PViMS Developers Manual

September 2020

This report is made possible by the generous support of the American people through the US Agency for International Development (USAID), under the terms of cooperative agreement number AID-OAA-A-11-00021. The contents are the responsibility of Management Sciences for Health and do not necessarily reflect the views of USAID or the United States Government.

About SIAPS

The goal of the Systems for Improved Access to Pharmaceuticals and Services (SIAPS) Program is to assure the availability of quality pharmaceutical products and effective pharmaceutical services to achieve desired health outcomes. Toward this end, the SIAPS result areas include improving governance, building capacity for pharmaceutical management and services, addressing information needed for decision-making in the pharmaceutical sector, strengthening financing strategies and mechanisms to improve access to medicines, and increasing quality pharmaceutical services.

Recommended Citation

This report may be reproduced if credit is given to SIAPS. Please use the following citation.

SIAPS Program. 2018. *PViMS Programmers' Manual.* Submitted to the US Agency for International Development by the Systems for Improved Access to Pharmaceuticals and Services (SIAPS) Program. Arlington, VA: Management Sciences for Health.

Systems for Improved Access to Pharmaceuticals and Services
Pharmaceuticals and Health Technologies Group
Management Sciences for Health
4301 North Fairfax Drive, Suite 400
Arlington, VA 22203 USA
Telephone: 703.524.6575
Fax: 703.524.7898
E-mail: phtmis@msh.org
Website: www.siapsprogram.org

Contents

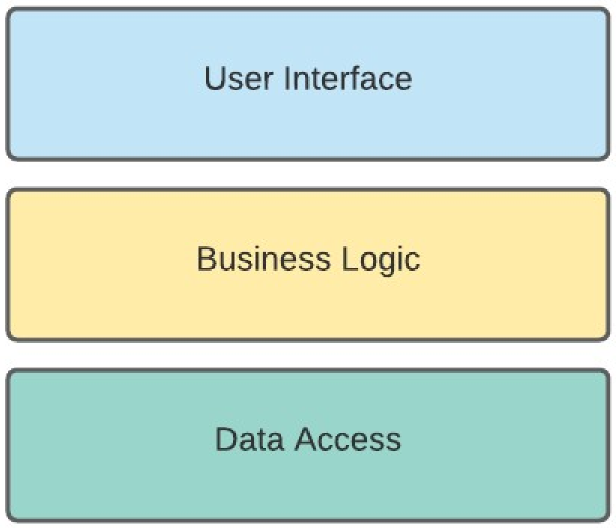
1 Program Model and Design

This section will give an overview of the data access method as well as patterns used within the development of the system

1.1 System Architecture

PViMS is implemented as a monolithic application which is entirely self-contained. The core of its behavior runs within its own process and the entire application is deployed as a single unit.

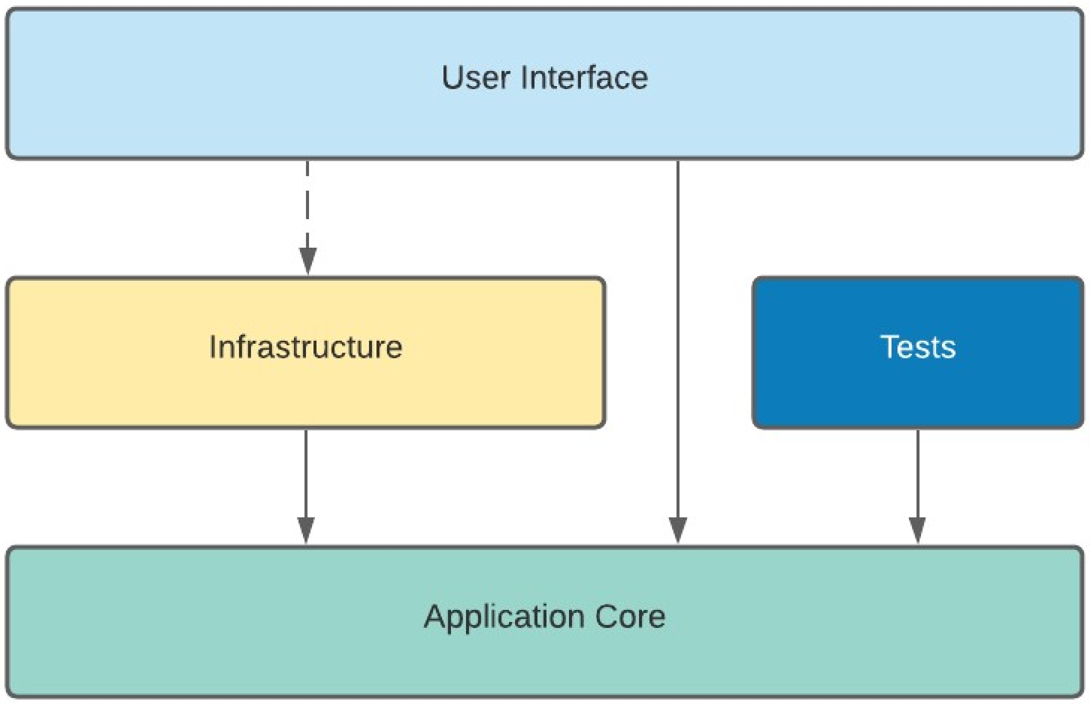
System complexity is managed through a layered architecture where functionality is separated following the separation of concerns principle. Layers represent logical separation within PViMS and can be defined as follows:



Based on this architecture, PViMS users make requests through the user interface layer, which interacts only with the business logic layer. This layer, in turn, calls the data access layer for data access requests. The user interface layer does not make any requests to the data access layer directly, nor does it interact with persistence directly through other means. In this way, each layer has its own defined responsibility, ensuring separation of concerns.

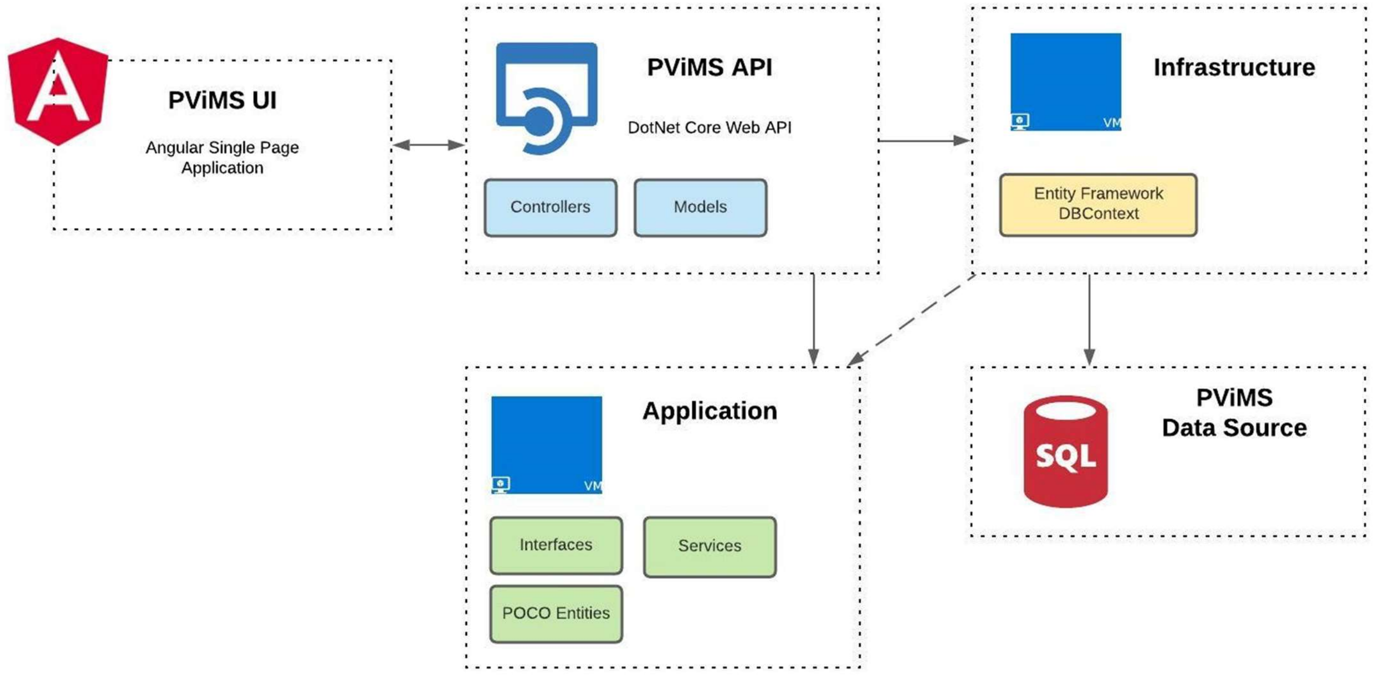
Through dependency injection and the implementation of domain driven design principles, this architecture is a form of clean architecture which puts the business logic and application model at the center of the application. Instead of having business logic depend on data access or other infrastructure concerns, this dependency is inverted: infrastructure and implementation details depend on the application core. This is achieved by defining abstractions, or interfaces, in the core which are then implemented by types defined in the Infrastructure layer.

The diagram below shows a more traditional horizontal layer diagram that better reflects the dependency between the UI and other layers within PViMS.



Note that the solid arrows represent compile-time dependencies, while the dashed arrow represents a runtime-only dependency. The user interface layer works with interfaces defined in the application core at compile time and doesn’t know about the implementation types defined in the Infrastructure layer. At run time these implementation types are required for the app to execute and are wired up to the application core interfaces via dependency injection.

The diagram below shows a more detailed view of the PViMS core application architecture:



1.1.1 User Interface – Angular Single Page Application (SPA)

The PViMS user interface is developed as a single page application wherein the application interacts with the user by dynamically rewriting the current page, rather than loading entire new pages from the server. This approach negates interruption of the user experience between successive pages, making PViMS behave more like a desktop application.

This approach provides several benefits:

- Improved application performance and consistency
- The burden of page redrawing is migrated from the server to the client, thereby reducing the load on the server
- The implementation of a fast and responsive user interface
- Enhanced user experience

Angular is a platform and framework for building single-page client applications using HTML and TypeScript. It implements core and optional functionality as a set of TypeScript libraries that are imported into the PViMS user interface.

1.1.2 Web API

The PViMS API layer is implemented as an ASP.Net Core RESTful API.

A RESTful API (Representational State Transfer) is designed to leverage existing protocols which in the case of a Web API is HTTP. REST has the ability to handle multiple types of calls, return different data formats and even change structurally with the correct implementation of hypermedia. Unlike SOAP, REST is not constrained to XML, but instead can return XML, JSON, YAML or any other format depending on what the client requests.

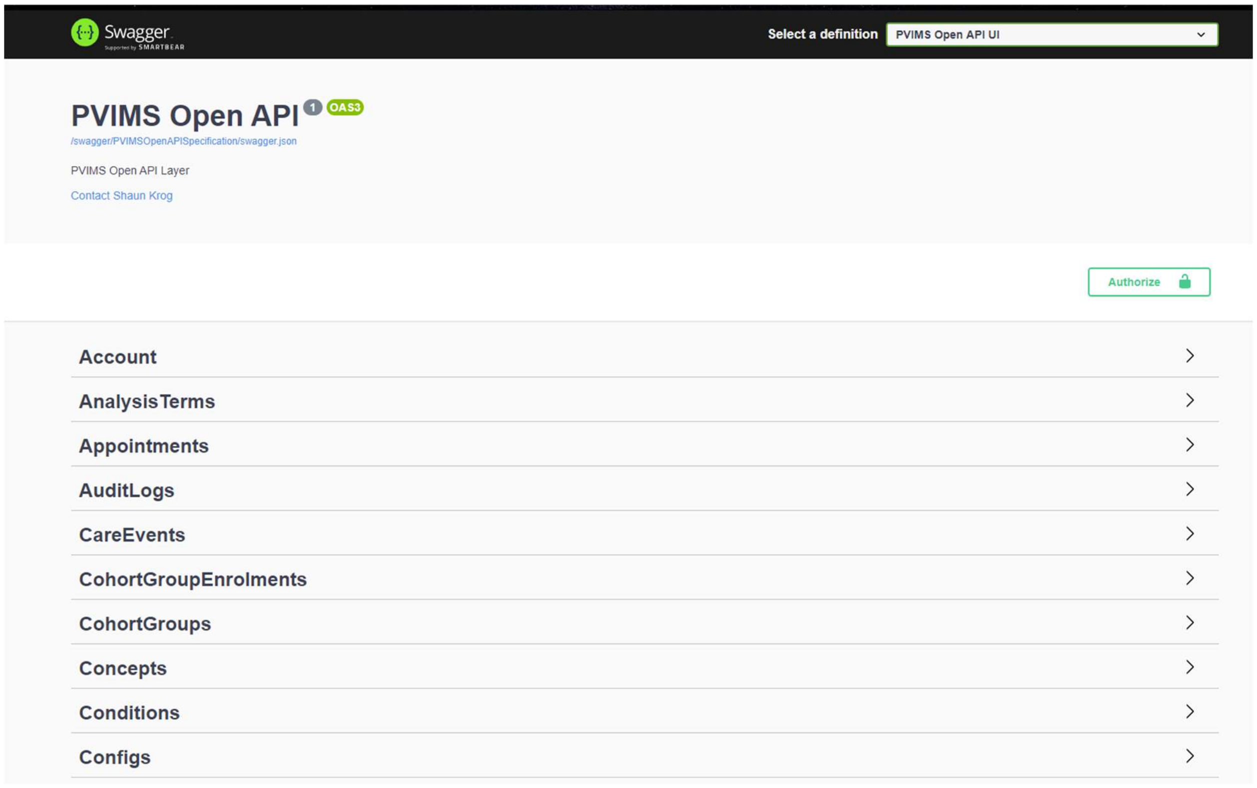
ASP.NET Core is a cross-platform, high-performance, open-source framework for building modern, cloud-enabled, Internet-connected apps.

ASP.NET Core provides the following benefits:

- Architected for testability.
- Ability to develop and run on Windows, macOS, and Linux.
- Open-source and community-focused.
- Integration of modern, client-side frameworks and development workflows.
- Built-in dependency injection.
- A lightweight, high-performance, and modular HTTP request pipeline.
- Ability to host on the following:
    - Kestrel
    - IIS
    - HTTP.sys
    - Nginx
    - Apache
    - Docker
- Side-by-side versioning.

When consuming a web API, understanding its various methods can be challenging for a developer. Swagger, also known as OpenAPI, solves the problem of generating useful documentation and help pages for web APIs. It provides benefits such as interactive documentation, client SDK generation, and API discoverability. OpenAPI is therefore a language-agnostic specification for describing REST APIs.

All PViMS API endpoints are documented through an OpenAPI documentation page which can be accessed by browsing to the root of the API service.



1.1.3 Infrastructure

Database access within PViMS is facilitated through Entity Framework v6.4.

1.1.3.1 Object-Relational Mapping

Object-relational mapping is a programming technique for converting data between incompatible type systems in object-oriented programming languages. This creates, in effect, a virtual object database that can be used from within the programming language.

1.1.3.2 Entity Framework

Entity Framework (EF) is an open source ORM framework for ADO.NET and is part of the DotNet Framework. It is a set of technologies that supports the development of data-oriented software applications. EF enables developers to work with data in the form of domain-specific objects and properties, such as customers and customer addresses, without having to concern themselves with the underlying database tables and columns where this data is stored. With EF, developers can work at a higher level of abstraction when they deal with data and can create and maintain data-oriented applications with less overhead.

Architecture is as follows:

- **Data source specific providers**, which abstract the ADO.NET interfaces to connect to the database when programming against the conceptual schema
- **Map provider**, a database-specific provider that translates the Entity SQL command tree into a query in the native SQL language of the database. It includes the Store-specific bridge, which is the component responsible for translating the generic command tree into a store-specific command tree
- **EDM parser and view mapping**, which takes the SDL specification of the data model and how it maps onto the underlying relational model and enables programming against the conceptual model. From the relational schema, it creates views of the data corresponding to the conceptual model. It aggregates information from multiple tables in order to aggregate them into an entity, and splits an update to an entity into multiple updates to whichever table(s) contributed to that entity
- **Query and update pipeline**, processes queries, filters and updates requests to convert them into canonical command trees which are then converted into store-specific queries by the map provider

- **Metadata services**, which handle all metadata related to entities, relationships and mappings
- **Transactions**, to integrate with transactional capabilities of the underlying store. If the underlying store does not support transactions, support for it needs to be implemented at this layer
- **Conceptual layer API**, the runtime that exposes the programming model for coding against the conceptual schema. It follows the ADO.NET pattern of using Connection objects to refer to the map provider, using Command objects to send the query, and returning *EntityResultSets* or *EntitySets* containing the result
- **Disconnected components,** which locally cache datasets and entity sets for using the ADO.NET Entity Framework in an occasionally connected environment
- **Embedded database**: ADO.NET Entity Framework includes a lightweight embedded database for client-side caching and querying of relational data.
- **Design tools**, such as Mapping Designer, are also included which simplifies the job of mapping a conceptual schema to the relational schema and specifying which properties of an entity type correspond to which table in the database
- **Programming layer**, which exposes the EDM as programming constructs which can be consumed by programming languages
- **Object services**, automatically generate code for CLR classes that expose the same properties as an entity, thus enabling instantiation of entities as .NET objects
- **Web services**, which expose entities as web services
- **High-level services**, such as reporting services which work on entities rather than relational data

1.1.3.3 LINQ to SQL

LINQ to SQL allows LINQ to be used to query the PViMS database. Since SQL Server data may reside on a remote server, and because SQL Server has its own query engine, it does not use the query engine of LINQ. Instead, it converts a LINQ query to an SQL query that is then sent to SQL Server for processing. However, since SQL Server stores the data as relational data and LINQ works with data encapsulated in objects, the two representations must be mapped to one another. For this reason, LINQ to SQL also defines a mapping framework. The mapping is done by defining classes that correspond to the tables in the database and containing all or a certain subset of the columns in the table as data members.

1.1.3.4 Code First design

PViMS is developed using Domain Driven Design (DDD) which is effectively a collection of principles and patterns that lead to software abstractions called domain models. These models encapsulate all complex PViMS business logic, closing the gap between reality and code. Domain driven design is about mapping business domain concepts into software artefacts, using the same common ubiquitous language.

Code-First is mainly used in Domain Driven Design and means that PViMS is developed through a Code-First approach. The PViMS data model is defined using C# classes and additional configuration can be performed using attributes within these classes. Model changes are committed to the PViMS database using Code-First Migrations.

Migrations allows us to have an ordered set of steps that describe how to upgrade (and downgrade) our PViMS database schema

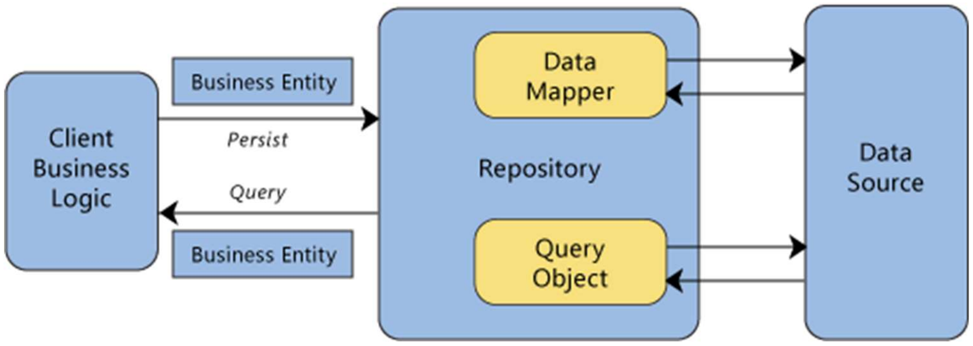
1.2 Design Patterns and Principals

A software design pattern is a general reusable solution to a common occurring problem within a given context. Design patterns are formalized best practices that describe the problem, the solution, when to apply the solution and its consequences, intended or unintended.

Design principles can alternatively help guide you toward architectural decisions that will result in clean, maintainable applications. These principles guide you toward building applications out of discrete components that are not tightly coupled to other parts of the application, but rather communicate through explicit interfaces or messaging systems.

In the case of PViMS, the following design patterns and principles have been implemented:

Pattern	Description
Dependency Injection	Dependency Injection (DI) is a design pattern that demonstrates how to create loosely coupled classes. DI is where one object supplies the dependencies of another object. A dependency is an object that can be used (a service). An injection is the passing of a dependency to a dependent object (a client) that would use it. The intent behind dependency injection is to decouple objects to the extent that no client code has to be changed simply because an object it depends on needs to be changed to a different one.
Repository and Unit of Work	A repository pattern separates the logic that retrieves the data and maps it to the entity model from the business logic that acts on the model. The business logic should be agnostic to the type of data that comprises the data source layer. The repository mediates between the data source layer and the business layers of the application. It queries the data source for the data, maps the data from the data source to a business entity, and persists changes in the business entity to the data source. A repository separates the business logic from the interactions with the underlying data source or Web service. There are two ways that the repository can query business entities. It can submit a query object to the client's business logic or it can use methods that specify the business criteria. In the latter case, the repository forms the query on the client's behalf. The repository returns a matching set of entities that satisfy the query. The following diagram shows the interactions of the repository with the client and the data source. The separation between the data and business

	tiers has three benefits: • It centralizes the data logic or Web service access logic • It provides a substitution point for the unit tests • It provides a flexible architecture that can be adapted as the overall design of the application evolves  Business Entity Client Business Logic Persist Query Business Entity Data Mapper Repository Query Object Data Source
Persistence Ignorance	Persistence ignorance (PI) refers to types that need to be persisted, but whose code is unaffected by the choice of persistence technology.
Separation of Concern	This principle asserts that software should be separated based on the kinds of work it performs. Architecturally, applications can be logically built to follow this principle by separating core business behavior from infrastructure and user-interface logic. Ideally, business rules and logic should reside in a separate project, which should not depend on other projects in the application. This separation helps ensure that the business model is easy to test and can evolve without being tightly coupled to low-level implementation details. Separation of concerns is a key consideration behind the use of layers in application architectures.

2 Development Environment

This section will give an overview of the development environment, including the method for source control and code sharing

2.1 Integrated Development Environment (IDE)

An IDE enables programmers to consolidate the different aspects of writing a computer program into one unified interface. Activities such as editing source code, building executables, and debugging are provided through this interface.

2.1.1 Visual Studio

Visual Studio is Microsoft's IDE and can be used edit PViMS source code, build executables for deployment and provide debugging for PViMS related issues. It is used to develop computer programs for Microsoft Windows, as well as web sites, web apps, web services and mobile apps.

The Visual Studio 2019 Community edition of the IDE is available as a free download and can be used for development and support of PViMS

2.1.2 Extensions

The following extensions are recommended for use when developing PViMS:

Visual Studio IntelliCode
Provides a set of AI-assisted capabilities that improve developer productivity with features like contextual IntelliSense, argument completion, code formatting, and style rule inference.

GitHub Extension for Visual Studio
The GitHub Extension for Visual Studio makes it easy to connect to and work with your repositories on GitHub and GitHub Enterprise from directly within Visual Studio 2015 or newer. Clone existing repositories or create new ones and start collaborating!

ReSharper
The legendary .NET productivity tool: find and fix errors and code smells; navigate and refactor; run unit tests and write quality code faster.

.

2.2 Source Control

Git is a version control system for tracking changes in source code files and coordinating work on those files among multiple software developers. PViMS utilizes GitHub, a Git repository hosting service, for management of PViMS source code. GitHub provides access control and several collaboration features such as a wiki and basic task management tools.

2.2.1 PViMS GitHub Repository

A GitHub repository is a storage space hosted within GitHub where you can access the PViMS project, source code and manuals.

The PViMS repository is hosted on the following URL:

https://github.com/MSH/PViMS-2

You will need to be registered on GitHub.com to be able to access the PViMS codebase.

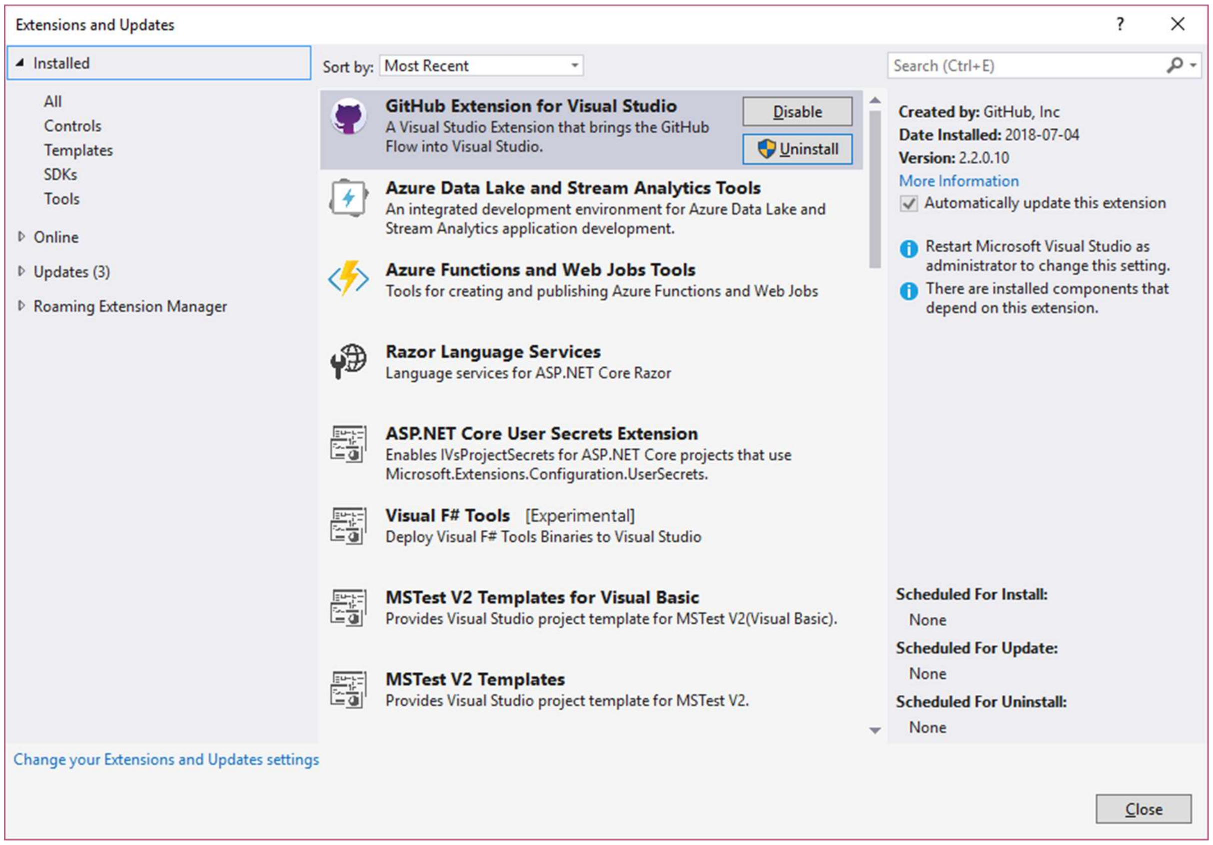
The following Git-related commands are useful to know:

Term	**Description**
Command Line	The command line is a non-native program that you can use to type text-based commands, known as prompts, to make changes within the repository
Version Control	In terms of software development, version control defines the ability to keep snap shots of source code files at various points in time, so that you never lose or overwrite code changes
Commit	A GIT commit effectively tracks changes to your local repository by creating a snap shot of your repository at the point of the commit. The commit itself stores what modifications have been made within the repository since the last commit.
Push	A GIT push command effectively publishes changes to the GitHub repository, thereby allowing your changes to be merged into the PViMS development branch.
Push	A GIT pull command effectively pulls latest changes from the GitHub

	repository and merges them into your local repository, thereby allowing you to consume other developer changes to PViMS into your local repository
Branch	Branching Is the mechanism that fundamentally allows multiple software developers to work on the same PViMS code at the same time. Branching allows each developer to branch out from the original code base and isolate their work from others. Different branches can ultimately be merged into any single branch, as long as it is part of the same repository.

2.2.2 GitHub Extension for Visual Studio

The easiest and most convenient way to connect to GitHub.com using Visual Studio is through the GitHub Extension, which can be added through the Visual Studio Extension Manager.



GitHub Extension for Visual Studio
A Visual Studio Extension that brings the GitHub Flow into Visual Studio.

Azure Data Lake and Stream Analytics Tools
An integrated development environment for Azure Data Lake and Stream Analytics application development.

Azure Functions and Web Jobs Tools
Tools for creating and publishing Azure Functions and Web Jobs

Razor Language Services
Language services for ASP.NET Core Razor

ASP.NET Core User Secrets Extension
Enables IVsProjectSecrets for ASP.NET Core projects that use Microsoft.Extensions.Configuration.UserSecrets.

Visual F# Tools [Experimental]
Deploy Visual F# Tools Binaries to Visual Studio

MSTest V2 Templates for Visual Basic
Provides Visual Studio project template for MSTest V2(Visual Basic).

MSTest V2 Templates
Provides Visual Studio project template for MSTest V2.

Created by: GitHub, Inc
Date Installed: 2018-07-04
Version: 2.2.0.10
More Information
[✓] Automatically update this extension
Restart Microsoft Visual Studio as administrator to change this setting.
There are installed components that depend on this extension.

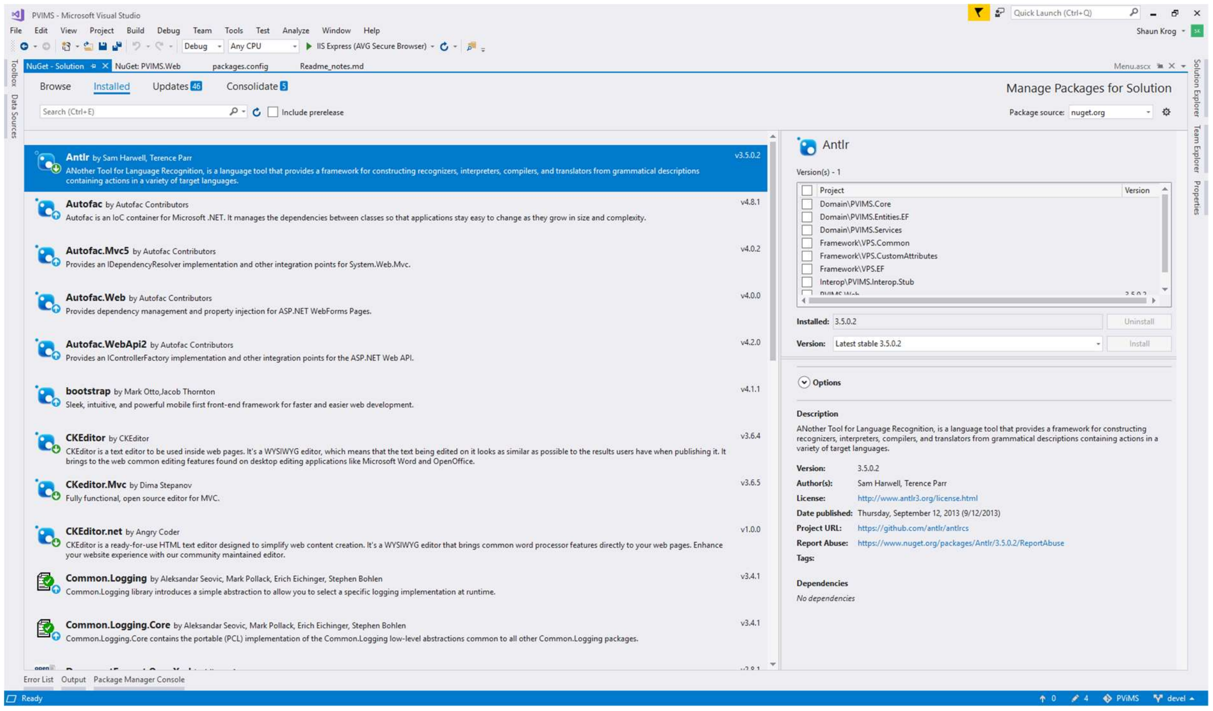
Scheduled For Install:
None
Scheduled For Update:
None
Scheduled For Uninstall:
None

2.3 NuGet Packages

For Dotnet, the Microsoft-supported mechanism for sharing code is NuGet, which defines how packages are created, hosted and consumed and provided the tools for each of these roles.

A NuGet package, in simplified terms, is a single compressed file with a nupkg extension that contains a set of DLLs', additional context related files and a manifest that contains details for the package, including the version number.

The system integrates with a number of open source components as NuGet packages which are available on the online NuGet repository. These packages are owned, maintained and licensed by third parties promote software reuse and increase productivity.



Antlr by Sam Harwell, Terence Parr — v3.5.0.2
ANother Tool for Language Recognition, is a language tool that provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions containing actions in a variety of target languages.

Autofac by Autofac Contributors — v4.8.1
Autofac is an IoC container for Microsoft .NET. It manages the dependencies between classes so that applications stay easy to change as they grow in size and complexity.

Autofac.Mvc5 by Autofac Contributors — v4.0.2
Provides an IDependencyResolver implementation and other integration points for System.Web.Mvc.

Autofac.Web by Autofac Contributors — v4.0.0
Provides dependency management and property injection for ASP.NET WebForms Pages.

Autofac.WebApi2 by Autofac Contributors — v4.2.0
Provides an IControllerFactory implementation and other integration points for the ASP.NET Web API.

bootstrap by Mark Otto,Jacob Thornton — v4.1.1
Sleek, intuitive, and powerful mobile first front-end framework for faster and easier web development.

CKEditor by CKEditor — v3.6.4
CKEditor is a text editor to be used inside web pages. It's a WYSIWYG editor, which means that the text being edited on it looks as similar as possible to the results users have when publishing it. It brings to the web common editing features found on desktop editing applications like Microsoft Word and OpenOffice.

CKeditor.Mvc by Dima Stepanov — v3.6.5
Fully functional, open source editor for MVC.

CKEditor.net by Angry Coder — v1.0.0
CKEditor is a ready-for-use HTML text editor designed to simplify web content creation. It's a WYSIWYG editor that brings common word processor features directly to your web pages. Enhance your website experience with our community maintained editor.

Common.Logging by Aleksandar Seovic, Mark Pollack, Erich Eichinger, Stephen Bohlen — v3.4.1
Common.Logging library introduces a simple abstraction to allow you to select a specific logging implementation at runtime.

Common.Logging.Core by Aleksandar Seovic, Mark Pollack, Erich Eichinger, Stephen Bohlen — v3.4.1
Common.Logging.Core contains the portable (PCL) implementation of the Common.Logging low-level abstractions common to all other Common.Logging packages.

Antlr

Version(s) - 1

Project	Version
Domain\PVIMS.Core	
Domain\PVIMS.Entities.EF	
Domain\PVIMS.Services	
Framework\VPS.Common	
Framework\VPS.CustomAttributes	
Framework\VPS.EF	
Interop\PVIMS.Interop.Stub	

Installed: 3.5.0.2

Version: Latest stable 3.5.0.2

Description

ANother Tool for Language Recognition, is a language tool that provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions containing actions in a variety of target languages.

Version: 3.5.0.2
Author(s): Sam Harwell, Terence Parr
License: http://www.antlr3.org/license.html
Date published: Thursday, September 12, 2013 (9/12/2013)
Project URL: https://github.com/antlr/antlrcs
Report Abuse: https://www.nuget.org/packages/Antlr/3.5.0.2/ReportAbuse
Tags:

Dependencies

No dependencies

A full set of NuGet packages used within PViMS can be described as follows:

Package	Details	Ver.
Autofac.Extensions. DependencyInjection	Autofac implementation of the interfaces in Microsoft.Extensions.DependencyInjection.Abstractions, the .NET Framework dependency injection abstraction.	6.0.0
AutoMapper.Extensions. Microsoft.DependencyInjection	Object-to-object mapping library that can be used to map objects belonging to dissimilar types	7.0.0
Common.Logging	Common.Logging library introduces a simple abstraction to allow you to select a specific logging implementation at runtime.	3.4.1
Common.Logging.Core	Common.Logging.Core contains the portable (PCL) implementation of the Common.Logging low-level abstractions common to all other Common.Logging packages.	3.4.1
DocumentFormat.OpenXml	The Open XML SDK provides tools for working with Office Word, Excel, and PowerPoint documents	2.8.1
DotNetZip	DotNetZip is a FAST, FREE class library and toolset for manipulating zip file	1.11.0
EntityFramework	Entity Framework is Microsoft's recommended data access technology for new applications.	6.4.0
EPPlus	Create advanced Excel spreadsheets using .NET	4.0.5
LINQKit.Core	LinqKit.Core contains extensions for LINQ to SQL and Entity Framework	1.1.17
Microsoft.AspNet.Identity.Core	Core interfaces for ASP.NET Identity.	2.2.1
Microsoft.AspNet.Identity. EntityFramework	ASP.NET Identity providers that use Entity Framework.	2.2.1
Microsoft.AspNet.Identity.Owin	Owin implementation for ASP.NET Identity	2.2.1

Microsoft.AspNet.Mvc	This package contains the runtime assemblies for ASP.NET MVC. ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and that gives you full control over markup	5.2.7
Microsoft.AspNet.Razor	This package contains the runtime assemblies for ASP.NET Web Pages. ASP.NET Web Pages and the new Razor syntax provide a fast, terse, clean and lightweight way to combine server code with HTML to create dynamic web content.	3.2.7
Microsoft.AspNet.WebPages	This package contains core runtime assemblies shared between ASP.NET MVC and ASP.NET Web Pages.	3.2.7
Microsoft.AspNetCore	Core packages	2.2.0
Microsoft.AspNetCore. Authentication.JwtBearer	ASP.NET Core middleware that enables an application to receive an OpenID Connect bearer token.	2.2.0
Microsoft.AspNetCore. Hosting.Abstractions	ASP.NET Core hosting and startup abstractions for web applications.	2.2.0
Microsoft.AspNetCore.Http	ASP.NET Core default HTTP feature implementations.	2.2.2
Microsoft.AspNetCore.Http. Abstractions	ASP.NET Core HTTP object model for HTTP requests and responses and also common extension methods for registering middleware in an IApplicationBuilder.	2.2.0
Microsoft.AspNetCore.Http. Features	ASP.NET Core HTTP feature interface definitions	2.2.0
Microsoft.AspNetCore.HttpsPolicy	ASP.NET Core basic middleware for supporting HTTPS Redirection and HTTP Strict-Transport-Security.	2.2.0
Microsoft.AspNetCore.Mvc	ASP.NET Core MVC is a web framework that	2.2.0

	gives you a powerful, patterns-based way to build dynamic websites and web APIs. ASP.NET Core MVC enables a clean separation of concerns and gives you full control over markup.	
Microsoft.AspNetCore.Mvc. Formatters.Xml	ASP.NET Core MVC formatters for XML input and output using DataContractSerializer and XmlSerializer	2.2.0
Microsoft.AspNetCore. SpaServices.Extensions	Helpers for building single-page applications on ASP.NET MVC Core.	2.2.0
Microsoft.AspNetCore. WebUtilities	ASP.NET Core utilities, such as for working with forms, multipart messages, and query strings	2.2.0
Microsoft.Extensions. DependencyInjection.Abstractions	Abstractions for dependency injection	2.2.0
Microsoft.Extensions.ObjectPool	A simple object pool implementation.	2.2.0
Microsoft.Extensions.Options	Provides a strongly typed way of specifying and accessing settings using dependency injection.	2.2.0
Microsoft.Extensions.Primitives	Primitives shared by framework extensions	2.2.0
Microsoft.Net.Http.Headers	HTTP header parser implementations	2.2.0
Microsoft.Web.Infrastructure	This package contains the Microsoft.Web.Infrastructure assembly that lets you dynamically register HTTP modules at run time	1.0.0
NewtonSoft.Json	Json.NET is a popular high-performance JSON framework for .NET	9.0.1
Swashbuckle.AspNetCore	Swagger tools for documenting APIs built on ASP.NET Core	5.0.0
Swashbuckle.AspNetCore.Swagger	Middleware to expose Swagger JSON endpoints from API's built on ASP.NET Core	5.0.0
System.Buffers	Provides resource pooling of any type for	4.5.0

	performance-critical applications that allocate and deallocate objects frequently.	
System.Numerics.Vectors	Provides hardware-accelerated numeric types, suitable for high-performance processing and graphics applications	4.4.0
System.Runtime.CompilerServices.Unsafe	Provides the System.Runtime.CompilerServices.Unsafe class, which provides generic, low-level functionality for manipulating pointers	4.5.1
System.Text.Encodings.Web	Provides types for encoding and escaping strings for use in JavaScript, HyperText Markup Language (HTML), and uniform resource locators (URL).	4.5.0

Packages can be installed through NuGet by using the package console manager within Visual Studio using the following syntax:

```
Install-Package Newtonsoft.Json -Version 12.0.3
```

3 Deployment

This section will give an overview of the process needed to prepare a PViMS deployment package. The deployment package prepared can be used for deployment as per the deployment manual.

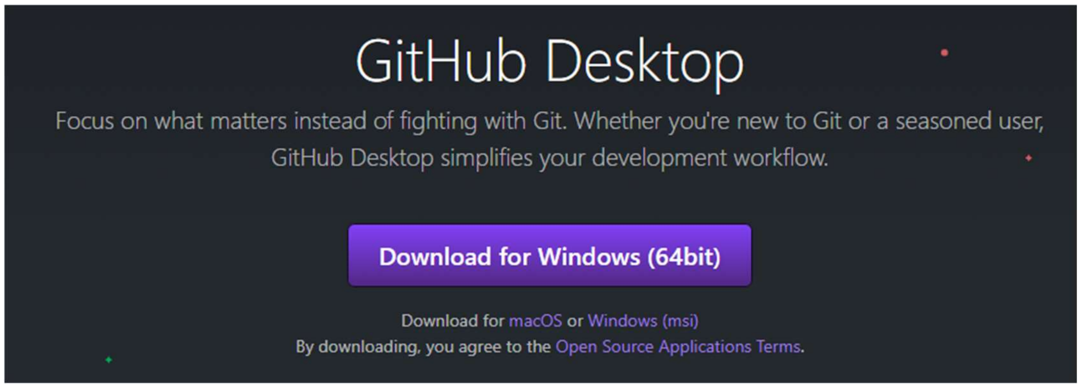
3.1 Preparing your environment for deployment

Step 1. Install GITHUB Desktop Application

- ❖ Browse to

https://desktop.github.com/

- ❖ Select the Download for Windows (64Bit) file



GitHub Desktop

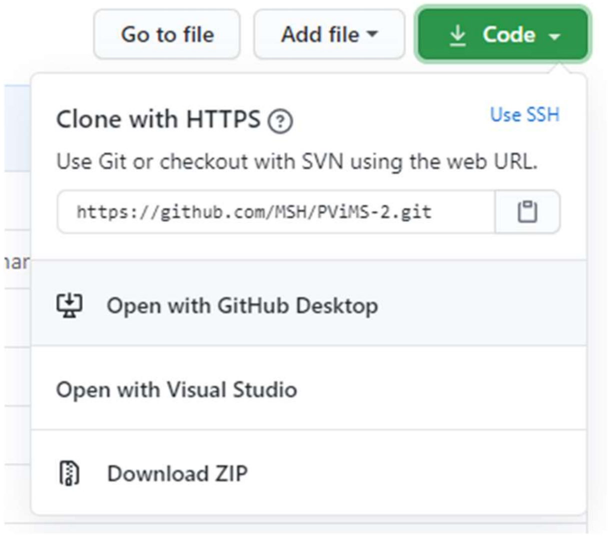
Focus on what matters instead of fighting with Git. Whether you're new to Git or a seasoned user, GitHub Desktop simplifies your development workflow.

Download for macOS or Windows (msi)
By downloading, you agree to the Open Source Applications Terms.

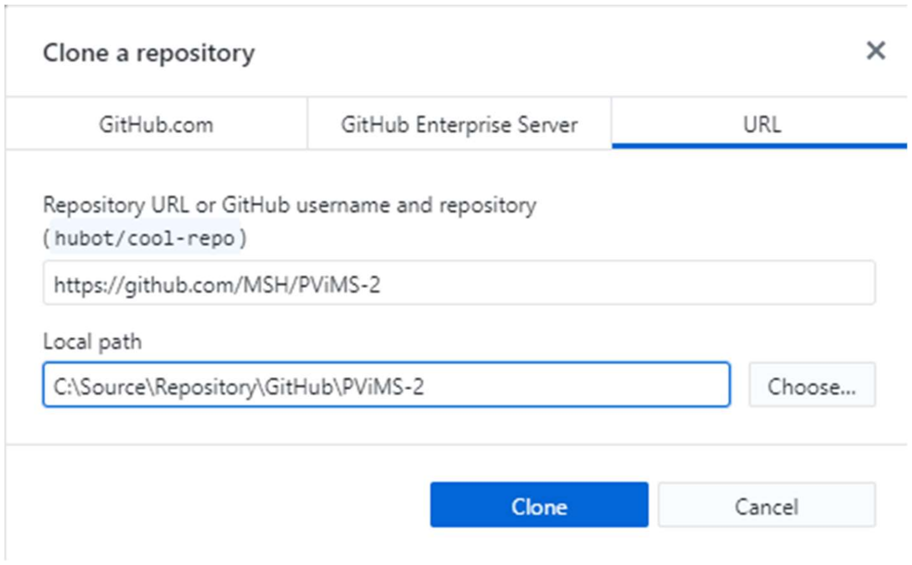
- ❖ Complete the installation of the GITHUB desktop application

Step 2. Clone the GITHUB Repository

- ❖ Browse to the GITHUB repository on https://github.com/MSH/PViMS-2
- ❖ Select the code → Open with GITHUB Desktop menu option



- ❖ Specify the location of the local path where you would like to clone the repository to and click the clone button to clone the source code



Repository URL or GitHub username and repository: https://github.com/MSH/PViMS-2

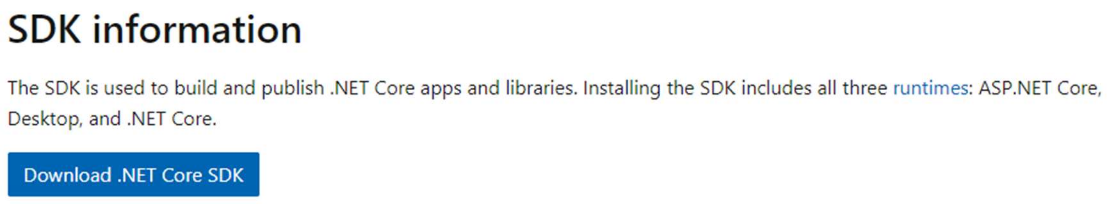
Local path: C:\Source\Repository\GitHub\PViMS-2

Step 3. Install .Net Core SDK

- Browse to

https://docs.microsoft.com/en-us/dotnet/core/install/windows?tabs=netcore31

- Locate the section for SDK Information
- Select the Download .Net Core SDK option


SDK information

The SDK is used to build and publish .NET Core apps and libraries. Installing the SDK includes all three runtimes: ASP.NET Core, Desktop, and .NET Core.

- Complete the .Net Core SDK installation

Step 4. Install Node.js

Please note that the PViMS user interface is developed and tested using node.js version 10.16.

- Browse to

https://nodejs.org/en/download/releases/

- Download and install node.js version 10.16

Step 5. Install angular CLI

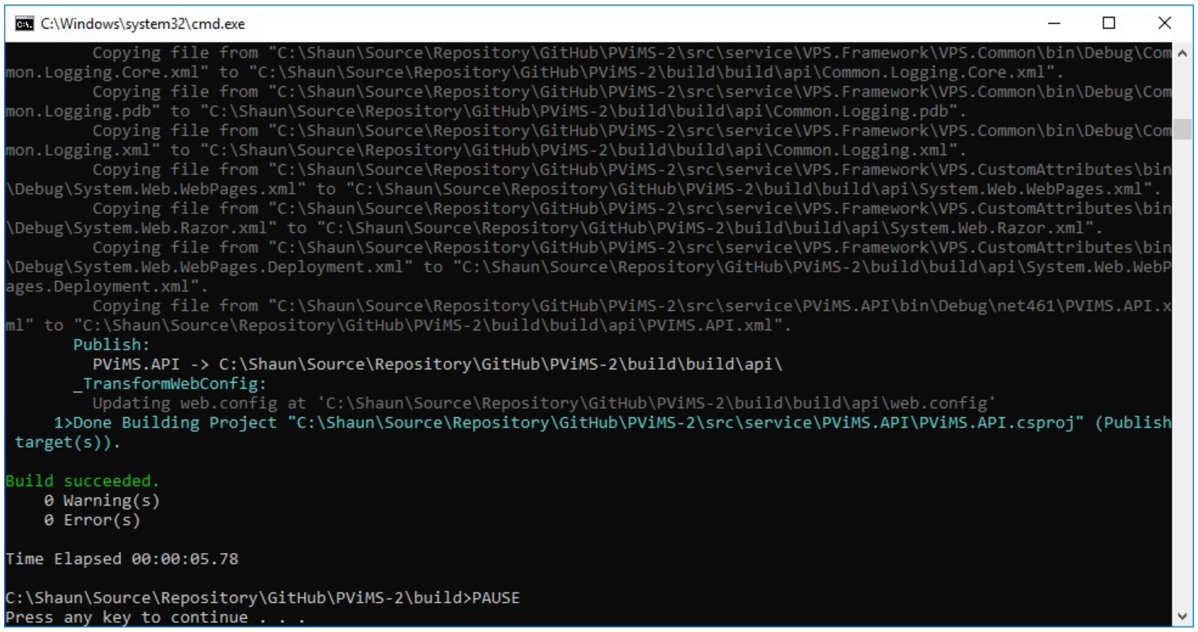
- To install the Angular CLI, open a terminal window and run the following command:

npm install -g @angular/cli

3.2 Preparing a package for deployment

Step 1. Build API Package

- ❖ Run the file build_api.cmd command file which can be located in the build folder



C:\Windows\system32\cmd.exe

```
         Copying file from "C:\Shaun\Source\Repository\GitHub\PViMS-2\src\service\VPS.Framework\VPS.Common\bin\Debug\Com
mon.Logging.Core.xml" to "C:\Shaun\Source\Repository\GitHub\PViMS-2\build\build\api\Common.Logging.Core.xml".
         Copying file from "C:\Shaun\Source\Repository\GitHub\PViMS-2\src\service\VPS.Framework\VPS.Common\bin\Debug\Com
mon.Logging.pdb" to "C:\Shaun\Source\Repository\GitHub\PViMS-2\build\build\api\Common.Logging.pdb".
         Copying file from "C:\Shaun\Source\Repository\GitHub\PViMS-2\src\service\VPS.Framework\VPS.Common\bin\Debug\Com
mon.Logging.xml" to "C:\Shaun\Source\Repository\GitHub\PViMS-2\build\build\api\Common.Logging.xml".
         Copying file from "C:\Shaun\Source\Repository\GitHub\PViMS-2\src\service\VPS.Framework\VPS.CustomAttributes\bin
\Debug\System.Web.WebPages.xml" to "C:\Shaun\Source\Repository\GitHub\PViMS-2\build\build\api\System.Web.WebPages.xml".
         Copying file from "C:\Shaun\Source\Repository\GitHub\PViMS-2\src\service\VPS.Framework\VPS.CustomAttributes\bin
\Debug\System.Web.Razor.xml" to "C:\Shaun\Source\Repository\GitHub\PViMS-2\build\build\api\System.Web.Razor.xml".
         Copying file from "C:\Shaun\Source\Repository\GitHub\PViMS-2\src\service\VPS.Framework\VPS.CustomAttributes\bin
\Debug\System.Web.WebPages.Deployment.xml" to "C:\Shaun\Source\Repository\GitHub\PViMS-2\build\build\api\System.Web.WebP
ages.Deployment.xml".
         Copying file from "C:\Shaun\Source\Repository\GitHub\PViMS-2\src\service\PViMS.API\bin\Debug\net461\PVIMS.API.x
ml" to "C:\Shaun\Source\Repository\GitHub\PViMS-2\build\build\api\PVIMS.API.xml".
       Publish:
         PViMS.API -> C:\Shaun\Source\Repository\GitHub\PViMS-2\build\build\api\
       _TransformWebConfig:
         Updating web.config at 'C:\Shaun\Source\Repository\GitHub\PViMS-2\build\build\api\web.config'
     1>Done Building Project "C:\Shaun\Source\Repository\GitHub\PViMS-2\src\service\PViMS.API\PViMS.API.csproj" (Publish
 target(s)).

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

Time Elapsed 00:00:05.78

C:\Shaun\Source\Repository\GitHub\PViMS-2\build>PAUSE
Press any key to continue . . .
```

- ❖ Ensure you received 0 Error(s) messages
- ❖ Please note, the build\api folder now contains the updated deployment files for the API

❖ Step 2. Edit Angular environment file

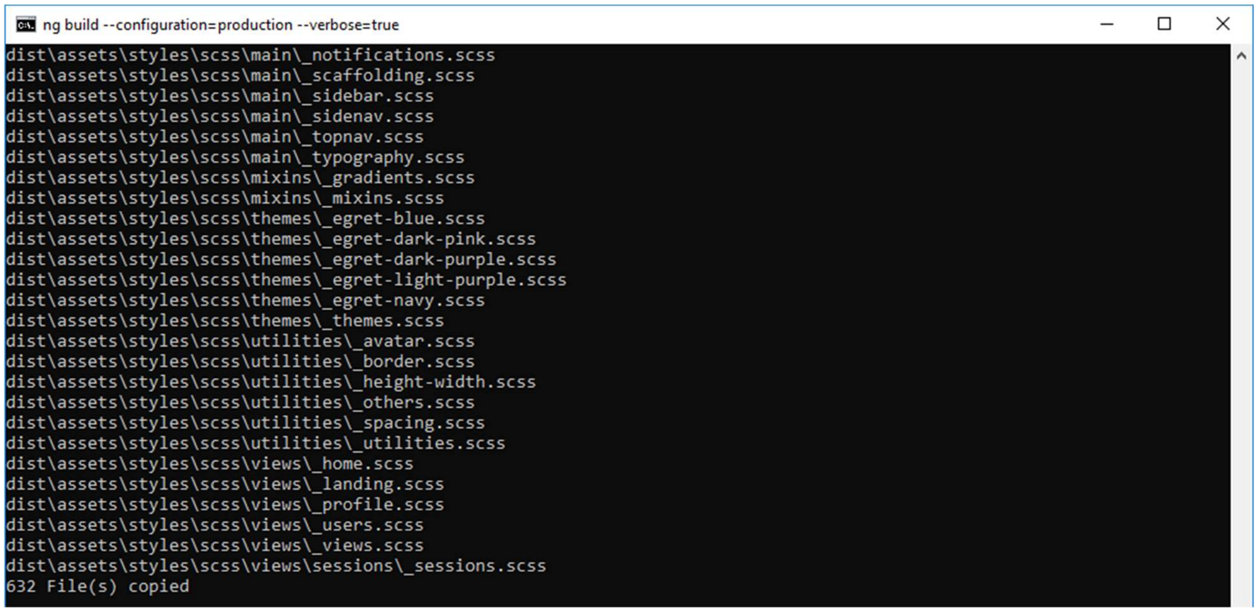
- ❖ Locate the file environment.prod.ts and edit this file with a source code editor such as Visual Studio, VS Code or Notepad ++
- ❖ Edit the apiURL variable to reflect the domain name of the API

e.g. apiURL: 'https://pvimstest-api.msh.org/api'

- ❖ Edit the installationDate variable to reflect the current date
- ❖ Save the file once you have made these modifications

❖ Step 3. Build APP Package

- ❖ Run the file build_app.cmd command file which can be located in the build folder
- ❖ Please note, the build\app folder now contains the updated deployment files for the APP



```
dist\assets\styles\scss\main\_notifications.scss
dist\assets\styles\scss\main\_scaffolding.scss
dist\assets\styles\scss\main\_sidebar.scss
dist\assets\styles\scss\main\_sidenav.scss
dist\assets\styles\scss\main\_topnav.scss
dist\assets\styles\scss\main\_typography.scss
dist\assets\styles\scss\mixins\_gradients.scss
dist\assets\styles\scss\mixins\_mixins.scss
dist\assets\styles\scss\themes\_egret-blue.scss
dist\assets\styles\scss\themes\_egret-dark-pink.scss
dist\assets\styles\scss\themes\_egret-dark-purple.scss
dist\assets\styles\scss\themes\_egret-light-purple.scss
dist\assets\styles\scss\themes\_egret-navy.scss
dist\assets\styles\scss\themes\_themes.scss
dist\assets\styles\scss\utilities\_avatar.scss
dist\assets\styles\scss\utilities\_border.scss
dist\assets\styles\scss\utilities\_height-width.scss
dist\assets\styles\scss\utilities\_others.scss
dist\assets\styles\scss\utilities\_spacing.scss
dist\assets\styles\scss\utilities\_utilities.scss
dist\assets\styles\scss\views\_home.scss
dist\assets\styles\scss\views\_landing.scss
dist\assets\styles\scss\views\_profile.scss
dist\assets\styles\scss\views\_users.scss
dist\assets\styles\scss\views\_views.scss
dist\assets\styles\scss\views\sessions\_sessions.scss
632 File(s) copied
```