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 3](#_Toc90932620)

[2 Frontend (React) 3](#_Toc90932621)

[2.1 Reason for choice 3](#_Toc90932622)

[2.2 Implementation Details 3](#_Toc90932623)

[2.2.1 Initial Setup 3](#_Toc90932624)

[2.2.2 Fixing CORS issue 4](#_Toc90932625)

[2.2.3 Implementation of the interface and private routes 5](#_Toc90932626)

[2.3 Evaluation 9](#_Toc90932627)

[3 Backend (ASP.Net) 9](#_Toc90932628)

[3.1 Reason for choice 9](#_Toc90932629)

[3.2 Implementation Details 10](#_Toc90932630)

[3.2.1 DTOs (Data Transfer Objects) 10](#_Toc90932631)

[3.2.2 Dependency Injection for classes using singletons 11](#_Toc90932632)

[3.2.3 Business logic for creating valid appointments 11](#_Toc90932633)

[3.3 Evaluation 13](#_Toc90932634)

[4 Observability and Maintainability 13](#_Toc90932635)

[4.1 Health Checks NuGet Package and integration with Kubernetes 14](#_Toc90932636)

[4.2 Postman 16](#_Toc90932637)

[4.3 Evaluation 16](#_Toc90932638)

[5 Continuous Integration (xUnit) 17](#_Toc90932639)

[5.1 Evaluation 18](#_Toc90932640)

[6 Continuous Deployment 19](#_Toc90932641)

[6.1 TravisCI 19](#_Toc90932642)

[6.2 Github Workflows 19](#_Toc90932643)

[6.3 Evaluation 21](#_Toc90932644)

[7 Scalability and Load balancing (Docker and Kubernetes) 22](#_Toc90932645)

[7.1 Reason for choice 22](#_Toc90932646)

[7.2 Implementation Details 22](#_Toc90932647)

[7.2.1 Docker 22](#_Toc90932648)

[7.2.2 Kubernetes 23](#_Toc90932649)

[7.3 Evaluation 26](#_Toc90932650)

[8 Bibliography 27](#_Toc90932651)

# 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.

# Frontend (React)

## 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].

## Implementation Details

### Initial Setup

Text

Description automatically generated

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]*.*

Text

Description automatically generated

Figure 2: Running the development environment using npm start



Figure 3: Installing required packages from NPM

A picture containing graphical user interface

Description automatically generated

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

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

### Fixing CORS issue

Text

Description automatically generated

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

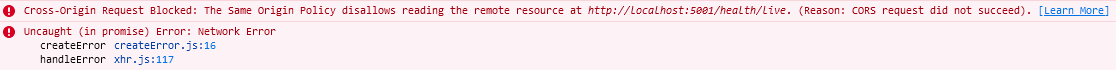


Figure 6: Health checks failed due to CORS error

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).

Graphical user interface, text, application

Description automatically generated

Figure 7: Disabled Https Redirection to fix CORS issue

Text

Description automatically generated

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].

### Implementation of the interface and private routes

Graphical user interface, application, email

Description automatically generated

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.

Text

Description automatically generated

Figure 10: useEffect hooks for retrieving list of appointments

Text

Description automatically generated

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.

Graphical user interface, text, application, email

Description automatically generated

Figure 12: Main page when not logged in

Graphical user interface, application

Description automatically generated

Figure 13: Login screen

Graphical user interface, application

Description automatically generated

Figure 14: Main page after login

Text

Description automatically generated

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).

Text

Description automatically generated

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

## Evaluation

Graphical user interface, text, application, email

Description automatically generated

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.

# Backend (ASP.Net)

## 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.

## Implementation Details

Graphical user interface, text

Description automatically generated

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.

### DTOs (Data Transfer Objects)

Text

Description automatically generated

Figure 19: Base DTO

Text

Description automatically generated

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].

### Dependency Injection for classes using singletons

Text

Description automatically generated

Figure 21: Class constructor for AppointmentController

Text

Description automatically generated

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.

### 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.

Text

Description automatically generated

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

Text

Description automatically generated

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.

Text

Description automatically generated

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.

## 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.

# 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.

## Health Checks NuGet Package and integration with Kubernetes

Text

Description automatically generated

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.

Graphical user interface, text, application, Teams

Description automatically generated

Figure 27: Liveness check showing that the server is running

Graphical user interface, text, application, email, Teams

Description automatically generated

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

Text

Description automatically generated

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.

## Postman

Graphical user interface, text, application

Description automatically generated

Figure 30: Postman collection generated by Swagger

Graphical user interface, application

Description automatically generated

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.

## 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.

# 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.

Text

Description automatically generated

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

Text

Description automatically generated

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



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.

Graphical user interface, text

Description automatically generated

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

Text

Description automatically generated

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

Text

Description automatically generated

Figure 37: Automatically running every test using dotnet test command

## Evaluation

Text

Description automatically generated

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.

# Continuous Deployment

## 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.

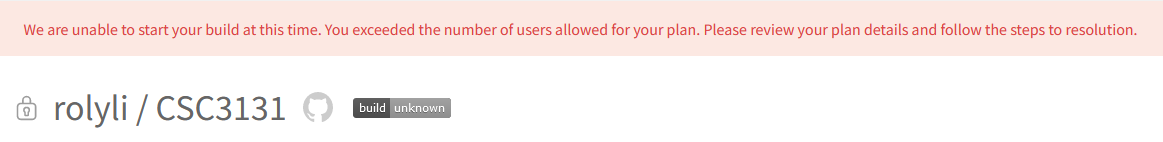


Figure 39: TravisCI showing user exceeded build error despite no builds made

A screenshot of a computer

Description automatically generated with medium confidence

Figure 40: .travis.yml configuration for TravisCI

## 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.

Graphical user interface, text, application, Word

Description automatically generated

Figure 41: GitHub Workflows configuration

Text

Description automatically generated

Figure 42: NETSDK1064 error from building the EClinic project

Text

Description automatically generated

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

## Evaluation

Graphical user interface, table

Description automatically generated with medium confidence

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.

# Scalability and Load balancing (Docker and Kubernetes)

## 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

## Implementation Details

### Docker

Text

Description automatically generated

Figure 45: Docker configuration for EClinic ASP.NET backend

Text

Description automatically generated

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.

Text

Description automatically generated

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.

### Kubernetes

The Docker images built by the above configuration is used by Kubernetes to run its container orchestration system. The configuration file (Figure 49) 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).

Text

Description automatically generated

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

|  |  |
| --- | --- |
| Text  Description automatically generated  Figure 49: Kubernetes configuration for EClinic ASP.NET backend | A picture containing timeline  Description automatically generated  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.

Graphical user interface, text, application

Description automatically generated

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

Text

Description automatically generated

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).

Text

Description automatically generated

Figure 53: Adding a Logger as a dependency

Text

Description automatically generated

Figure 54: Logging information about retrieved appointments for demonstrative purposes

Graphical user interface, text, application, email

Description automatically generated

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

Text

Description automatically generated

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

Text

Description automatically generated

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.

## 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.

# 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]. |