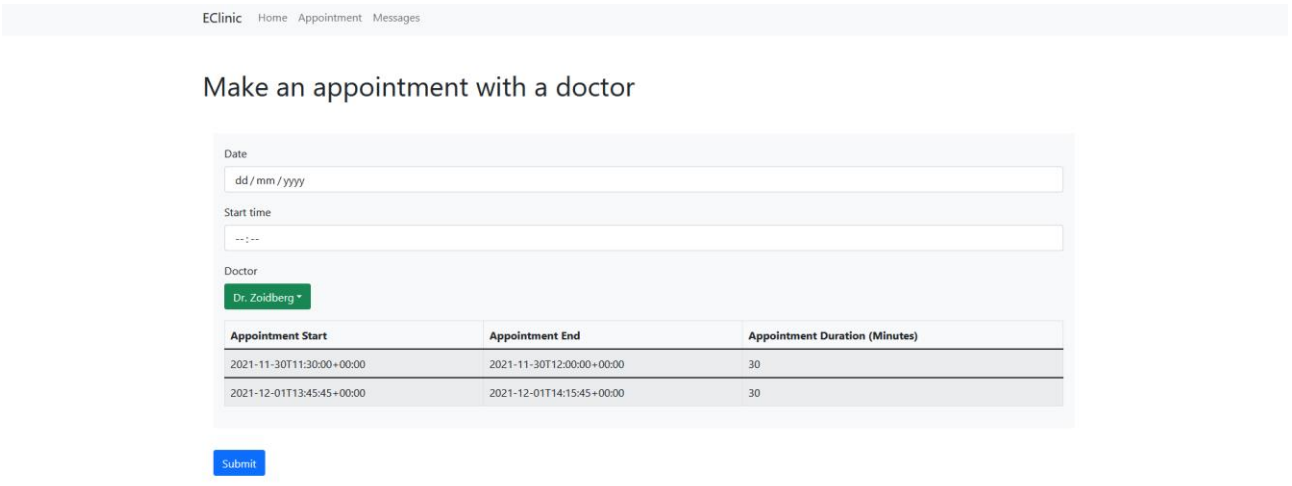


```
app.UseRouting();
            app.UseCors(builder => builder
    .WithOrigins("http://localhost:3000")
    .AllowAnyHeader()
    .AllowAnyMethod()
);
```

Figure 8: Added a CORS header for any outgoing requests

After removing HTTPS redirection and adding a CORS header to any outgoing request (Figure 7, Figure 8), the error was resolved [4].

2.2.3 Implementation of the interface and private routes



Make an appointment with a doctor

Date: dd / mm / yyyy

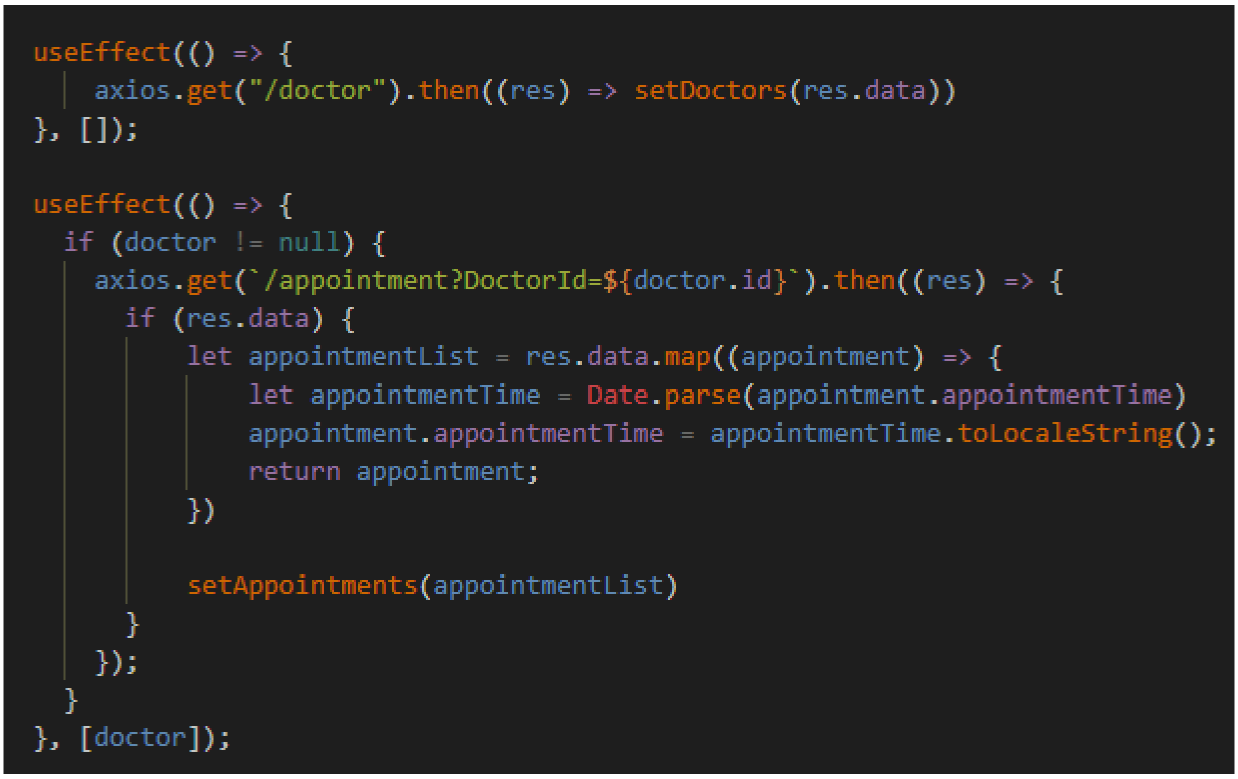
Start time: --:--

Doctor

Appointment Start	Appointment End	Appointment Duration (Minutes)
2021-11-30T11:30:00+00:00	2021-11-30T12:00:00+00:00	30
2021-12-01T13:45:45+00:00	2021-12-01T14:15:45+00:00	30

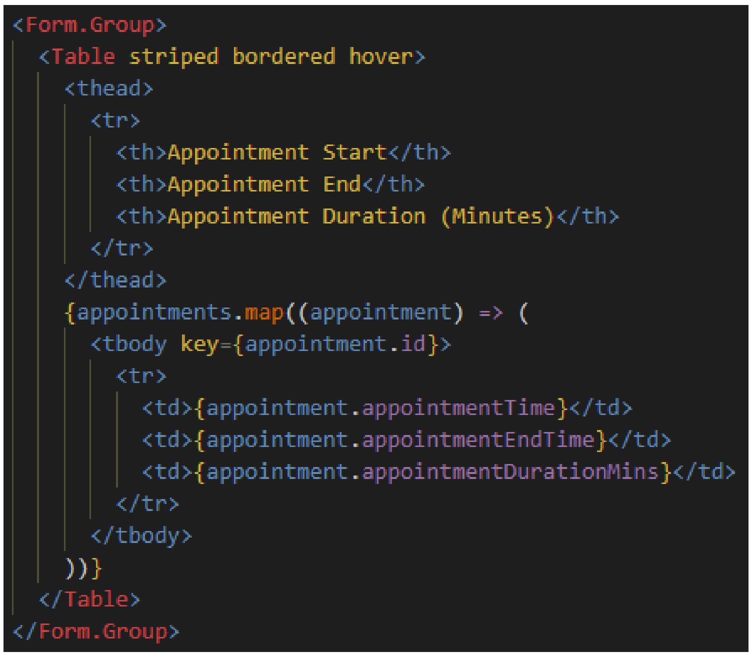
Figure 9: List of appointments for the selected doctor

For intuitive and continual design, I used Bootstrap, an open-source front end toolkit for creating mobile-first user interfaces. Although Bootstrap works primarily by providing HTML classes and styles that can be added, I installed *react-bootstrap* using NPM to turn these class names into React components (Figure 11) such as the *Form* component.



```
useEffect(() => {
  axios.get("/doctor").then((res) => setDoctors(res.data))
}, []);

useEffect(() => {
  if (doctor != null) {
    axios.get(`/appointment?DoctorId=${doctor.id}`).then((res) => {
      if (res.data) {
        let appointmentList = res.data.map((appointment) => {
            let appointmentTime = Date.parse(appointment.appointmentTime)
            appointment.appointmentTime = appointmentTime.toLocaleString();
            return appointment;
        })

        setAppointments(appointmentList)
      }
    });
  }
}, [doctor]);
```

Figure 10: useEffect hooks for retrieving list of appointments

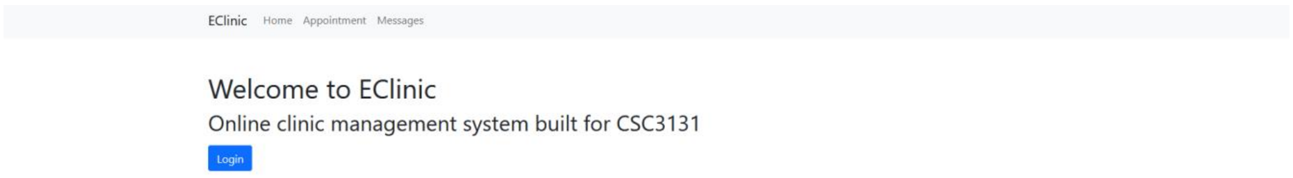


```
<Form.Group>
  <Table striped bordered hover>
    <thead>
      <tr>
        <th>Appointment Start</th>
        <th>Appointment End</th>
        <th>Appointment Duration (Minutes)</th>
      </tr>
    </thead>
    {appointments.map((appointment) => (
      <tbody key={appointment.id}>
        <tr>
          <td>{appointment.appointmentTime}</td>
          <td>{appointment.appointmentEndTime}</td>
          <td>{appointment.appointmentDurationMins}</td>
        </tr>
      </tbody>
    ))}
  </Table>
</Form.Group>
```

Figure 11: Using a .map function to iterate over list of appointments and return a tbody component

Within the Appointment page, in order to retrieve the list of doctors for the dropdown when the page is loaded, I used useEffect hooks to send a request to /doctor endpoint (Figure 10). Once a doctor has been chosen from the dropdown list, the second useEffect hook runs to retrieve list of appointments for this doctor (Figure 9). Unlike standard JavaScript approach of using an EventListener, React will re-render the page when there is a change in any of its states. Hence, clicking on a different doctor will automatically trigger a re-render of the table after the data from the backend has been assigned.

The tables are generated programmatically using a .map function (Figure 10). React uses JSX meaning that JavaScript code can be written within the HTML, allowing functionalities shown in Figure 11 where the list of appointments can be iterated to generate the rows of the table.



Welcome to EClinic

Online clinic management system built for CSC3131

Figure 12: Main page when not logged in

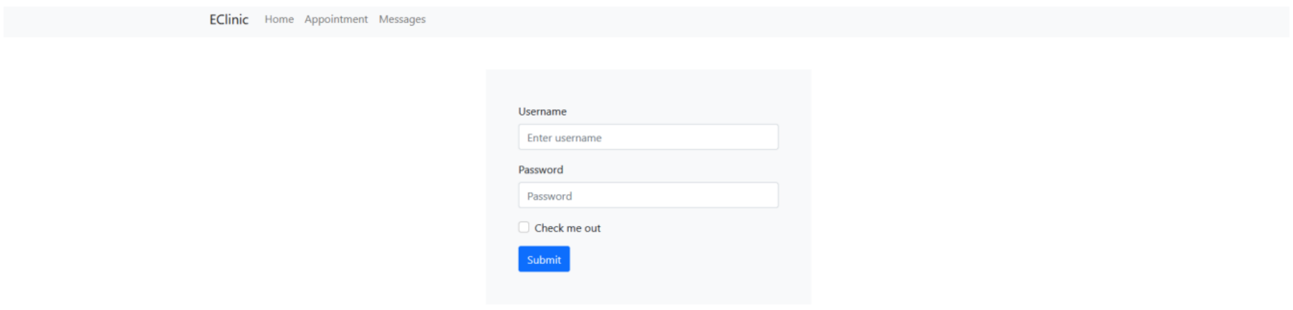
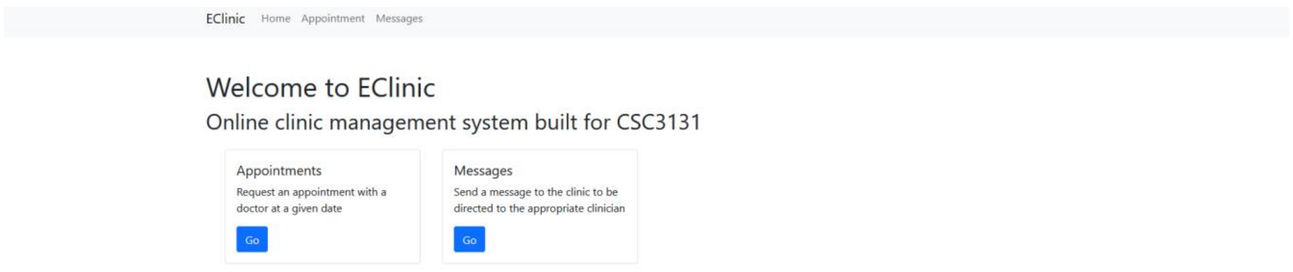


Figure 13: Login screen


EClinic Home Appointment Messages

Welcome to EClinic

Online clinic management system built for CSC3131

Appointments

Request an appointment with a doctor at a given date

Go

Messages

Send a message to the clinic to be directed to the appropriate clinician

Go

Figure 14: Main page after login

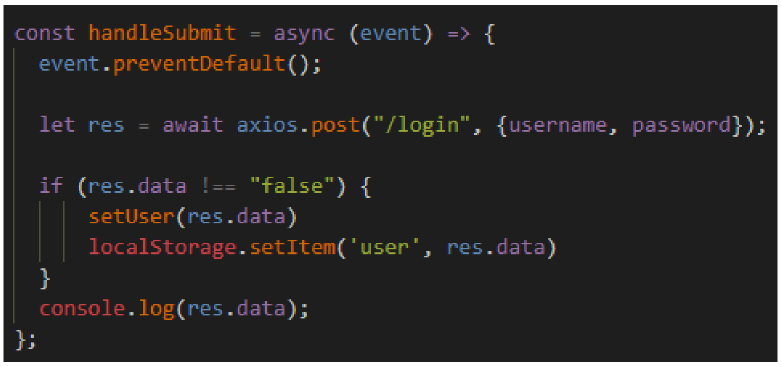
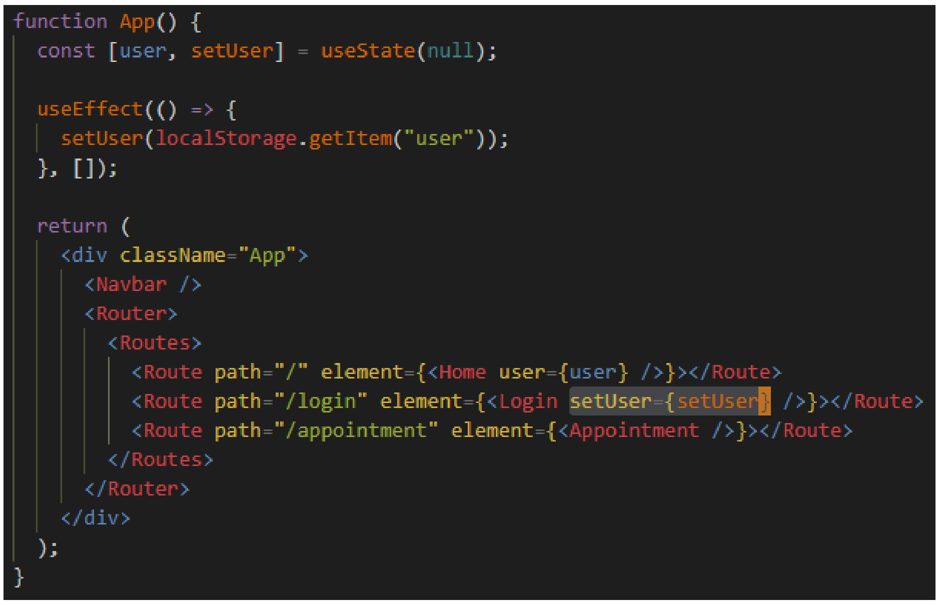

```
const handleSubmit = async (event) => {
  event.preventDefault();

  let res = await axios.post("/login", {username, password});

  if (res.data !== "false") {
      setUser(res.data)
      localStorage.setItem('user', res.data)
  }
  console.log(res.data);
};
```

Figure 15: Logic for saving user token to local storage

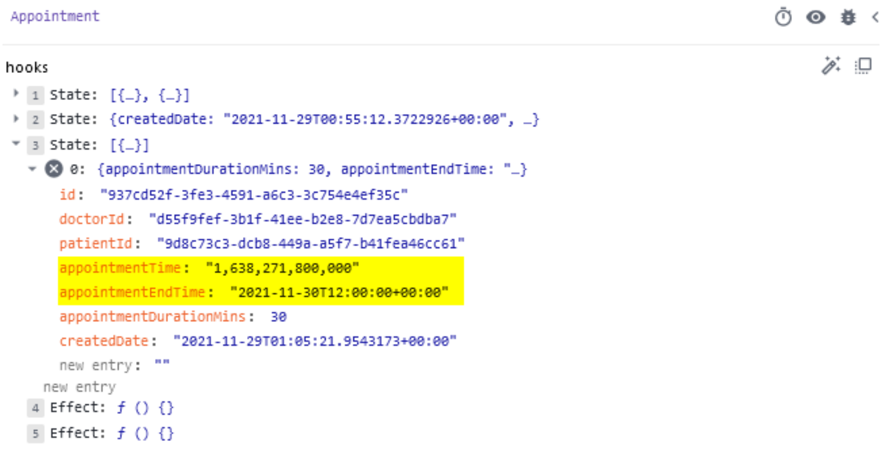
Security is important for services such as EClinic, hence private pages are only shown once the user is logged in. Clicking on any of the private areas such as Appointments will redirect users back to the login screen if the user is not logged in. I achieved these aims with React Router that provides JavaScript based routing, as well as localStorage (provided by the browser, i.e., for saving cookies). Specifically, after login, the user information is saved within memory using setUser (Figure 15). setUser assigns the returned JSON data from the backend to the *user* state that's declared with useState React hook. The setUser method is made available to the Appointment component by passing it as a 'prop' (highlighted in Figure 16).



```
function App() {
  const [user, setUser] = useState(null);

  useEffect(() => {
    setUser(localStorage.getItem("user"));
  }, []);

  return (
    <div className="App">
      <Navbar />
      <Router>
        <Routes>
          <Route path="/" element={<Home user={user} />}></Route>
          <Route path="/login" element={<Login setUser={setUser} />}></Route>
          <Route path="/appointment" element={<Appointment />}></Route>
        </Routes>
      </Router>
    </div>
  );
}
```

Figure 16: Main page showing useState hook for user state as well as drilling down setUser method to child routes

2.3 Evaluation



```
Appointment

hooks
▸ 1 State: [{…}, {…}]
▸ 2 State: {createdDate: "2021-11-29T00:55:12.3722926+00:00", …}
▾ 3 State: [{…}]
  ▾ 0: {appointmentDurationMins: 30, appointmentEndTime: "…}
      id: "937cd52f-3fe3-4591-a6c3-3c754e4ef35c"
      doctorId: "d55f9fef-3b1f-41ee-b2e8-7d7ea5cbdba7"
      patientId: "9d8c73c3-dcb8-449a-a5f7-b41fea46cc61"
      appointmentTime: "1,638,271,800,000"
      appointmentEndTime: "2021-11-30T12:00:00+00:00"
      appointmentDurationMins: 30
      createdDate: "2021-11-29T01:05:21.9543173+00:00"
      new entry: ""
    new entry
  4 Effect: ƒ () {}
  5 Effect: ƒ () {}
```

Figure 17: Error in converting from UTC format

Some aspects of the user interface are unfriendly, such as the appointment list displaying dates in UTC format. I attempted to convert this to more friendly format using toLocaleString() (Figure 10) however it resulted a strange output of “1,638,271,800,00” when converting from “2021-11-30T:11:30:00” (Figure 17). In addition, an actual calendar showing available dates and times, instead of a table will be more intuitive to use.

Login functionality is also incomplete. In the future, sending and parsing JWT tokens that contain encrypted information about the user will be more secure than the current approach that will show private routes when the server sends a Boolean for confirming that the user exists.

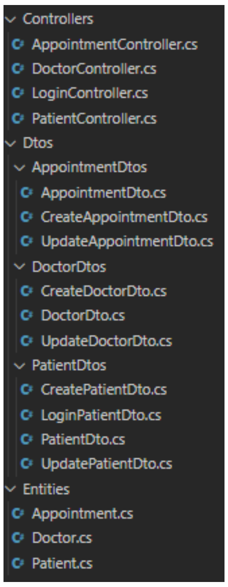
3 Backend (ASP.Net)

3.1 Reason for choice

The following are the reasons I chose ASP.NET for the backend:

1) Compared to alternatives such as NodeJS, ASP.NET is strongly typed and object oriented. Although initial setup of the objects and object-oriented patterns can be time-consuming, they help reduce difficult-to-maintain source code.
2) ASP.NET boilerplate strongly supports data transfer object [5], contributing to agile development where, in the future, internal data structures can be modified (e.g., another field is added to an object) without breaking client service layer.

3.2 Implementation Details

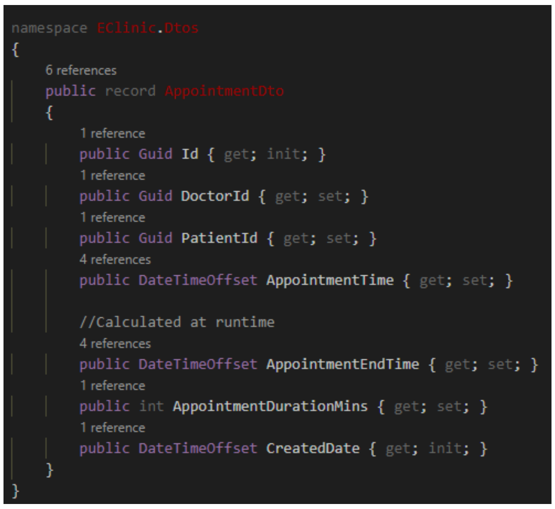


```
Controllers
  AppointmentController.cs
  DoctorController.cs
  LoginController.cs
  PatientController.cs
Dtos
  AppointmentDtos
    AppointmentDto.cs
    CreateAppointmentDto.cs
    UpdateAppointmentDto.cs
  DoctorDtos
    CreateDoctorDto.cs
    DoctorDto.cs
    UpdateDoctorDto.cs
  PatientDtos
    CreatePatientDto.cs
    LoginPatientDto.cs
    PatientDto.cs
    UpdatePatientDto.cs
Entities
  Appointment.cs
  Doctor.cs
  Patient.cs
```

Figure 18: Project hierarchy featuring MVC including DTOs

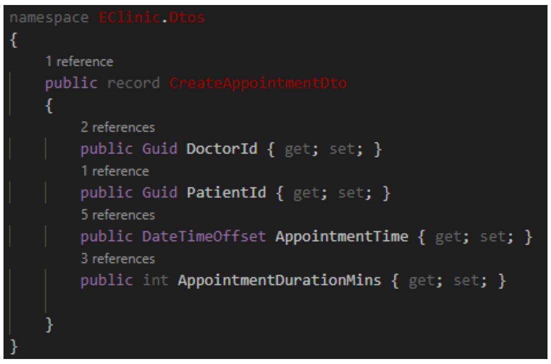
ASP.NET is a C# platform for developing web applications supporting web architectures such as MVC (Model-View-Controller). Since I used React to render the UI client-side, I configured the backend server to function as a REST API with CRUD (Create, Read, Update, Delete) methods. Through web request methods such as GET, POST, etc, requests can be made to the server from clients such as React to retrieve and render the appropriate data.

3.2.1 DTOs (Data Transfer Objects)



```
namespace EClinic.Dtos
{
    6 references
    public record AppointmentDto
    {
        1 reference
        public Guid Id { get; init; }
        1 reference
        public Guid DoctorId { get; set; }
        1 reference
        public Guid PatientId { get; set; }
        4 references
        public DateTimeOffset AppointmentTime { get; set; }

        //Calculated at runtime
        4 references
        public DateTimeOffset AppointmentEndTime { get; set; }
        1 reference
        public int AppointmentDurationMins { get; set; }
        1 reference
        public DateTimeOffset CreatedDate { get; init; }
    }
}
```

Figure 19: Base DTO

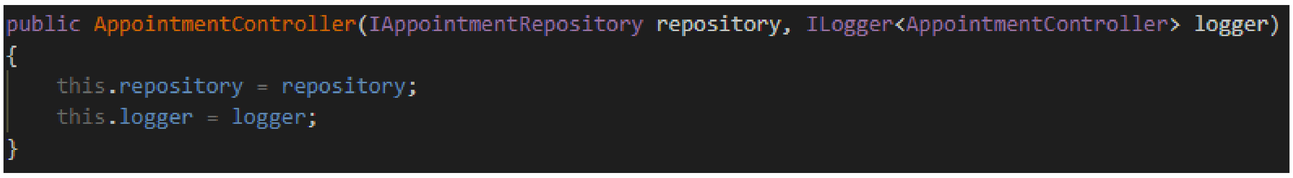


```
namespace EClinic.Dtos
{
    1 reference
    public record CreateAppointmentDto
    {
        2 references
        public Guid DoctorId { get; set; }
        1 reference
        public Guid PatientId { get; set; }
        5 references
        public DateTimeOffset AppointmentTime { get; set; }
        3 references
        public int AppointmentDurationMins { get; set; }

    }
}
```

Figure 20: CreateAppointment DTO

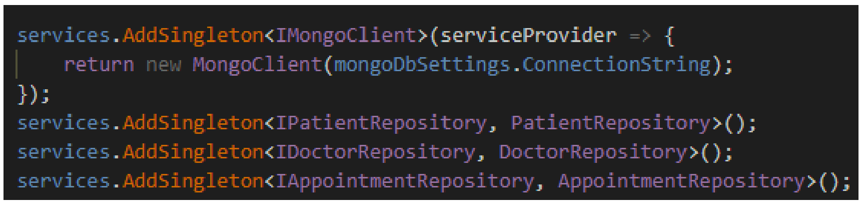
I used DTOs (Data transfer objects) instead of only entities as a data type for sending and retrieving data. DTOs decouple the internal data structure (i.e., entity) of the application to the clients. Hence, future modifications to the internal data structure can be made without affecting the contract to the client, contributing to maintainability and scalability. For this reason, the CreateAppointment DTO (Figure 20) features only some of the parameters of the base DTO (Figure 19), as only some of the parameters are needed by the client to the REST API to create an appointment. Another advantage of DTOs is that they are record types and hence support value-based comparisons, which is useful for cases such as login when validating users against an existing record [6].

3.2.2 Dependency Injection for classes using singletons



```
public AppointmentController(IAppointmentRepository repository, ILogger<AppointmentController> logger)
{
    this.repository = repository;
    this.logger = logger;
}
```

Figure 21: Class constructor for AppointmentController



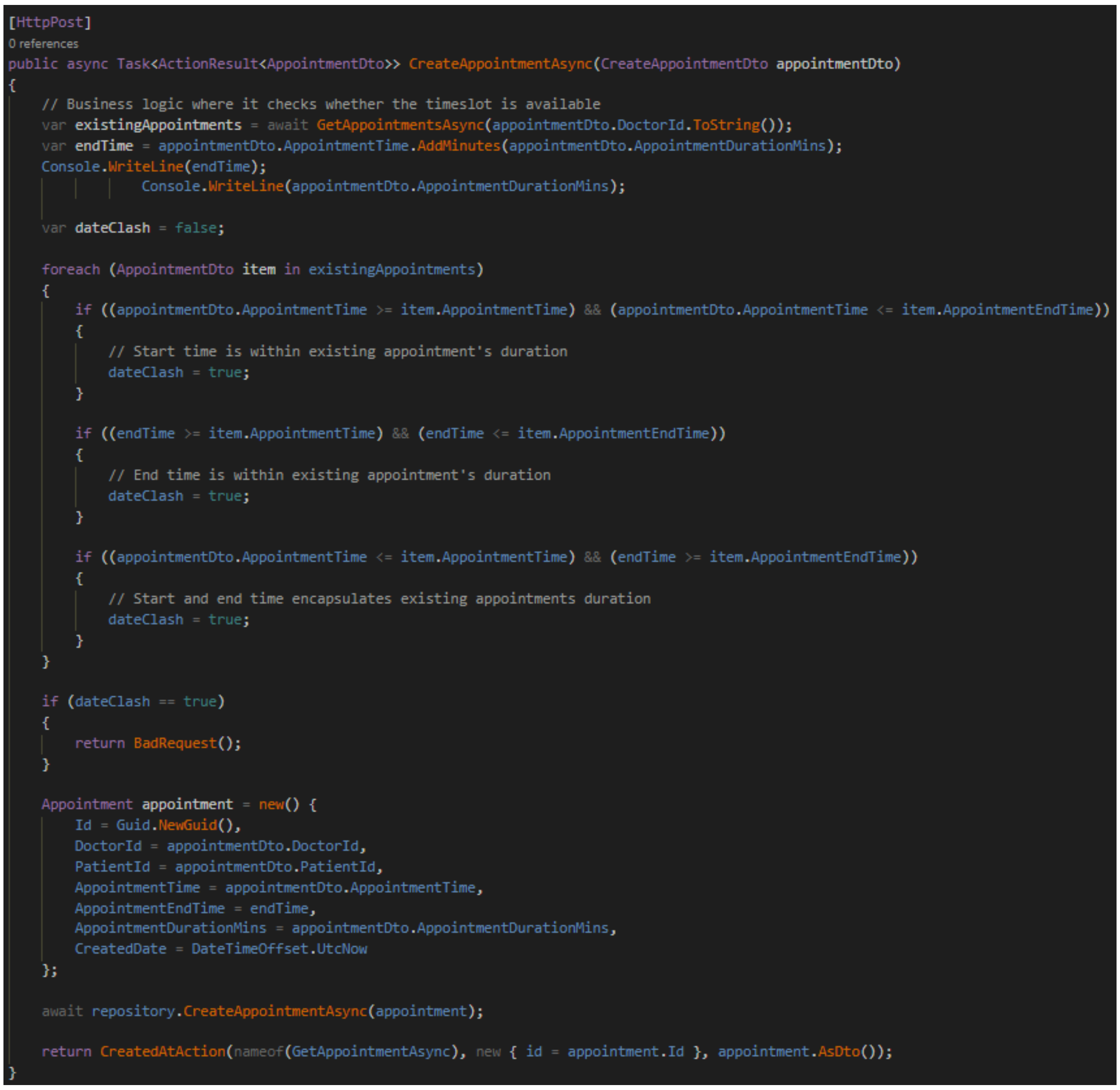
```
services.AddSingleton<IMongoClient>(serviceProvider => {
    return new MongoClient(mongoDbSettings.ConnectionString);
});
services.AddSingleton<IPatientRepository, PatientRepository>();
services.AddSingleton<IDoctorRepository, DoctorRepository>();
services.AddSingleton<IAppointmentRepository, AppointmentRepository>();
```

Figure 22: Dependency injection using AddSingleton method

ASP.NET is object oriented, which means classes are instantiated using the class constructor. In the constructor for AppointmentController, *repository* is a *dependency* that gets passed as a parameter. However, this would usually cause multiple instantiations of the repository, which wastes resources. Requiring multiple dependencies means that unit-testing is harder as the dependencies would need to be mocked (See section 13 on unit testing). Additionally, depending on an a hard-coded dependency is hard to maintain as replacing the repository to a different implementation would require modification of the controller code. Hence, dependency injection solves this issue by decoupling the class from its dependencies [6], instead depending on an abstraction or an interface, in this case the IAppointmentRepository. The dependency is injected using AddSingleton method (Figure 22), specifying the interface as well as its implementation.

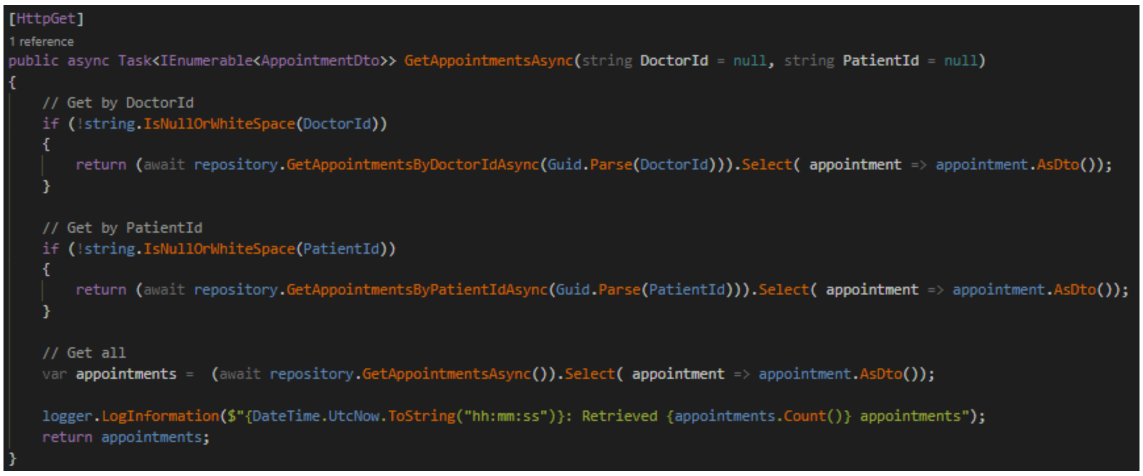
3.2.3 Business logic for creating valid appointments

Each controller features at least a basic CRUD interface, however, CreateAppointment method within Appointment controller features business logic, amongst other methods.



```
[HttpPost]
0 references
public async Task<ActionResult<AppointmentDto>> CreateAppointmentAsync(CreateAppointmentDto appointmentDto)
{
    // Business logic where it checks whether the timeslot is available
    var existingAppointments = await GetAppointmentsAsync(appointmentDto.DoctorId.ToString());
    var endTime = appointmentDto.AppointmentTime.AddMinutes(appointmentDto.AppointmentDurationMins);
    Console.WriteLine(endTime);
                Console.WriteLine(appointmentDto.AppointmentDurationMins);

    var dateClash = false;

    foreach (AppointmentDto item in existingAppointments)
    {
        if ((appointmentDto.AppointmentTime >= item.AppointmentTime) && (appointmentDto.AppointmentTime <= item.AppointmentEndTime))
        {
            // Start time is within existing appointment's duration
            dateClash = true;
        }

        if ((endTime >= item.AppointmentTime) && (endTime <= item.AppointmentEndTime))
        {
            // End time is within existing appointment's duration
            dateClash = true;
        }

        if ((appointmentDto.AppointmentTime <= item.AppointmentTime) && (endTime >= item.AppointmentEndTime))
        {
            // Start and end time encapsulates existing appointments duration
            dateClash = true;
        }
    }

    if (dateClash == true)
    {
        return BadRequest();
    }

    Appointment appointment = new() {
        Id = Guid.NewGuid(),
        DoctorId = appointmentDto.DoctorId,
        PatientId = appointmentDto.PatientId,
        AppointmentTime = appointmentDto.AppointmentTime,
        AppointmentEndTime = endTime,
        AppointmentDurationMins = appointmentDto.AppointmentDurationMins,
        CreatedDate = DateTimeOffset.UtcNow
    };

    await repository.CreateAppointmentAsync(appointment);

    return CreatedAtAction(nameof(GetAppointmentAsync), new { id = appointment.Id }, appointment.AsDto());
}
```

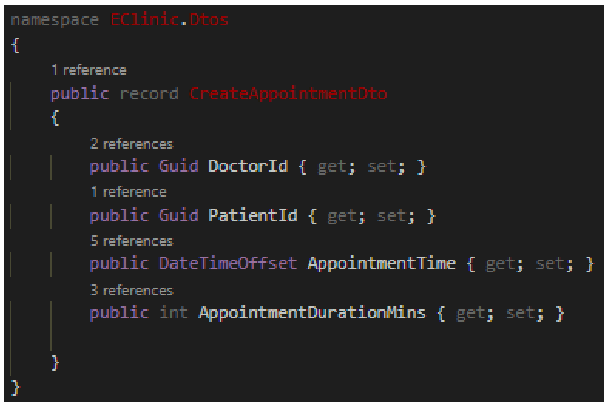
Figure 23: CreateAppointment method with business logic for creating valid appointments with no overlap in appointment time



```
[HttpGet]
1 reference
public async Task<IEnumerable<AppointmentDto>> GetAppointmentsAsync(string DoctorId = null, string PatientId = null)
{
    // Get by DoctorId
    if (!string.IsNullOrWhiteSpace(DoctorId))
    {
        return (await repository.GetAppointmentsByDoctorIdAsync(Guid.Parse(DoctorId))).Select( appointment => appointment.AsDto());
    }

    // Get by PatientId
    if (!string.IsNullOrWhiteSpace(PatientId))
    {
        return (await repository.GetAppointmentsByPatientIdAsync(Guid.Parse(PatientId))).Select( appointment => appointment.AsDto());
    }

    // Get all
    var appointments =  (await repository.GetAppointmentsAsync()).Select( appointment => appointment.AsDto());

    logger.LogInformation($"{DateTime.UtcNow.ToString("hh:mm:ss")}: Retrieved {appointments.Count()} appointments");
    return appointments;
}
```

Figure 24: GetAppointment method used by CreateAppointment to retrieve list of all existing appointments

This method accepts a CreateAppointmentDto (Figure 25). Appointment-wise, this method only includes appointment start time and its duration, and the end times are calculated during runtime for convenience.



```
namespace EClinic.Dtos
{
    1 reference
    public record CreateAppointmentDto
    {
        2 references
        public Guid DoctorId { get; set; }
        1 reference
        public Guid PatientId { get; set; }
        5 references
        public DateTimeOffset AppointmentTime { get; set; }
        3 references
        public int AppointmentDurationMins { get; set; }

    }
}
```

Figure 25: CreateAppointmentDto

I used an existing GetAppointmentsAsync method (Figure 24) to retrieve list of all chosen doctor's appointments. As ASP.NET supports only one of each type of REST API method, instead of using operator overloading which would have looked cleaner, I'm using a single method with if statements to query by doctor or/and patient ID.

The foreach statement (Figure 23) checks the requested appointment against list of all existing appointments. I used three conditions to check whether there is an overlap in appointment start or end time:

1. Start time is within existing appointment's duration
2. End time is within existing appointment's duration
3. Start and end time encapsulates existing appointments duration

Start and end time both being inside the appointment time is already filtered by 1 and 2, hence a fourth condition is not needed. With the above three conditions, all appointments can be made sure that its times do not overlap another appointment. In the case of a date clash, a 400 Bad Request response status is sent. Otherwise, the created appointment is sent as per convention.

3.3 Evaluation

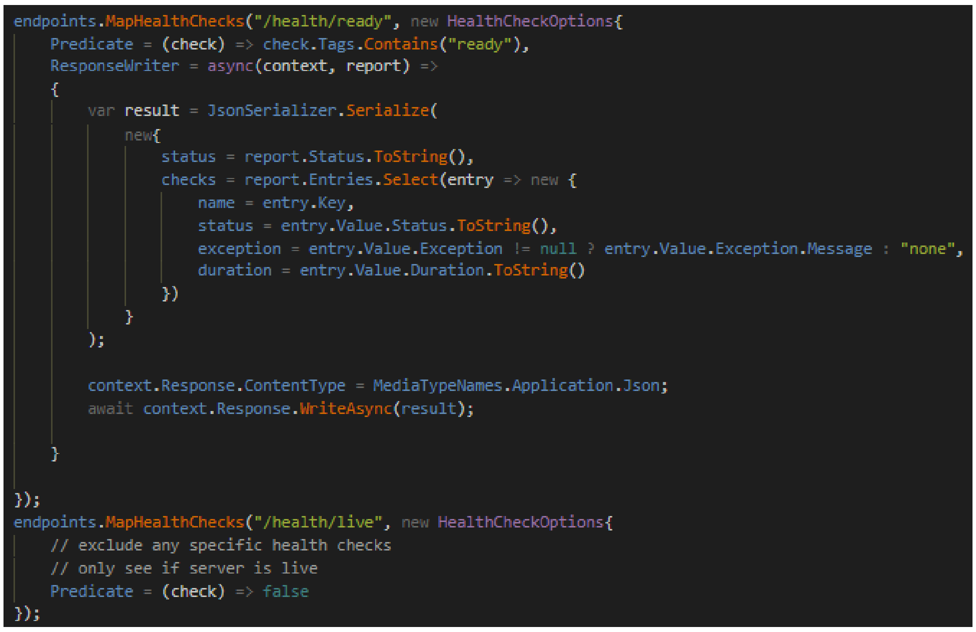
Alternatively, the DTOs could be configured automatically using a NuGet package called AutoMapper. This is more maintainable than manually creating considerable number of DTOs.

In the future, I want to further improve the business logic for creating appointments by generating available timeslots automatically, instead of only checking if there are overlaps for the requested appointment.

4 Observability and Maintainability

Unexpected bugs and situations can arise to web services. Observability and maintainability concerns impact how quickly, and effectively unexpected scenarios can be fixed. EClinic currently uses health monitoring with integration to Kubernetes, and interactable documentation with Postman to address such concerns.

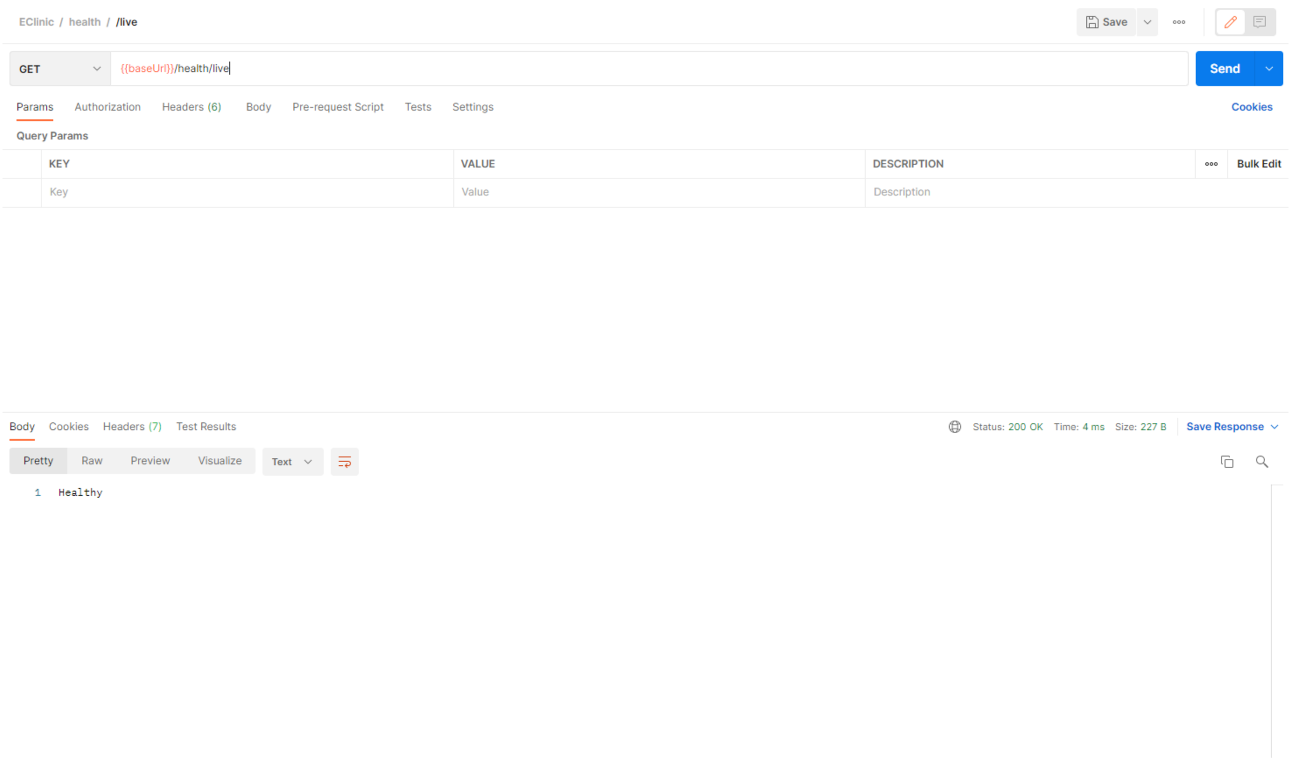
4.1 Health Checks NuGet Package and integration with Kubernetes



```
endpoints.MapHealthChecks("/health/ready", new HealthCheckOptions{
    Predicate = (check) => check.Tags.Contains("ready"),
    ResponseWriter = async(context, report) =>
    {
        var result = JsonSerializer.Serialize(
            new{
                status = report.Status.ToString(),
                checks = report.Entries.Select(entry => new {
                    name = entry.Key,
                    status = entry.Value.Status.ToString(),
                    exception = entry.Value.Exception != null ? entry.Value.Exception.Message : "none",
                    duration = entry.Value.Duration.ToString()
                })
            }
        );

        context.Response.ContentType = MediaTypeNames.Application.Json;
        await context.Response.WriteAsync(result);

    }

});
endpoints.MapHealthChecks("/health/live", new HealthCheckOptions{
    // exclude any specific health checks
    // only see if server is live
    Predicate = (check) => false
});
```

Figure 26: implementation for 'ready' and 'live' health check endpoints using HealthCheck package

Health checks allow a status of the server to be outputted through its API. I implemented a /health/ready and /health/live endpoints for checking liveness and readiness of the server. Liveness check only checks whether the server is running. Readiness check provides information about whether the server is ready to use, including database connections. HealthCheck NuGet package provides interfaces for customizing output message of the health check endpoints. For example, I customized the readiness check to provide information about MongoDB connection.



EClinic / health / /live

GET {{baseUrl}}/health/live

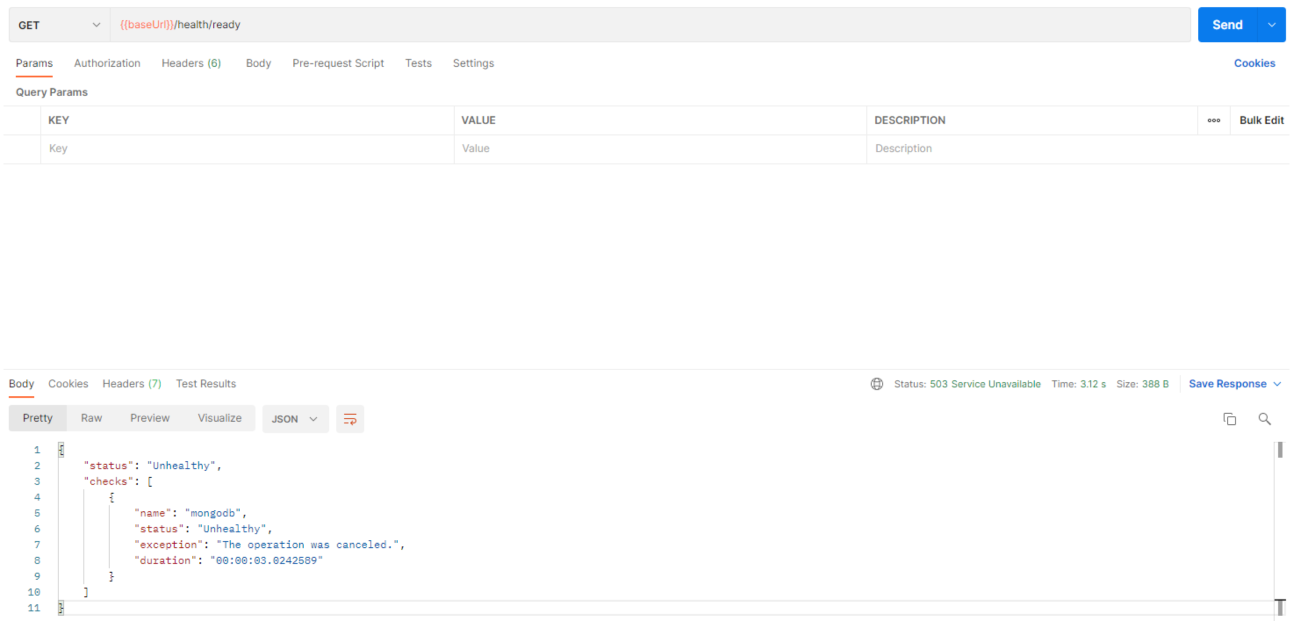
Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Status: 200 OK Time: 4 ms Size: 227 B

```
1   Healthy
```

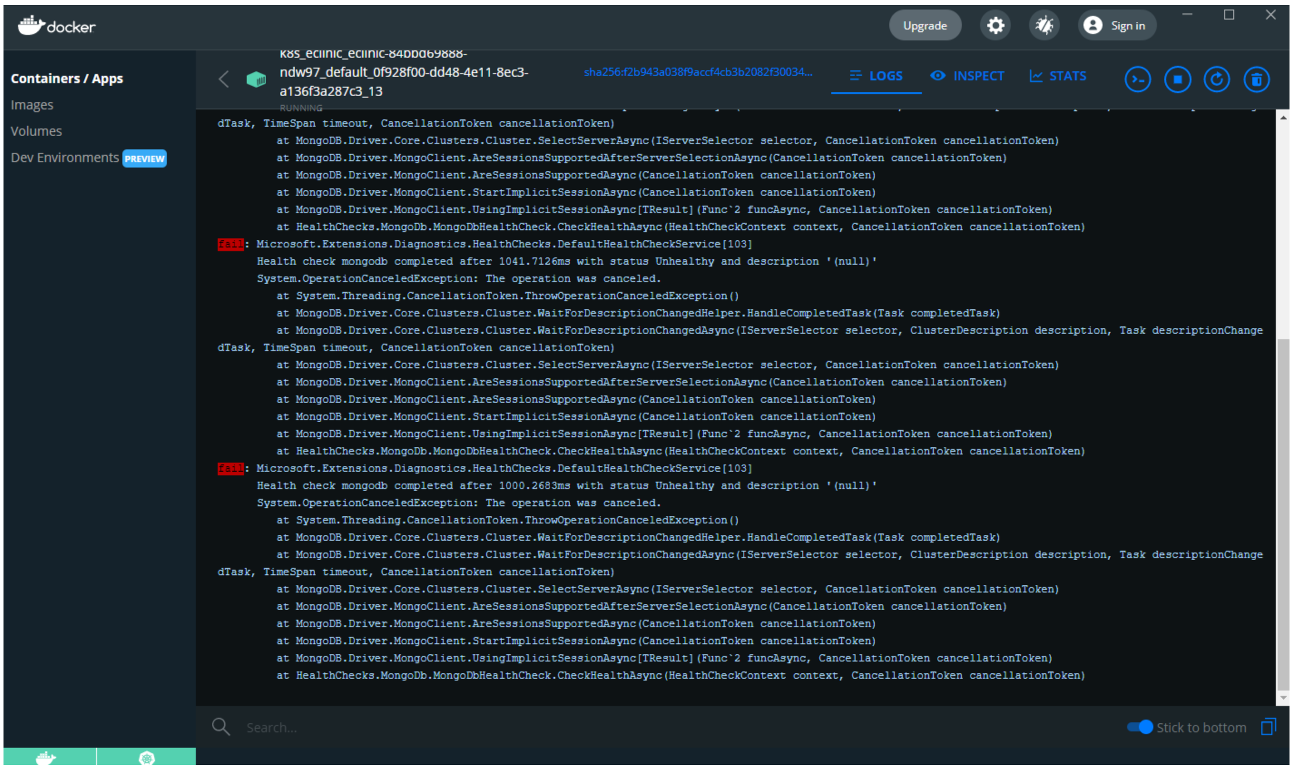
Figure 27: Liveness check showing that the server is running


GET {{baseUrl}}/health/ready

Status: 503 Service Unavailable Time: 3.12 s Size: 388 B

```json
{
    "status": "Unhealthy",
    "checks": [
        {
            "name": "mongodb",
            "status": "Unhealthy",
            "exception": "The operation was canceled.",
            "duration": "00:00:03.0242589"
        }
    ]
}
```

Figure 28: Readiness check on /health/ready endpoint fails due to database connection error


k8s_eclinic_eclinic-84bbd69888-ndw97_default_0f928f00-dd48-4e11-8ec3-a136f3a287c3_13

RUNNING

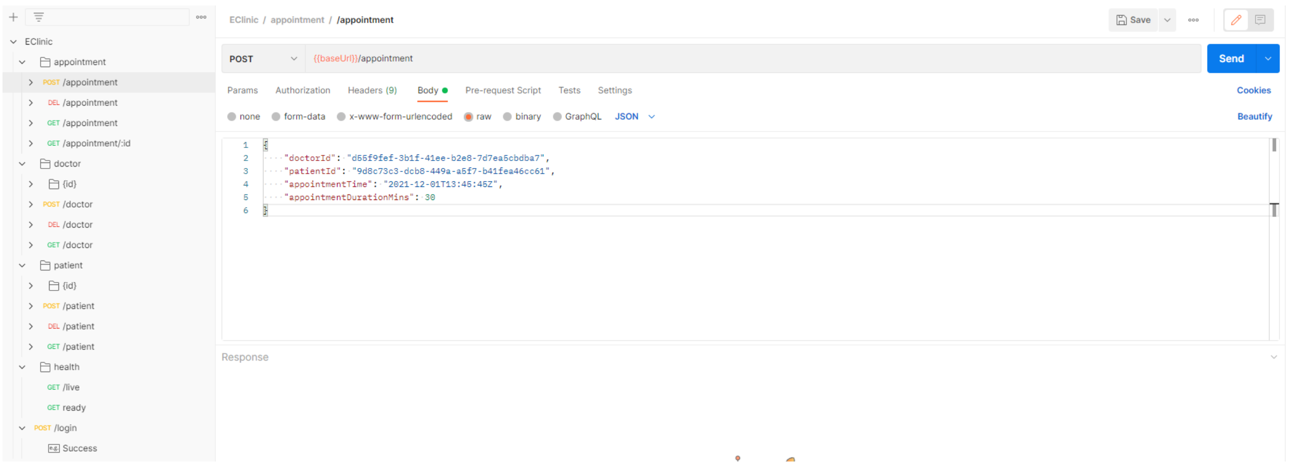
sha256:f2b943a038f9accf4cb3b2082f30034...

```
dTask, TimeSpan timeout, CancellationToken cancellationToken)
         at MongoDB.Driver.Core.Clusters.Cluster.SelectServerAsync(IServerSelector selector, CancellationToken cancellationToken)
         at MongoDB.Driver.MongoClient.AreSessionsSupportedAfterServerSelectionAsync(CancellationToken cancellationToken)
         at MongoDB.Driver.MongoClient.AreSessionsSupportedAsync(CancellationToken cancellationToken)
         at MongoDB.Driver.MongoClient.StartImplicitSessionAsync(CancellationToken cancellationToken)
         at MongoDB.Driver.MongoClient.UsingImplicitSessionAsync[TResult](Func`2 funcAsync, CancellationToken cancellationToken)
         at HealthChecks.MongoDb.MongoDbHealthCheck.CheckHealthAsync(HealthCheckContext context, CancellationToken cancellationToken)
fail: Microsoft.Extensions.Diagnostics.HealthChecks.DefaultHealthCheckService[103]
      Health check mongodb completed after 1041.7126ms with status Unhealthy and description '(null)'
      System.OperationCanceledException: The operation was canceled.
         at System.Threading.CancellationToken.ThrowOperationCanceledException()
         at MongoDB.Driver.Core.Clusters.Cluster.WaitForDescriptionChangedHelper.HandleCompletedTask(Task completedTask)
         at MongoDB.Driver.Core.Clusters.Cluster.WaitForDescriptionChangedAsync(IServerSelector selector, ClusterDescription description, Task descriptionChange
dTask, TimeSpan timeout, CancellationToken cancellationToken)
         at MongoDB.Driver.Core.Clusters.Cluster.SelectServerAsync(IServerSelector selector, CancellationToken cancellationToken)
         at MongoDB.Driver.MongoClient.AreSessionsSupportedAfterServerSelectionAsync(CancellationToken cancellationToken)
         at MongoDB.Driver.MongoClient.AreSessionsSupportedAsync(CancellationToken cancellationToken)
         at MongoDB.Driver.MongoClient.StartImplicitSessionAsync(CancellationToken cancellationToken)
         at MongoDB.Driver.MongoClient.UsingImplicitSessionAsync[TResult](Func`2 funcAsync, CancellationToken cancellationToken)
         at HealthChecks.MongoDb.MongoDbHealthCheck.CheckHealthAsync(HealthCheckContext context, CancellationToken cancellationToken)
fail: Microsoft.Extensions.Diagnostics.HealthChecks.DefaultHealthCheckService[103]
      Health check mongodb completed after 1000.2683ms with status Unhealthy and description '(null)'
      System.OperationCanceledException: The operation was canceled.
         at System.Threading.CancellationToken.ThrowOperationCanceledException()
         at MongoDB.Driver.Core.Clusters.Cluster.WaitForDescriptionChangedHelper.HandleCompletedTask(Task completedTask)
         at MongoDB.Driver.Core.Clusters.Cluster.WaitForDescriptionChangedAsync(IServerSelector selector, ClusterDescription description, Task descriptionChange
dTask, TimeSpan timeout, CancellationToken cancellationToken)
         at MongoDB.Driver.Core.Clusters.Cluster.SelectServerAsync(IServerSelector selector, CancellationToken cancellationToken)
         at MongoDB.Driver.MongoClient.AreSessionsSupportedAfterServerSelectionAsync(CancellationToken cancellationToken)
         at MongoDB.Driver.MongoClient.AreSessionsSupportedAsync(CancellationToken cancellationToken)
         at MongoDB.Driver.MongoClient.StartImplicitSessionAsync(CancellationToken cancellationToken)
         at MongoDB.Driver.MongoClient.UsingImplicitSessionAsync[TResult](Func`2 funcAsync, CancellationToken cancellationToken)
         at HealthChecks.MongoDb.MongoDbHealthCheck.CheckHealthAsync(HealthCheckContext context, CancellationToken cancellationToken)
```

Figure 29: Kubernetes restarting the EClinic pod until the readiness check is successful

I integrated the health checks with Kubernetes through an option Kubernetes provides in its configuration files in the form of a livenessProbe and readinessProbe parameter where I provided the API endpoint (Figure 49). Kubernetes uses these health checks to know when to restart its pods, or to know when the pod has successfully started. For example, Figure 29 shows Kubernetes behaviour where Kubernetes constantly restarts the EClinic ASP.NET pod until the connection to MongoDB database is re-established.

4.2 Postman


EClinic
appointment
POST /appointment
DEL /appointment
GET /appointment
GET /appointment/:id
doctor
{id}
POST /doctor
DEL /doctor
GET /doctor
patient
{id}
POST /patient
DEL /patient
GET /patient
health
GET /live
GET ready
POST /login
Success

EClinic / appointment / /appointment

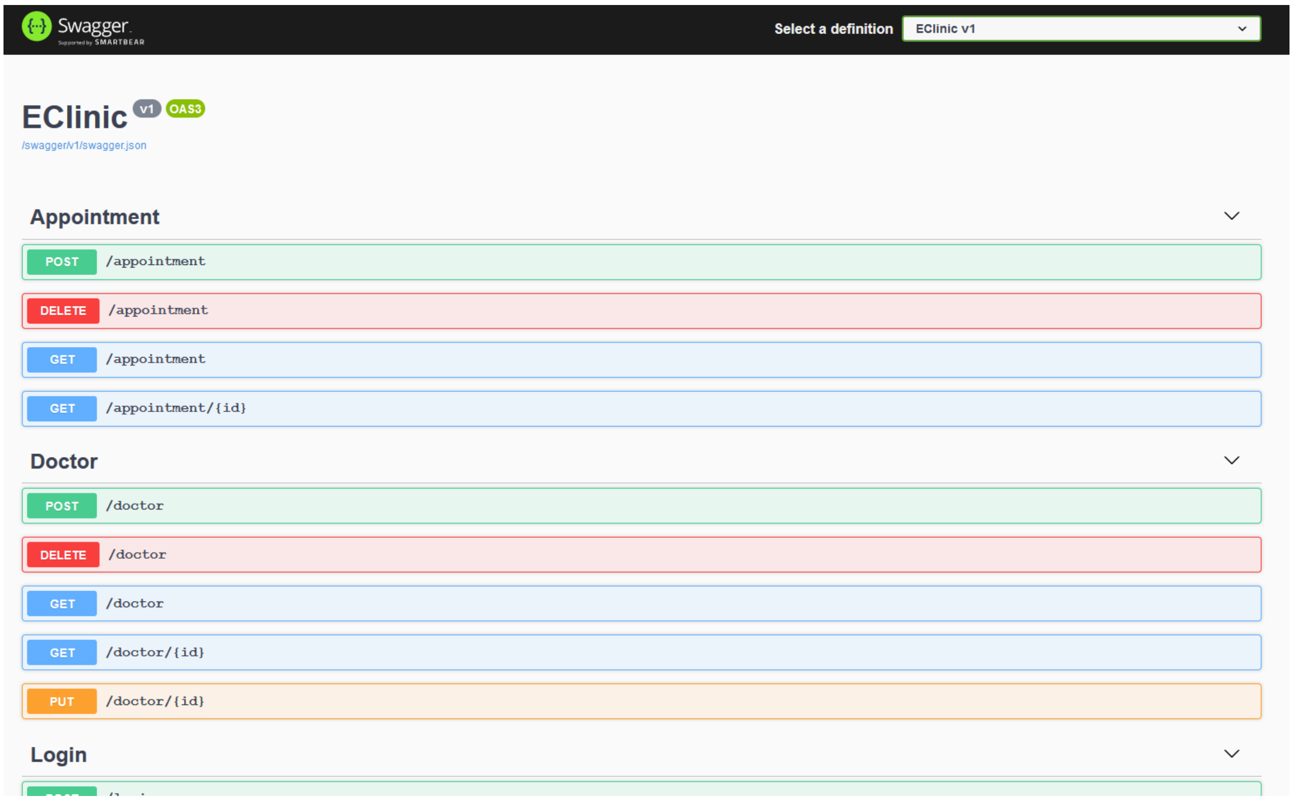
POST {{baseUrl}}/appointment

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

```json
{
    "doctorId": "d55f9fef-3b1f-41ee-b2e8-7d7ea5cbdba7",
    "patientId": "9d8c73c3-dcb8-449a-a5f7-b41fea46cc61",
    "appointmentTime": "2021-12-01T13:45:45Z",
    "appointmentDurationMins": 30
}
```

Figure 30: Postman collection generated by Swagger


Select a definition: EClinic v1

EClinic v1 OAS3
/swagger/v1/swagger.json

Appointment
POST /appointment
DELETE /appointment
GET /appointment
GET /appointment/{id}

Doctor
POST /doctor
DELETE /doctor
GET /doctor
GET /doctor/{id}
PUT /doctor/{id}

Login

Figure 31: Automatically generated Swagger API documentation

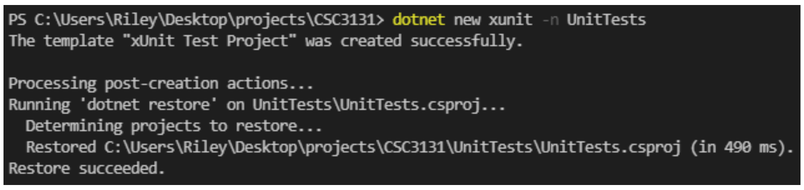
ASP.NET 5 is bundled with Swagger. Swagger automatically generates an interactable web API documentation from the controller methods. I used Swagger's option to generate a JSON collection for Postman, a platform for building interactable API documentation and collection. Swagger is only available on a developmental instance, whereas Postman can query deployed websites. Coupled with the above health checks, the developer team can use Postman to test problematic API methods when unexpected errors arise.

4.3 Evaluation

I want to add Prometheus to Kubernetes to send automatic alerts by email when any of the health checks fails. When an unexpected error arises, and it is not able to be addressed by Kubernetes restarting the pod, email alerts to let the developer team know that a fix is required will be useful.

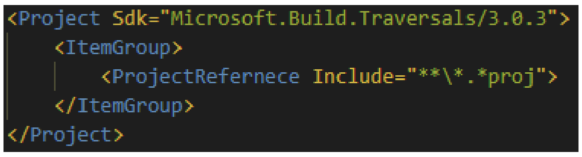
5 Continuous Integration (xUnit)

For quickly validating and merging changes, xUnit provides a convenient way to automatically unit test API methods in an ASP.NET project. Compared to alternative such as NUnit (derived from JUnit), and MSTest, xUnit is an active open-source framework that is part of the .NET Foundation [7]. I created a new xUnit project using dotnet new to generate a template and made appropriate configurations.



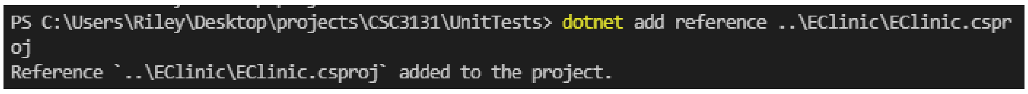
```
PS C:\Users\Riley\Desktop\projects\CSC3131> dotnet new xunit -n UnitTests
The template "xUnit Test Project" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on UnitTests\UnitTests.csproj...
  Determining projects to restore...
  Restored C:\Users\Riley\Desktop\projects\CSC3131\UnitTests\UnitTests.csproj (in 490 ms).
Restore succeeded.
```

Figure 32: Creating a new unit tests project with xUnit template



```
<Project Sdk="Microsoft.Build.Traversals/3.0.3">
    <ItemGroup>
        <ProjectRefernece Include="**\*.*proj">
    </ItemGroup>
</Project>
```

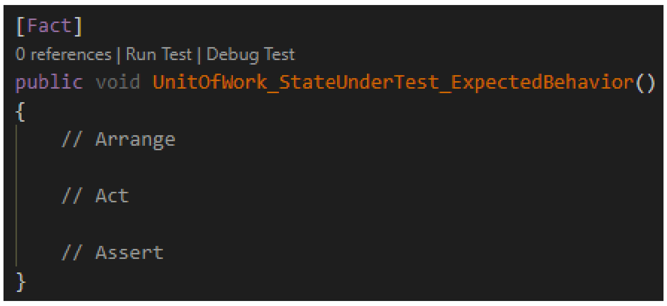
Figure 33: Including the newly generated project in the build list for .csproj



```
PS C:\Users\Riley\Desktop\projects\CSC3131\UnitTests> dotnet add reference ..\EClinic\EClinic.cspr
oj
Reference `..\EClinic\EClinic.csproj` added to the project.
```

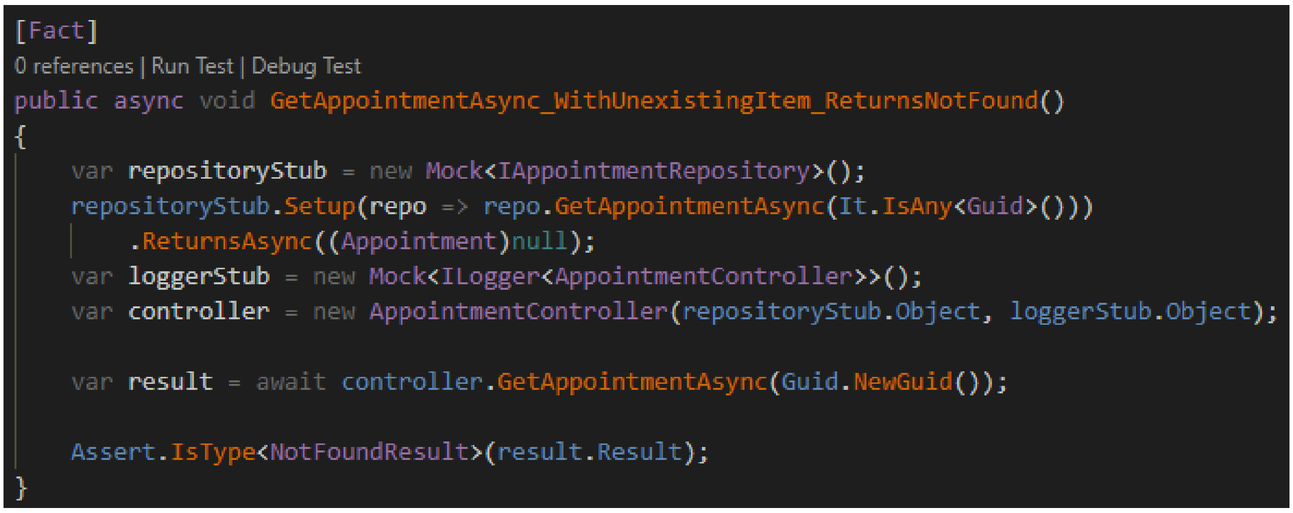
Figure 34: Adding references to EClinic for Visual Studio intelliSense to work correctly

A good pattern for writing tests is the Arrange-Act-Assert pattern [7]. In Arrange, the appropriate dependencies are configured. I used Mock, a NuGet package to mock the dependent repositories. In Act, the appropriate API method calls are made to test the mock repository. Finally, in Assert, the appropriate assertion is made.



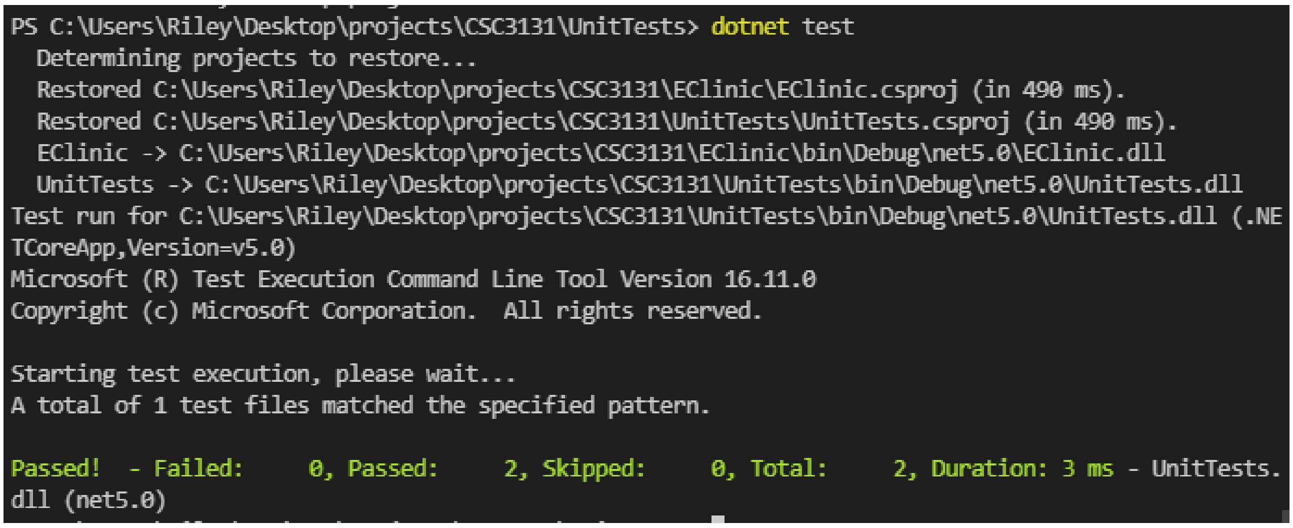
```
[Fact]
0 references | Run Test | Debug Test
public void UnitOfWork_StateUnderTest_ExpectedBehavior()
{
    // Arrange

    // Act

    // Assert
}
```

Figure 35: AAA (Arrange-Act-Assert) pattern for writing tests



```csharp
[Fact]
0 references | Run Test | Debug Test
public async void GetAppointmentAsync_WithUnexistingItem_ReturnsNotFound()
{
    var repositoryStub = new Mock<IAppointmentRepository>();
    repositoryStub.Setup(repo => repo.GetAppointmentAsync(It.IsAny<Guid>()))
        .ReturnsAsync((Appointment)null);
    var loggerStub = new Mock<ILogger<AppointmentController>>();
    var controller = new AppointmentController(repositoryStub.Object, loggerStub.Object);

    var result = await controller.GetAppointmentAsync(Guid.NewGuid());

    Assert.IsType<NotFoundResult>(result.Result);
}
```

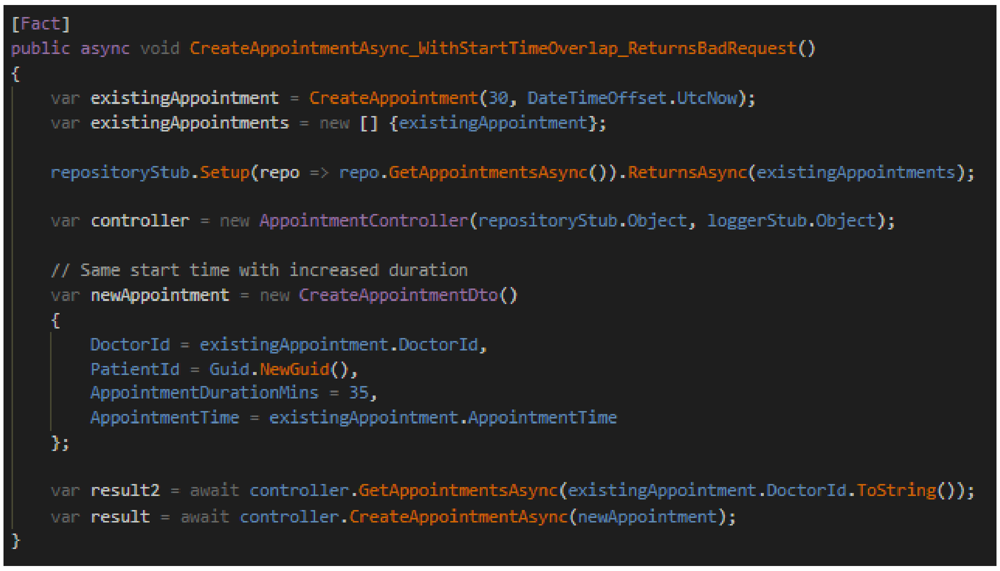
Figure 36: Testing GetAppointmentAsync method for requesting an non-existing item that should return ReturnsNotFound status



```
PS C:\Users\Riley\Desktop\projects\CSC3131\UnitTests> dotnet test
  Determining projects to restore...
  Restored C:\Users\Riley\Desktop\projects\CSC3131\EClinic\EClinic.csproj (in 490 ms).
  Restored C:\Users\Riley\Desktop\projects\CSC3131\UnitTests\UnitTests.csproj (in 490 ms).
  EClinic -> C:\Users\Riley\Desktop\projects\CSC3131\EClinic\bin\Debug\net5.0\EClinic.dll
  UnitTests -> C:\Users\Riley\Desktop\projects\CSC3131\UnitTests\bin\Debug\net5.0\UnitTests.dll
Test run for C:\Users\Riley\Desktop\projects\CSC3131\UnitTests\bin\Debug\net5.0\UnitTests.dll (.NE
TCoreApp,Version=v5.0)
Microsoft (R) Test Execution Command Line Tool Version 16.11.0
Copyright (c) Microsoft Corporation.  All rights reserved.

Starting test execution, please wait...
A total of 1 test files matched the specified pattern.

Passed!  - Failed:     0, Passed:     2, Skipped:     0, Total:     2, Duration: 3 ms - UnitTests.
dll (net5.0)
```

Figure 37: Automatically running every test using dotnet test command

5.1 Evaluation



```csharp
[Fact]
public async void CreateAppointmentAsync_WithStartTimeOverlap_ReturnsBadRequest()
{
    var existingAppointment = CreateAppointment(30, DateTimeOffset.UtcNow);
    var existingAppointments = new [] {existingAppointment};

    repositoryStub.Setup(repo => repo.GetAppointmentsAsync()).ReturnsAsync(existingAppointments);

    var controller = new AppointmentController(repositoryStub.Object, loggerStub.Object);

    // Same start time with increased duration
    var newAppointment = new CreateAppointmentDto()
    {
        DoctorId = existingAppointment.DoctorId,
        PatientId = Guid.NewGuid(),
        AppointmentDurationMins = 35,
        AppointmentTime = existingAppointment.AppointmentTime
    };

    var result2 = await controller.GetAppointmentsAsync(existingAppointment.DoctorId.ToString());
    var result = await controller.CreateAppointmentAsync(newAppointment);
}
```

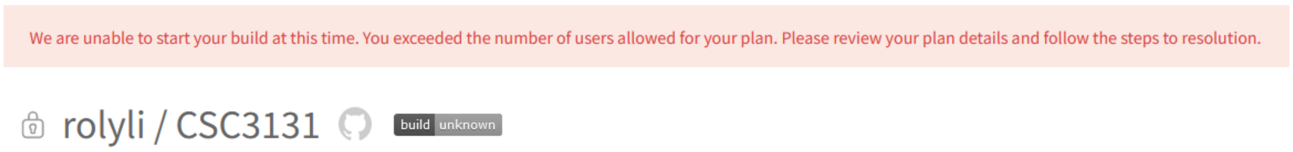
Figure 38: Testing CreateAppointmentAsync for validating business logic for precluding overlapping appointments

In the future, I would like to complete the CreateAppointmentAsync test for business logic of checking overlapping appointments, as the Mock repository does not work in the implementation in Figure 38. As the business logic contains many conditionals and distinct cases, it is more prone to human error, hence testing this method will be important.

6 Continuous Deployment

6.1 TravisCI

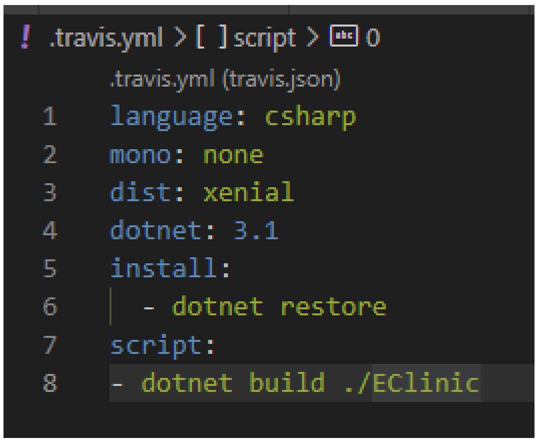
Unlike alternatives such as Jenkins, Travis CI is convenient because a hosted service hence a local instance of a continuous integration server is not required [8]. Unfortunately, I could not get Travis CI to work despite many attempts.



We are unable to start your build at this time. You exceeded the number of users allowed for your plan. Please review your plan details and follow the steps to resolution.

rolyli / CSC3131 build unknown

Figure 39: TravisCI showing user exceeded build error despite no builds made



! .travis.yml > [ ] script > 0

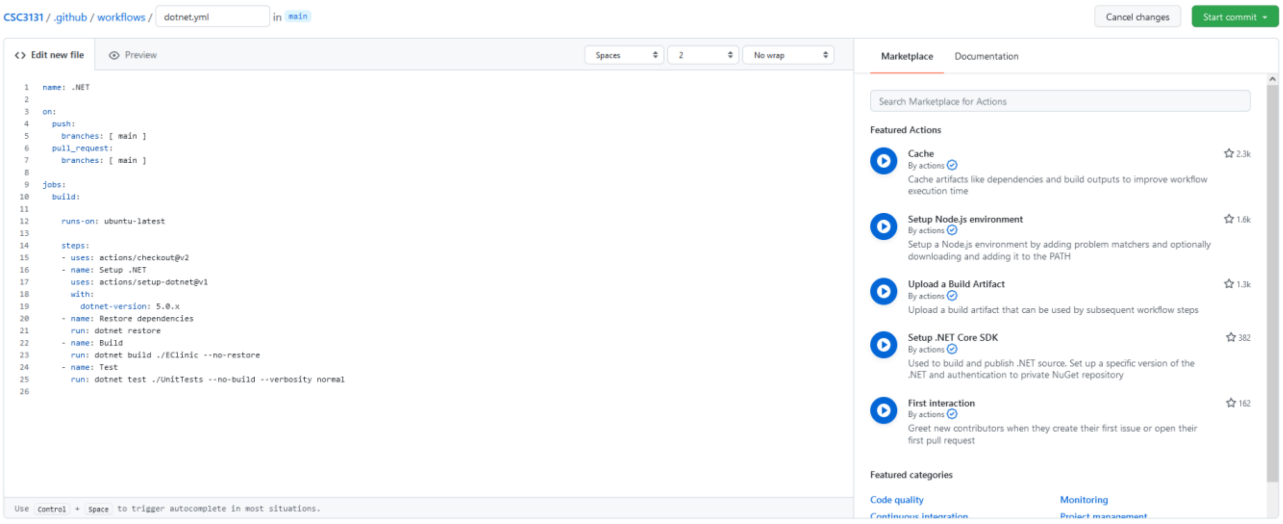
.travis.yml (travis.json)

```yaml
language: csharp
mono: none
dist: xenial
dotnet: 3.1
install:
  - dotnet restore
script:
- dotnet build ./EClinic
```

Figure 40: .travis.yml configuration for TravisCI

6.2 Github Workflows

As I was already using GitHub, I decided to use GitHub Workflows as it is a hosted service like TravisCI. Although I was presented with a build error (Figure 42), I persevered with the implementation as builds were being made unlike TravisCI. The error was fixed by removing the obj folder as there were package conflicts to existing packages in the obj folder [9]. Figure 41 shows the automatic build and unit tests succeeding after making a commit to the GitHub repository.



CSC3131 / .github / workflows / dotnet.yml in main

```yaml
name: .NET

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Setup .NET
      uses: actions/setup-dotnet@v1
      with:
        dotnet-version: 5.0.x
    - name: Restore dependencies
      run: dotnet restore
    - name: Build
      run: dotnet build ./EClinic --no-restore
    - name: Test
      run: dotnet test ./UnitTests --no-build --verbosity normal

```

Use Control + Space to trigger autocomplete in most situations.

Featured Actions

Cache ☆ 2.3k
By actions
Cache artifacts like dependencies and build outputs to improve workflow execution time

Setup Node.js environment ☆ 1.6k
By actions
Setup a Node.js environment by adding problem matchers and optionally downloading and adding it to the PATH

Upload a Build Artifact ☆ 1.3k
By actions
Upload a build artifact that can be used by subsequent workflow steps

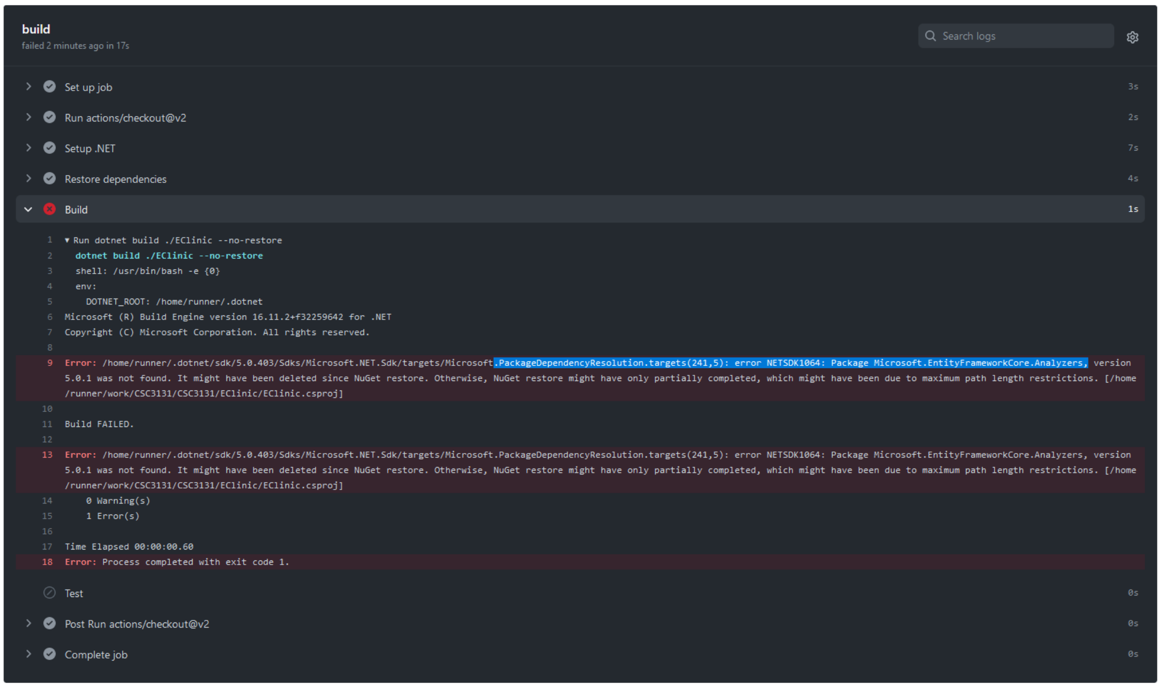
Setup .NET Core SDK ☆ 382
By actions
Used to build and publish .NET source. Set up a specific version of the .NET and authentication to private NuGet repository

First interaction ☆ 162
By actions
Greet new contributors when they create their first issue or open their first pull request

Featured categories

Code quality Monitoring

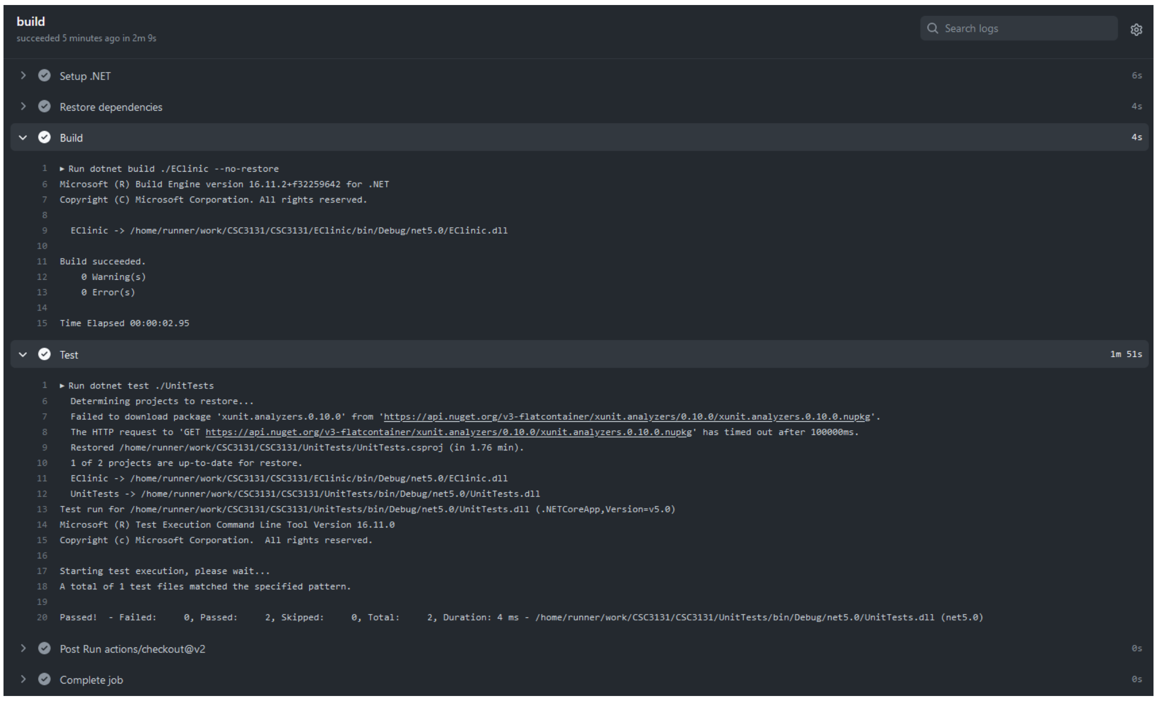
Figure 41: GitHub Workflows configuration



build
failed 2 minutes ago in 17s

```
Set up job                      3s
Run actions/checkout@v2         2s
Setup .NET                      7s
Restore dependencies            4s
Build                           1s
▾ Run dotnet build ./EClinic --no-restore
  dotnet build ./EClinic --no-restore
  shell: /usr/bin/bash -e {0}
  env:
    DOTNET_ROOT: /home/runner/.dotnet
Microsoft (R) Build Engine version 16.11.2+f32259642 for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

Error: /home/runner/.dotnet/sdk/5.0.403/Sdks/Microsoft.NET.Sdk/targets/Microsoft.PackageDependencyResolution.targets(241,5): error NETSDK1064: Package Microsoft.EntityFrameworkCore.Analyzers, version 5.0.1 was not found. It might have been deleted since NuGet restore. Otherwise, NuGet restore might have only partially completed, which might have been due to maximum path length restrictions. [/home/runner/work/CSC3131/CSC3131/EClinic/EClinic.csproj]

Build FAILED.

Error: /home/runner/.dotnet/sdk/5.0.403/Sdks/Microsoft.NET.Sdk/targets/Microsoft.PackageDependencyResolution.targets(241,5): error NETSDK1064: Package Microsoft.EntityFrameworkCore.Analyzers, version 5.0.1 was not found. It might have been deleted since NuGet restore. Otherwise, NuGet restore might have only partially completed, which might have been due to maximum path length restrictions. [/home/runner/work/CSC3131/CSC3131/EClinic/EClinic.csproj]
    0 Warning(s)
    1 Error(s)

Time Elapsed 00:00:00.60
Error: Process completed with exit code 1.
Test                            0s
Post Run actions/checkout@v2    0s
Complete job                    0s
```

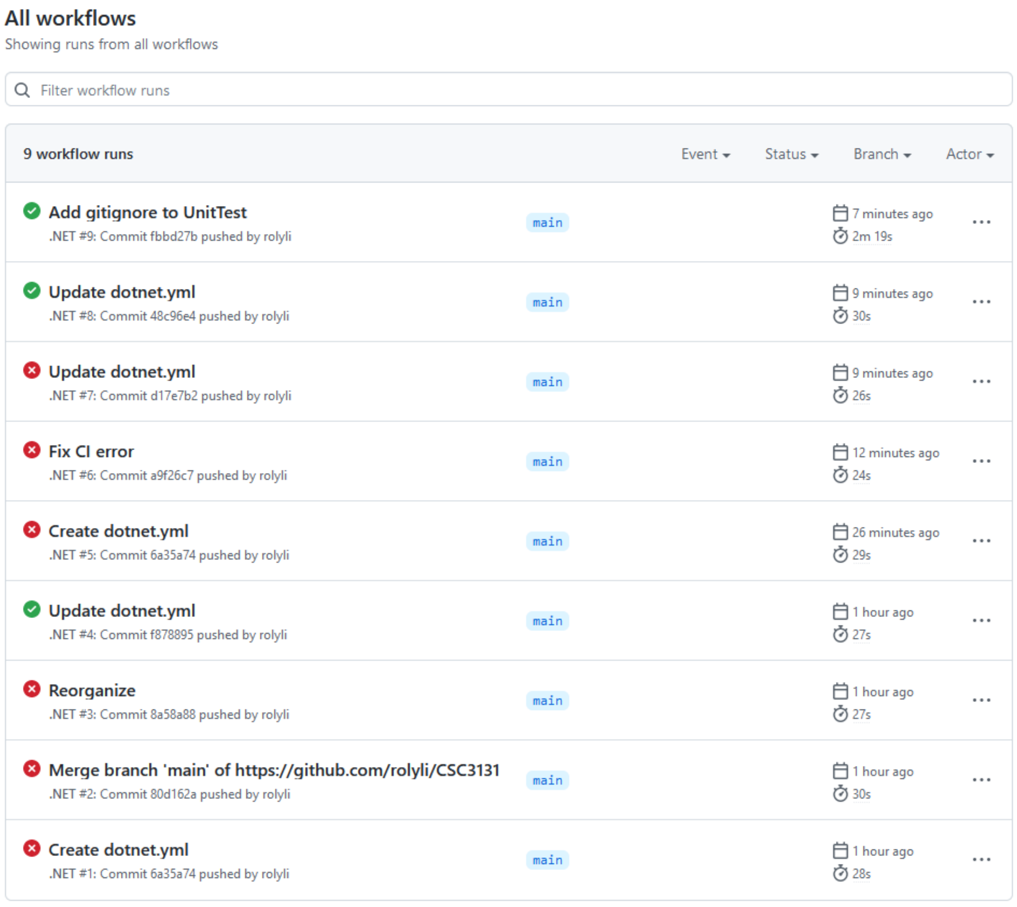
Figure 42: NETSDK1064 error from building the EClinic project



```
build
succeeded 5 minutes ago in 2m 9s

> Setup .NET                                                    6s
> Restore dependencies                                          4s
v Build                                                         4s
▸ Run dotnet build ./EClinic --no-restore
Microsoft (R) Build Engine version 16.11.2+f32259642 for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

  EClinic -> /home/runner/work/CSC3131/CSC3131/EClinic/bin/Debug/net5.0/EClinic.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:02.95
v Test                                                          1m 51s
▸ Run dotnet test ./UnitTests
  Determining projects to restore...
  Failed to download package 'xunit.analyzers.0.10.0' from 'https://api.nuget.org/v3-flatcontainer/xunit.analyzers/0.10.0/xunit.analyzers.0.10.0.nupkg'.
  The HTTP request to 'GET https://api.nuget.org/v3-flatcontainer/xunit.analyzers/0.10.0/xunit.analyzers.0.10.0.nupkg' has timed out after 100000ms.
  Restored /home/runner/work/CSC3131/CSC3131/UnitTests/UnitTests.csproj (in 1.76 min).
  1 of 2 projects are up-to-date for restore.
  EClinic -> /home/runner/work/CSC3131/CSC3131/EClinic/bin/Debug/net5.0/EClinic.dll
  UnitTests -> /home/runner/work/CSC3131/CSC3131/UnitTests/bin/Debug/net5.0/UnitTests.dll
Test run for /home/runner/work/CSC3131/CSC3131/UnitTests/bin/Debug/net5.0/UnitTests.dll (.NETCoreApp,Version=v5.0)
Microsoft (R) Test Execution Command Line Tool Version 16.11.0
Copyright (c) Microsoft Corporation.  All rights reserved.

Starting test execution, please wait...
A total of 1 test files matched the specified pattern.

Passed!  - Failed:     0, Passed:     2, Skipped:     0, Total:     2, Duration: 4 ms - /home/runner/work/CSC3131/CSC3131/UnitTests/bin/Debug/net5.0/UnitTests.dll (net5.0)
> Post Run actions/checkout@v2                                  0s
> Complete job                                                  0s
```

Figure 43: Automatically building and running xUnit unit tests for EClinic after a commit is pushed to GitHub

6.3 Evaluation



All workflows

Showing runs from all workflows

Filter workflow runs

9 workflow runs	Event	Status	Branch	Actor
✓ **Add gitignore to UnitTest** .NET #9: Commit fbbd27b pushed by rolyli			main	7 minutes ago 2m 19s
✓ **Update dotnet.yml** .NET #8: Commit 48c96e4 pushed by rolyli			main	9 minutes ago 30s
✗ **Update dotnet.yml** .NET #7: Commit d17e7b2 pushed by rolyli			main	9 minutes ago 26s
✗ **Fix CI error** .NET #6: Commit a9f26c7 pushed by rolyli			main	12 minutes ago 24s
✗ **Create dotnet.yml** .NET #5: Commit 6a35a74 pushed by rolyli			main	26 minutes ago 29s
✓ **Update dotnet.yml** .NET #4: Commit f878895 pushed by rolyli			main	1 hour ago 27s
✗ **Reorganize** .NET #3: Commit 8a58a88 pushed by rolyli			main	1 hour ago 27s
✗ **Merge branch 'main' of https://github.com/rolyli/CSC3131** .NET #2: Commit 80d162a pushed by rolyli			main	1 hour ago 30s
✗ **Create dotnet.yml** .NET #1: Commit 6a35a74 pushed by rolyli			main	1 hour ago 28s

Figure 44: Continuous deployment can be tricky

Although configuration was tricky, automatically running builds and unit testing ensures that changes can be integrated and deployed seamlessly. In the future, actual deployment commands to cloud providers such as Heroku can be included in the GitHub Workflows configuration to deploy after building and unit testing.

7 Scalability and Load balancing (Docker and Kubernetes)

7.1 Reason for choice

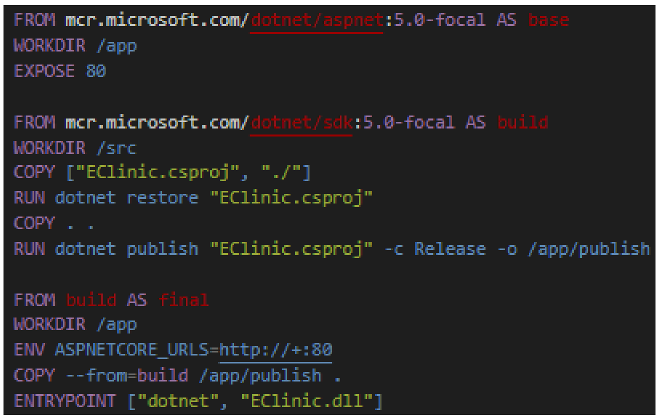
I chose Docker and Kubernetes for scalability concerns. Containerization with Docker increases performance and start-up speed, as well as modularity provided by containerization is better for portability (i.e., microservices) [10]. In addition, services can be scaled easily using Kubernetes with a single command by allocating more resources (pods).

Additionally, the following are the reasons I chose Docker and Kubernetes amongst its competitors:

- Docker is free compared to alternatives such as Amazon ECS
- Kubernetes comes bundled with Docker Dashboard, and it is an industry leading container-orchestration system

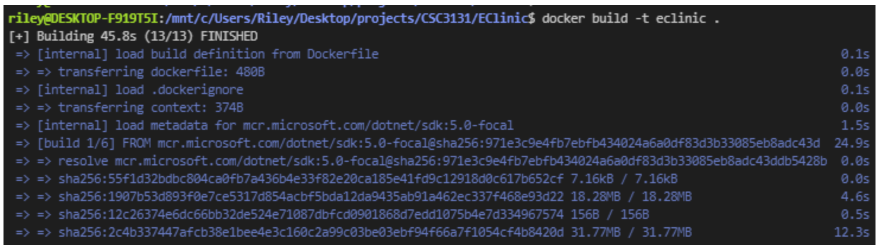
7.2 Implementation Details

7.2.1 Docker



```
FROM mcr.microsoft.com/dotnet/aspnet:5.0-focal AS base
WORKDIR /app
EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:5.0-focal AS build
WORKDIR /src
COPY ["EClinic.csproj", "./"]
RUN dotnet restore "EClinic.csproj"
COPY . .
RUN dotnet publish "EClinic.csproj" -c Release -o /app/publish

FROM build AS final
WORKDIR /app
ENV ASPNETCORE_URLS=http://+:80
COPY --from=build /app/publish .
ENTRYPOINT ["dotnet", "EClinic.dll"]
```

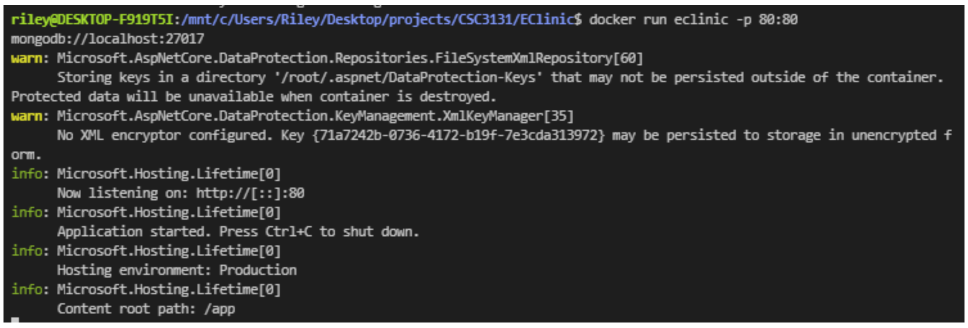
Figure 45: Docker configuration for EClinic ASP.NET backend



```
riley@DESKTOP-F919T5I:/mnt/c/Users/Riley/Desktop/projects/CSC3131/EClinic$ docker build -t eclinic .
[+] Building 45.8s (13/13) FINISHED
 => [internal] load build definition from Dockerfile                                                                    0.1s
 => => transferring dockerfile: 480B                                                                                    0.0s
 => [internal] load .dockerignore                                                                                       0.1s
 => => transferring context: 374B                                                                                       0.0s
 => [internal] load metadata for mcr.microsoft.com/dotnet/sdk:5.0-focal                                                 1.5s
 => [build 1/6] FROM mcr.microsoft.com/dotnet/sdk:5.0-focal@sha256:971e3c9e4fb7ebfb434024a6a0df83d3b33085eb8adc43d  24.9s
 => => resolve mcr.microsoft.com/dotnet/sdk:5.0-focal@sha256:971e3c9e4fb7ebfb434024a6a0df83d3b33085eb8adc43ddb5428b  0.0s
 => => sha256:55f1d32bdbc804ca0fb7a436b4e33f82e20ca185e41fd9c12918d0c617b652cf 7.16kB / 7.16kB                          0.0s
 => => sha256:1907b53d893f0e7ce5317d854acbf5bda12da9435ab91a462ec337f468e93d22 18.28MB / 18.28MB                        4.6s
 => => sha256:12c26374e6dc66bb32de524e71087dbfcd0901868d7edd1075b4e7d334967574 156B / 156B                              0.5s
 => => sha256:2c4b337447afcb38e1bee4e3c160c2a99c03be03ebf94f66a7f1054cf4b8420d 31.77MB / 31.77MB                       12.3s
```

Figure 46: Building the EClinic ASP.NET Docker image from Dockerfile

The images for the container were built using a Dockerfile configuration and command in Figure 45 and Figure 46. The csproj file specifies compile instructions for the whole project and is used by the Dockerfile. Environment variables such as the port number for the server was passed using the ENV keyword.



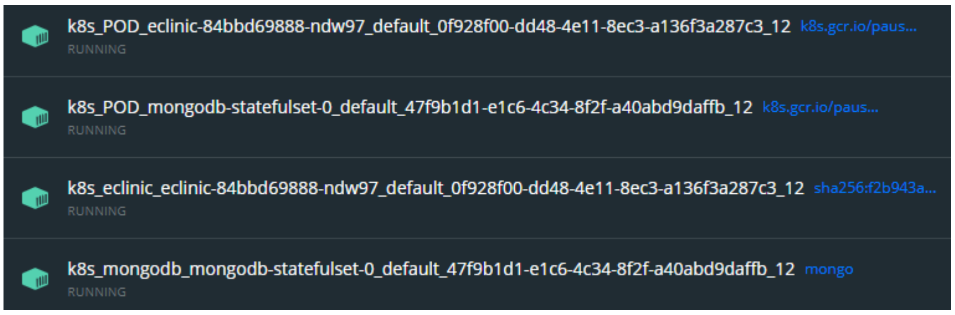
```
riley@DESKTOP-F919T5I:/mnt/c/Users/Riley/Desktop/projects/CSC3131/EClinic$ docker run eclinic -p 80:80
mongodb://localhost:27017
warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSystemXmlRepository[60]
      Storing keys in a directory '/root/.aspnet/DataProtection-Keys' that may not be persisted outside of the container.
Protected data will be unavailable when container is destroyed.
warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
      No XML encryptor configured. Key {71a7242b-0736-4172-b19f-7e3cda313972} may be persisted to storage in unencrypted f
orm.
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
```

Figure 47: Running the EClinic docker image and exposing port 80

Running the image and mapping docker host port to the internal port (port 80 -> port 80), the backend is confirmed to be working with a successful health check.

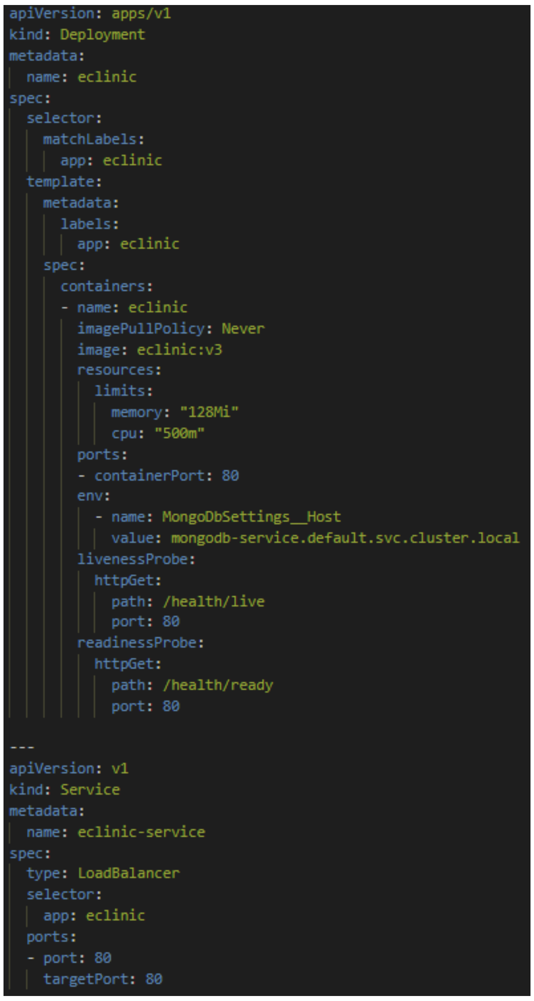
7.2.2 Kubernetes

The Docker images built by the above configuration is used by Kubernetes to run its container orchestration system. The configuration file in **Error! Reference source not found.** shows that the image has been declared as *eclinic:v3*. The image is configured to run as a LoadBalancer service. This option exposes the service externally by automatically routing to different NodePort services which routes to a ClusterIP service (only reachable within the cluster).



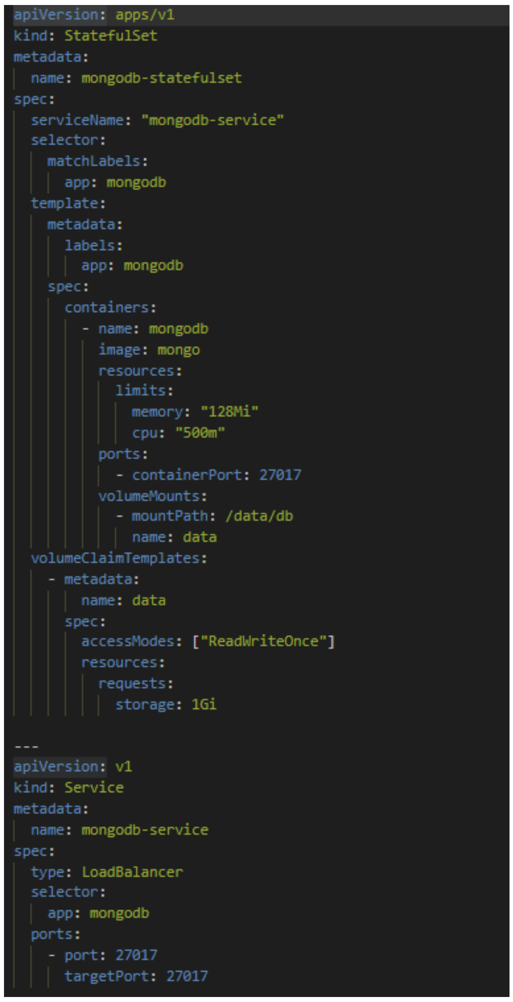
```
k8s_POD_eclinic-84bbd69888-ndw97_default_0f928f00-dd48-4e11-8ec3-a136f3a287c3_12  k8s.gcr.io/paus...
RUNNING
k8s_POD_mongodb-statefulset-0_default_47f9b1d1-e1c6-4c34-8f2f-a40abd9daffb_12  k8s.gcr.io/paus...
RUNNING
k8s_eclinic_eclinic-84bbd69888-ndw97_default_0f928f00-dd48-4e11-8ec3-a136f3a287c3_12  sha256:f2b943a...
RUNNING
k8s_mongodb_mongodb-statefulset-0_default_47f9b1d1-e1c6-4c34-8f2f-a40abd9daffb_12  mongo
RUNNING
```

Figure 48: Kubernetes pods running ASP.NET backend and MongoDB shown on Docker Dashboard



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: eclinic
spec:
  selector:
    matchLabels:
      app: eclinic
  template:
    metadata:
      labels:
        app: eclinic
    spec:
      containers:
      - name: eclinic
        imagePullPolicy: Never
        image: eclinic:v3
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 80
        env:
          - name: MongoDbSettings__Host
            value: mongodb-service.default.svc.cluster.local
        livenessProbe:
          httpGet:
            path: /health/live
            port: 80
        readinessProbe:
          httpGet:
            path: /health/ready
            port: 80

---
apiVersion: v1
kind: Service
metadata:
  name: eclinic-service
spec:
  type: LoadBalancer
  selector:
    app: eclinic
  ports:
  - port: 80
    targetPort: 80
```

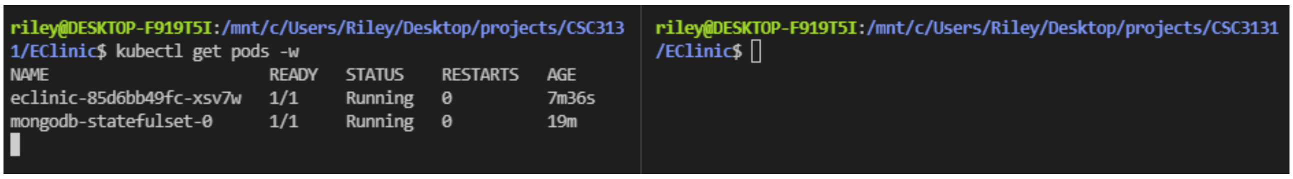
Figure 49: Kubernetes configuration for EClinic ASP.NET backend



```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mongodb-statefulset
spec:
  serviceName: "mongodb-service"
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
        - name: mongodb
          image: mongo
          resources:
            limits:
              memory: "128Mi"
              cpu: "500m"
          ports:
            - containerPort: 27017
          volumeMounts:
            - mountPath: /data/db
              name: data
  volumeClaimTemplates:
    - metadata:
        name: data
      spec:
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: 1Gi

---
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
spec:
  type: LoadBalancer
  selector:
    app: mongodb
  ports:
    - port: 27017
      targetPort: 27017
```

Figure 50: Kubernetes configuration for MongoDB instance

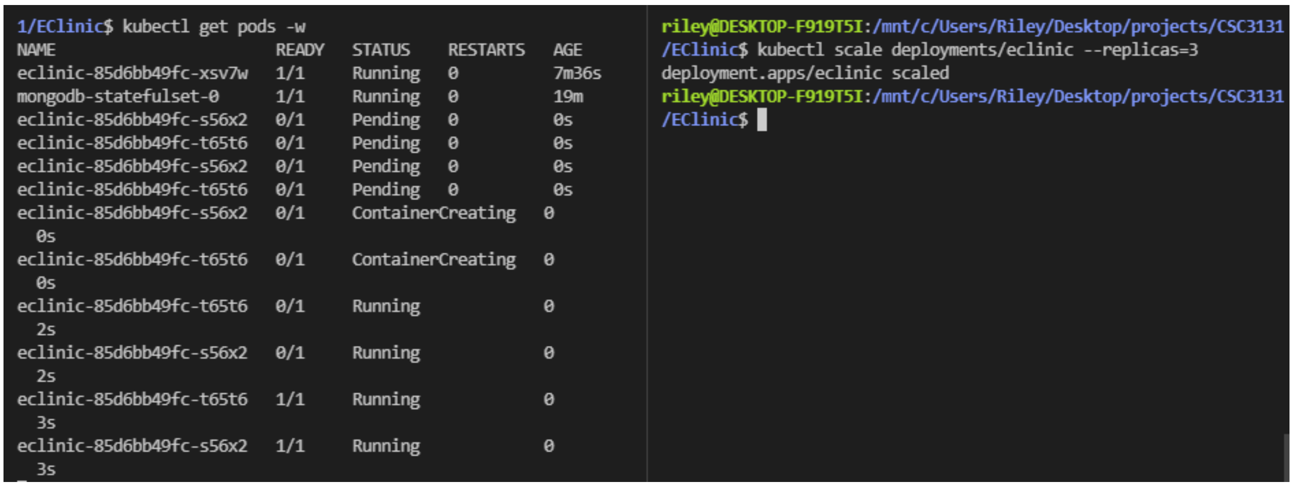
Kubernetes allows scaling of the pods on which the instance of the application runs through a simple command. Figure 51 and Figure 52 shows pod status logs from before and after scaling the application from one to three identical pods by specifying –replicas=3 with the scale command.



```
riley@DESKTOP-F919T5I:/mnt/c/Users/Riley/Desktop/projects/CSC313
1/EClinic$ kubectl get pods -w
NAME                       READY   STATUS    RESTARTS   AGE
eclinic-85d6bb49fc-xsv7w   1/1     Running   0          7m36s
mongodb-statefulset-0      1/1     Running   0          19m
```

```
riley@DESKTOP-F919T5I:/mnt/c/Users/Riley/Desktop/projects/CSC3131
/EClinic$
```

Figure 51: Currently running pods with -w option for logging

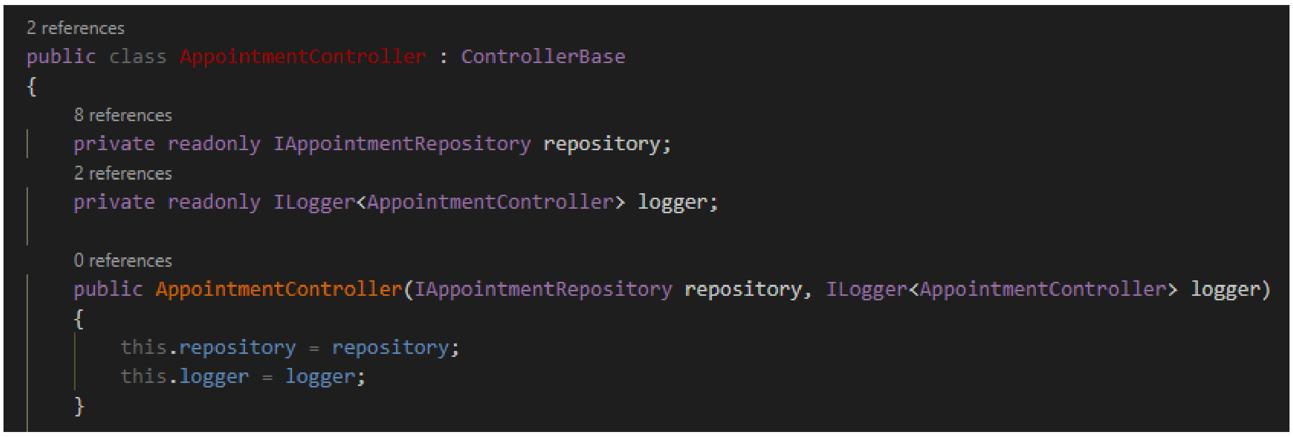


```
1/EClinic$ kubectl get pods -w
NAME                        READY   STATUS    RESTARTS   AGE
eclinic-85d6bb49fc-xsv7w    1/1     Running   0          7m36s
mongodb-statefulset-0       1/1     Running   0          19m
eclinic-85d6bb49fc-s56x2    0/1     Pending   0          0s
eclinic-85d6bb49fc-t65t6    0/1     Pending   0          0s
eclinic-85d6bb49fc-s56x2    0/1     Pending   0          0s
eclinic-85d6bb49fc-t65t6    0/1     Pending   0          0s
eclinic-85d6bb49fc-s56x2    0/1     ContainerCreating   0
  0s
eclinic-85d6bb49fc-t65t6    0/1     ContainerCreating   0
  0s
eclinic-85d6bb49fc-t65t6    0/1     Running             0
  2s
eclinic-85d6bb49fc-s56x2    0/1     Running             0
  2s
eclinic-85d6bb49fc-t65t6    1/1     Running             0
  3s
eclinic-85d6bb49fc-s56x2    1/1     Running             0
  3s
```

```
riley@DESKTOP-F919T5I:/mnt/c/Users/Riley/Desktop/projects/CSC3131
/EClinic$ kubectl scale deployments/eclinic --replicas=3
deployment.apps/eclinic scaled
riley@DESKTOP-F919T5I:/mnt/c/Users/Riley/Desktop/projects/CSC3131
/EClinic$
```

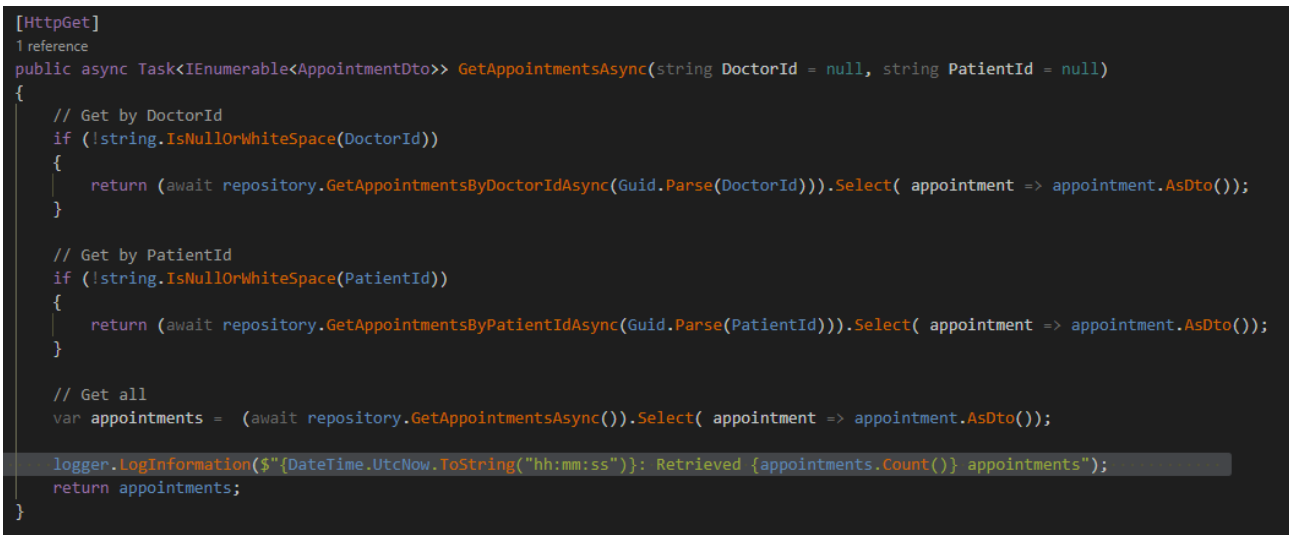
Figure 52: Currently running pods after scaling backend with --replicas=3

In order to confirm that the load balancing to different pods works correctly, I added a Logger dependency to AppointmentController, which logs the output of a GetAppointmentsAsync method (Figure 53, Figure 54).



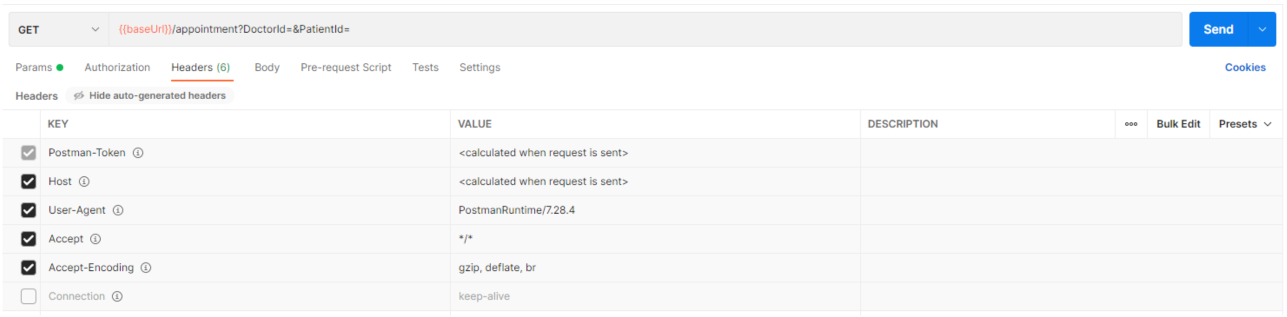
```csharp
2 references
public class AppointmentController : ControllerBase
{
    8 references
    private readonly IAppointmentRepository repository;
    2 references
    private readonly ILogger<AppointmentController> logger;

    0 references
    public AppointmentController(IAppointmentRepository repository, ILogger<AppointmentController> logger)
    {
        this.repository = repository;
        this.logger = logger;
    }
```

Figure 53: Adding a Logger as a dependency



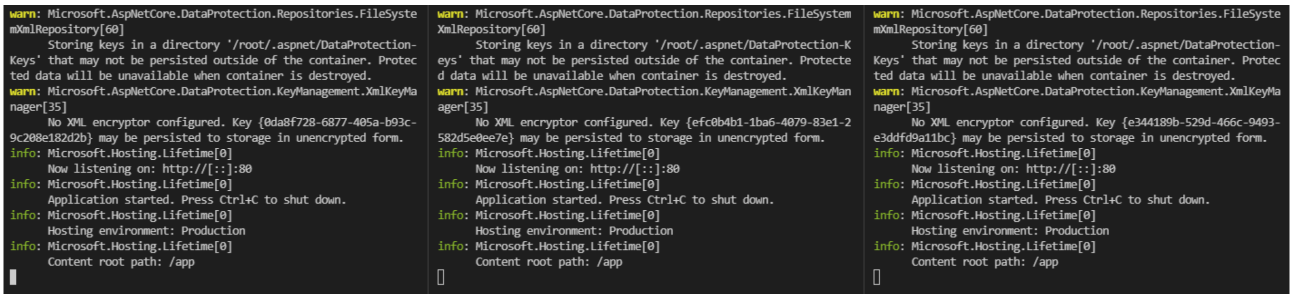
```csharp
[HttpGet]
1 reference
public async Task<IEnumerable<AppointmentDto>> GetAppointmentsAsync(string DoctorId = null, string PatientId = null)
{
    // Get by DoctorId
    if (!string.IsNullOrWhiteSpace(DoctorId))
    {
        return (await repository.GetAppointmentsByDoctorIdAsync(Guid.Parse(DoctorId))).Select( appointment => appointment.AsDto());
    }

    // Get by PatientId
    if (!string.IsNullOrWhiteSpace(PatientId))
    {
        return (await repository.GetAppointmentsByPatientIdAsync(Guid.Parse(PatientId))).Select( appointment => appointment.AsDto());
    }

    // Get all
    var appointments =  (await repository.GetAppointmentsAsync()).Select( appointment => appointment.AsDto());

    logger.LogInformation($"{DateTime.UtcNow.ToString("hh:mm:ss")}: Retrieved {appointments.Count()} appointments");
    return appointments;
}
```

Figure 54: Logging information about retrieved appointments for demonstrative purposes


GET {{baseUrl}}/appointment?DoctorId=&PatientId=

	KEY	VALUE	DESCRIPTION
✓	Postman-Token	<calculated when request is sent>	
✓	Host	<calculated when request is sent>	
✓	User-Agent	PostmanRuntime/7.28.4	
✓	Accept	*/*	
✓	Accept-Encoding	gzip, deflate, br	
	Connection	keep-alive	

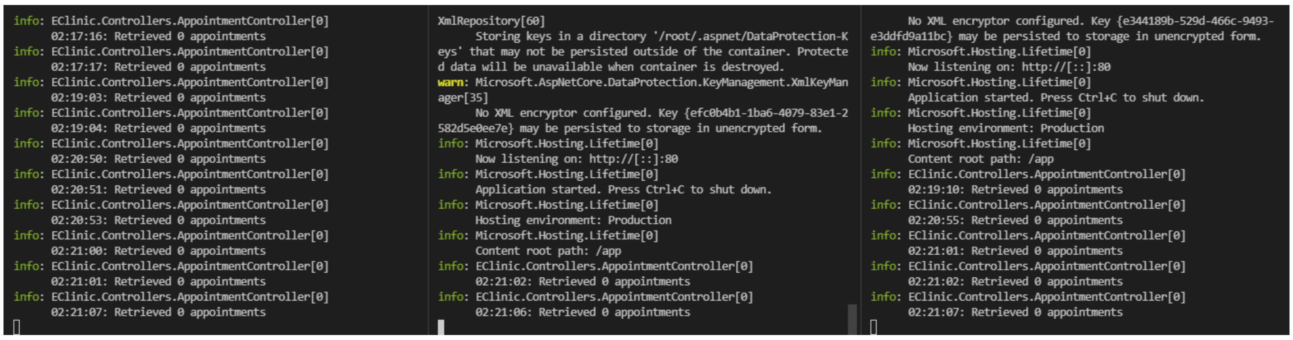
Figure 55: Sending requests in quick succession in Postman to test LoadBalancer


```
warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSyste
mXmlRepository[60]
      Storing keys in a directory '/root/.aspnet/DataProtection-
Keys' that may not be persisted outside of the container. Protec
ted data will be unavailable when container is destroyed.
warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyMa
nager[35]
      No XML encryptor configured. Key {0da8f728-6877-405a-b93c-
9c208e182d2b} may be persisted to storage in unencrypted form.
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
```

```
warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSystem
XmlRepository[60]
      Storing keys in a directory '/root/.aspnet/DataProtection-K
eys' that may not be persisted outside of the container. Protecte
d data will be unavailable when container is destroyed.
warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyMan
ager[35]
      No XML encryptor configured. Key {efc0b4b1-1ba6-4079-83e1-2
582d5e0ee7e} may be persisted to storage in unencrypted form.
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
```

```
warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSyste
mXmlRepository[60]
      Storing keys in a directory '/root/.aspnet/DataProtection-
Keys' that may not be persisted outside of the container. Protec
ted data will be unavailable when container is destroyed.
warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyMa
nager[35]
      No XML encryptor configured. Key {e344189b-529d-466c-9493-
e3ddfd9a11bc} may be persisted to storage in unencrypted form.
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
```

Figure 56: Console output from ASP.NET backend after start-up


```
info: EClinic.Controllers.AppointmentController[0]
      02:17:16: Retrieved 0 appointments
info: EClinic.Controllers.AppointmentController[0]
      02:17:17: Retrieved 0 appointments
info: EClinic.Controllers.AppointmentController[0]
      02:19:03: Retrieved 0 appointments
info: EClinic.Controllers.AppointmentController[0]
      02:19:04: Retrieved 0 appointments
info: EClinic.Controllers.AppointmentController[0]
      02:20:50: Retrieved 0 appointments
info: EClinic.Controllers.AppointmentController[0]
      02:20:51: Retrieved 0 appointments
info: EClinic.Controllers.AppointmentController[0]
      02:20:53: Retrieved 0 appointments
info: EClinic.Controllers.AppointmentController[0]
      02:21:00: Retrieved 0 appointments
info: EClinic.Controllers.AppointmentController[0]
      02:21:01: Retrieved 0 appointments
info: EClinic.Controllers.AppointmentController[0]
      02:21:07: Retrieved 0 appointments
```

```
XmlRepository[60]
      Storing keys in a directory '/root/.aspnet/DataProtection-K
eys' that may not be persisted outside of the container. Protecte
d data will be unavailable when container is destroyed.
warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyMan
ager[35]
      No XML encryptor configured. Key {efc0b4b1-1ba6-4079-83e1-2
582d5e0ee7e} may be persisted to storage in unencrypted form.
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
info: EClinic.Controllers.AppointmentController[0]
      02:21:02: Retrieved 0 appointments
info: EClinic.Controllers.AppointmentController[0]
      02:21:06: Retrieved 0 appointments
```

```
      No XML encryptor configured. Key {e344189b-529d-466c-9493-
e3ddfd9a11bc} may be persisted to storage in unencrypted form.
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
info: EClinic.Controllers.AppointmentController[0]
      02:19:10: Retrieved 0 appointments
info: EClinic.Controllers.AppointmentController[0]
      02:20:55: Retrieved 0 appointments
info: EClinic.Controllers.AppointmentController[0]
      02:21:01: Retrieved 0 appointments
info: EClinic.Controllers.AppointmentController[0]
      02:21:02: Retrieved 0 appointments
info: EClinic.Controllers.AppointmentController[0]
      02:21:07: Retrieved 0 appointments
```

Figure 57: Console output from ASP.NET backend after sending multiple GET requests to /appointment showing load balancing by Kubernetes

Sending multiple requests in quick succession using Postman (Figure 55) shows that the requests are routed to the three pods with a round-robin DNS, where each pod is chosen equally in succession (Figure 56, Figure 57) [11]. In an actual deployment scenario on a cloud service, Kubernetes allows versatile scaling of the operation by increasing the number of instances of pods running, even if the pods are spread out over different physical machines.

7.3 Evaluation

I was not able to connect to MongoDB instance when a username and password was set in the mongodb.yaml configuration file (Figure 50) due to an UserNotFound error, even after confirming that the MongoDB connection string was correctly formed by the backend. I confirmed that the DNS was working correctly by running a netstat command from ASP.NET pod from a coredns pod [12] [13]. I tried solutions suggested after posting on StackOverflow, to no avail. Although in terms of security, this is not a problem as the MongoDB pod is not configured to be reachable from outside Kubernetes, in a deployment scenario in an industry setting, this is poor practice.

8 Bibliography

[1] React, “Components and Props,” 2021. [Online]. Available: https://reactjs.org/docs/components-and-props.html. [Accessed 20 12 2021].

[2] Stack Overflow, “Stack Overflow Trends,” 20 12 2021. [Online]. Available: https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular%2Csvelte%2Cangularjs.

[3] React, “Create a New React App,” 2021. [Online]. Available: https://reactjs.org/docs/create-a-new-react-app.html. [Accessed 20 12 2021].

[4] StackOverflow, “Trouble with CORS Policy and .NET Core 3.1,” 10 2021. [Online]. Available: https://stackoverflow.com/questions/59317789/trouble-with-cors-policy-and-net-core-3-1. [Accessed 20 12 2021].

[5] Microsoft, “Dependency injection in ASP.NET Core,” 2021. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-6.0. [Accessed 20 12 2021].

[6] Microsoft, “Dependency Injection,” 2016. [Online]. Available: https://jakeydocs.readthedocs.io/en/latest/fundamentals/dependency-injection.html.

[7] Microsoft, “Testing in .NET,” 09 15 2021. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/core/testing/. [Accessed 20 12 2021].

[8] P. Gomes, “Configuring CI for .Net Core using Travis CI and Xunit,” 06 04 2017. [Online]. Available: https://medium.com/@pjbgf/configuring-ci-for-net-core-using-travis-ci-and-xunit-cc0f809df4fb. [Accessed 20 12 2021].

[9] Microsoft, “Nuget NETSDK1 Package ERROR,” 07 01 2020. [Online]. Available: https://developercommunity.visualstudio.com/t/nuget-netsdk1-package-error/876789. [Accessed 20 12 2021].

[10] Newcastle University, “CSC3131-week03-Docker.pptx,” [Online]. Available: https://ncl.instructure.com/courses/39992/files/4849415?module_item_id=1991352. [Accessed 20 12 2021].

[11] Kubernetes, “Service,” 20 11 2021. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/service/. [Accessed 20 12 2021].

[12] Kubernetes, “Debugging DNS Resolution,” 01 10 2021. [Online]. Available: https://kubernetes.io/docs/tasks/administer-cluster/dns-debugging-resolution/. [Accessed 20 12 2021].

[13] Infoblox, “CoreDNS for Kubernetes Service Discovery,” 7 11 2016. [Online]. Available: https://blogs.infoblox.com/community/coredns-for-kubernetes-service-discovery/. [Accessed 20 12 2021].

[14] Kubernetes, “Services, Load Balancing, and Networking,” 2021. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/_print/. [Accessed 20 12 2021].

[15] Microsoft, “Unit testing best practices with .NET Core and .NET Standard,” 29 11 2021. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices. [Accessed 20 12 2021].