# Simple clinic booking system

Student exercise

rev 1.0

Table of Contents

[Simple clinic booking system 1](#_Toc515619135)

[Introduction 2](#_Toc515619136)

[Specification 3](#_Toc515619137)

[Application description (customer input) 3](#_Toc515619138)

[Functional specification 3](#_Toc515619139)

[Technical constraints 4](#_Toc515619140)

[Performance requirements 4](#_Toc515619141)

[Exercise 5](#_Toc515619142)

[Planning 5](#_Toc515619143)

[Design 5](#_Toc515619144)

[Implementation 5](#_Toc515619145)

[Source control 6](#_Toc515619146)

[Services/back-end 6](#_Toc515619147)

[Database access/Repository layer 6](#_Toc515619148)

[Front-end 6](#_Toc515619149)

[Validation 7](#_Toc515619150)

[Logging 7](#_Toc515619151)

[Dependency injection/IOC container 7](#_Toc515619152)

[Progress tracking 8](#_Toc515619153)

# Introduction

This exercise is an introduction to programming a small full stack web application in C# using the Rio/Oceano Common Codebase architecture.

Learning to design and plan your work from a specification using a process similar to the development process used at Servelec.

This exercise should be completed on an individual basis but you can discuss with fellow students and use any resources either supplied or online.

Please read through the entire document first before you start. If you’re still not sure what you need to do or where to start, do ask your mentor or each other.

# Specification

## Application description (customer input)

Hospital X requires a simple clinic booking system in which they can book existing patients into a clinic that can deal with their ailment on the day.

Hospital staff requires a single overview screen a summary of all appointments in the booking system and need to see which patient has the appointment, the clinic and the time of appointment.

The overview screen should be able to filter by patient and clinic and by a date range.

All columns should be sortable.

Users should be able to book patients into a clinic and capture the following information:

* The patient attending the appointment
* The appointment start date and time
* The intended duration
* The clinic
* The urgency (optional)
* The appointment type (optional)
* Contact phone number or email address

The appointment can be at any clinic provided the requested slot is available for the requested specialty.

Users should be able to edit an appointment.

Users should be able to cancel an appointment.

User should be able to undo a cancellation if the original appointment slot is still available.

## Functional specification

A clinic can only accept patients that are already on the system.

A clinic can only accept patients for a single specialty.

Appointments can be of variable length: 5, 10 and 15 minutes.

The appointment urgency is either routine, urgent or 2 week wait.

The appointment type is either a new appointment or a follow-up.

Only one patient can have an appointment at any time. This means there cannot be multiple patients on a single appointment.

## Technical constraints

The application should adhere to the company architecture.

The application needs to be an ASPX web application.

The minimum resolution that the application should support is 1024x768 pixels.

## Performance requirements

The overview screen should display its data within 3 seconds.

# Exercise

Let’s build the application.

## Planning

We’d like you to plan and design your solution before you start building. We expect you to be able to complete the exercise in 2 to 3 weeks but that’s not set in stone.

You may need to do a quick rough design first so you get an idea of how you break down your implementation. This is a Work Breakdown Structure or WBS.

Once you know what your WBS is, populate the WBS.xlsx spreadsheet and your estimates. You can have subtasks where appropriate. The sheet automatically calculates your final estimate from your worst case, best case and likely estimate. It’s easiest to do it in hours.

Hint: Don’t forget to estimate some time for your design phase in the next section.

Consider creating the services layer first and with a simple console front-end initially so you know your services work before you do any front-end work.

## Design

Assuming you’ve done a rough design in your planning phase, create a final design showing components and classes where possible. This can still be high level and doesn’t have to be UML, as long as it’s clear. This will help you to stick to what you set out to build and will help in discussions with your mentor or developers.

Remember the application needs to be in line with the company architecture.

Hint: You should have 1 or more WCF services which the front-end communicates with.

For the web front-end the only requirement is that the functionality in the specification is implemented so it doesn’t matter too much what it looks like. Do however draw some mock-up screens so you know what needs building. If screens have certain behaviour (e.g. Select this and then this gets loaded) write a short description.

## Implementation

Before you start building your application, discuss your WBS and design with your mentor.

We’ve compiled a number or resources and examples for you do help you do the implementation. If you can’t find what you’re looking for, research it on Google for it in the first instance. StackOverflow is a popular source of very good information. Your mentors or fellow students are able to help you if you’ve exhausted all resources.

There is an example application in the exercise folder that implements a barebones set of WCF services and fetches data from these services in a console application.

\\hs-fs-01\Products\Oceano\New Starters\Exercise\Example\Exercise

It’s not a complete implementation and doesn’t fully implement the required architecture. For instance, it doesn’t have a business layer, no DTOs and the logger is very basic.

**IMPORTANT**: **You can use this example application as a reference but don’t open the solution from the network. Please make a copy onto your own development machine before opening.**

There are no pre-built dependencies to this exercise. We can simply rely on the NuGet package manager to fetch them for you.

### Source control

You all have an area in SVN to commit your code to:

http://forth:8080/repository/sandbox/jdoe

Where instead of jdoe use the first letter of your name and your surname in lowercase without any hyphens or spaces.

It’s advisable to check out the sandbox folder in C:\environments\sandbox\jdoe

### Services/back-end

For the services you’ll need:

* Fluent NHibernate
* Unity
* Apache Log4Net

The example application hosts its services in IIS. This is the preferred way but not mandatory for the exercise. See the IIS Setup.txt in the example folder for details how this is set up.

### Database access/Repository layer

You need to create a local SQL Server database. We have supplied you with existing data but there are no tables yet.

The data can be found in [\\hs-fs-01\Products\Oceano\New Starters\Exercise\Exercise Data.xlsx](file:///\\hs-fs-01\Products\Oceano\New%20Starters\Exercise\Exercise%20Data.xlsx)

We use (Fluent) NHibernate to access your SQL Server database. NHibernate is an ORM. This requires classes to be created to represent your database tables in so called class maps.

It’s also possible to map to classes using store procedures (research if you think you need this).

If you don’t know what an ORM is or don’t understand how (Fluent) NHibernate works, do some research or ask someone.

The example application in the exercise folder shows how to configure it, connect to your database, create a class map and do a query.

### Front-end

For the web front-end create an ASP.NET web application complimented by the jQuery and knockout.js javascript frameworks should you need them. NuGet can add these to your project.

Consider using proper CSS styling where appropriate like headings and layout, but don’t worry too much about how pretty it looks. Your user interface should definitely be clear and useable!

Hint: When calling your services you can add them as a service reference in your project. These can be re-synchronised as you make changes to your services interface.

### Validation

Validation is important! Make sure you validate inputs on the front-end as well as on the back-end.

Remember, WCF services expose their methods so can be used by another front-end (i.e. console app) so wrong input shouldn’t break your service!

### Logging

Logging will help you in bug fixing your application.

We use Log4Net within Servelec to log to a file. As a general rule, you shouldn’t swallow exceptions without logging. Also, if you need to inform the user about something that went wrong in the service call, pass something useful back to the caller (hint: WCF FaultException).

It’s advisable to have entry and exit logging in your service methods so you can see what “has” been called before your application crashed.

The example application in the exercise folder has an injected logger but doesn’t configure or use log4net.

### Dependency injection/IOC container

We use Unity to dependency inject services, the business factory and the repository factory.

The example application in the exercise folder shows how to configure the Unity container (web.config) and inject your different components.

If you don’t know what dependency injection or an IOC container is, do some research or ask someone.

## Progress tracking

As you implement your application, please record your actual time spent in the WBS spreadsheet you created for the planning phase. Don’t panic if you don’t make your original estimate. It’s there to get used to a tracking process and teach you to learn how to estimate.

Hint: If you’ve spent half your time on a task already but you still have a more than half to go, it’s a good time to have another look at your task and re-estimate.