# Harris Healthcare Workload Application Developer Guide

COSC 470 Computer Science Department Okanagan College

Version 1.9

Taylor Adam
Nelson Murray
Grayson King
Jackson Bates
Christian Slater
George Cairney
Dakota Logan
Brandon Chesley
Daniel Claggett

April 2020

## **Revision History**

Revision	Date	Brief Summary of Changes	Author
Version 1.4	2020/01/14	Updated Developers guide for Sprint 1	N. Murray
Version 1.5	2020/01/25	Updated Developers guide for Sprint 3	D.
version 1.5			Claggett
Version 1.6	2020/02/11	Updated Developers guide for Sprint 4	D.
			Claggett
Version 1.7 2020/03/10 Updated Dev	Updated Developers guide for Sprint 5	D.	
	2020/03/10	opulated Developers guide for Sprint 3	Claggett
Version 1.8   2020/03/23	Updated Developers guide and	D.	
version 1.6	2020/03/23	Architecture Diagram for Sprint 6	Claggett
Version 1.9	2020/04/08	Updated, Added, and formatted new	D.
version 1.9		diagrams for Sprint 7	Claggett

## Contents

Intr	roduction	3
Des	ign	3
2.1	Architecture Diagrams	3
2.2	Sequence Diagrams	3
2.3	Activity Diagrams	7
2.4	Collaboration Diagrams	8
2.5	Class Diagrams	8
2.6	Database Diagrams	14
2.7	Use Case	15
2.8	Technology Stack	16
Coc	le Description	17
3.1	Azure	17
3.2		17
3.3	SignalR	17
3.4		17
3.5		17
3.6	•	17
3.7	·	17
3.8		18
3.9	FIDO2	18
Tes	ting the Code	18
	9	18
4.2		18
4.3		19
4.4	Integration Testing	19
	Des 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 Coo 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 Tes 4.1 4.2 4.3	2.2 Sequence Diagrams 2.3 Activity Diagrams 2.4 Collaboration Diagrams 2.5 Class Diagrams 2.6 Database Diagrams 2.7 Use Case 2.8 Technology Stack  Code Description 3.1 Azure 3.2 ASAP.NET Core 3.0 3.3 SignalR 3.4 Blazor Server 3.5 MS SQL Server 3.6 Entity Framework Core 3.7 Bootstrap 3.8 Microsoft Identity Framework 3.9 FIDO2  Testing the Code 4.1 Unit Testing 4.2 Code Reviews 4.3 Regression Testing

#### 1 Introduction

This developers guide has been created as a reference to the development practices and standards that were used for the Harris Healthcare Workload application.

This is an evolving document that may be subject to change as the application progresses.

## 2 Design

#### 2.1 Architecture Diagrams

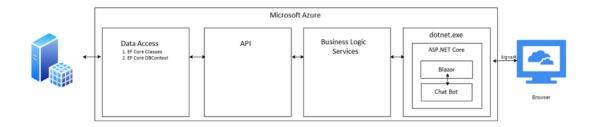


Figure 1: Architecture Diagram

#### 2.2 Sequence Diagrams

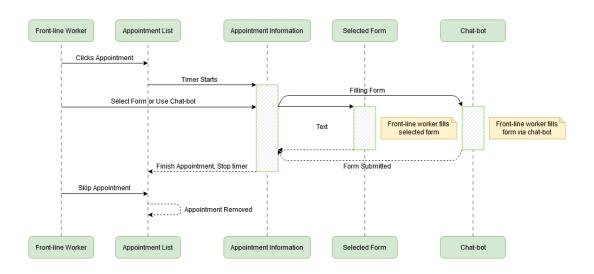


Figure 2: Sequence Diagram for Appointments

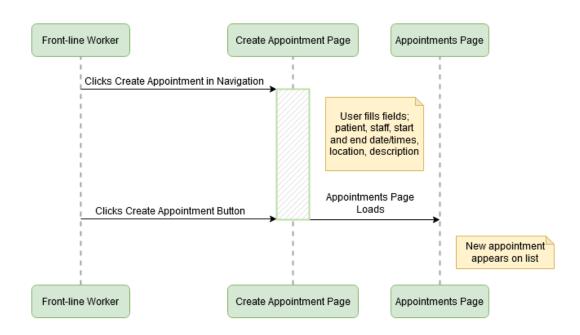


Figure 3: Sequence Diagram for Creating Appointments

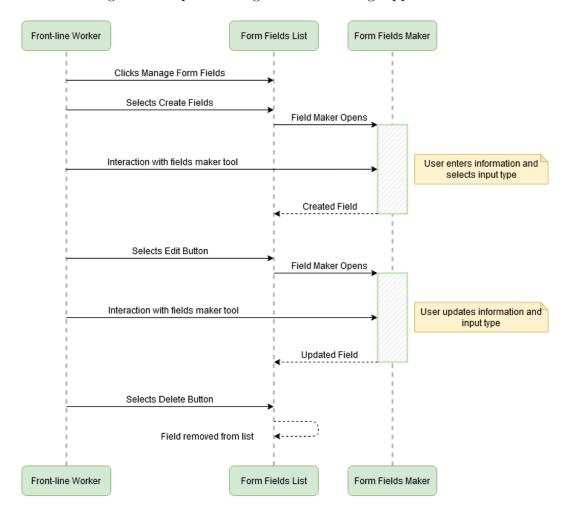


Figure 4: Sequence Diagram for Fields

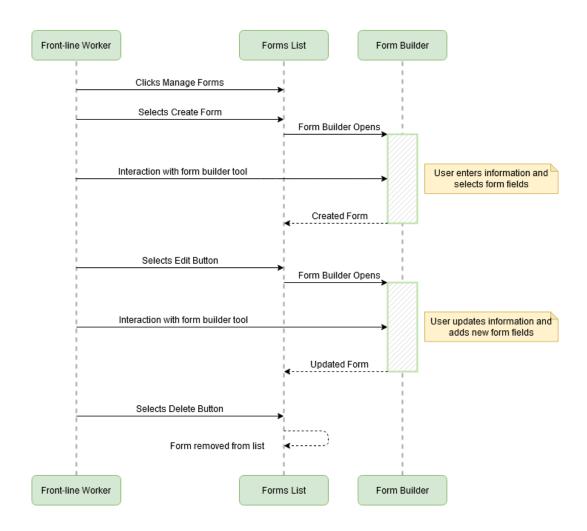


Figure 5: Sequence Diagram for Forms

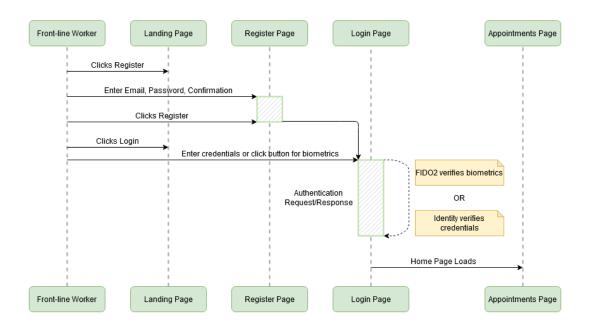


Figure 6: Sequence Diagram for Registration and Log In

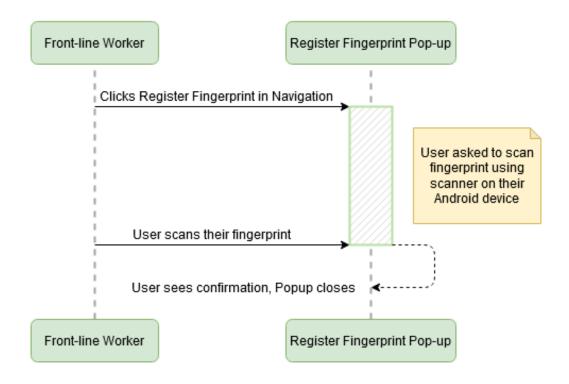


Figure 7: Sequence Diagram for Biometrics

## 2.3 Activity Diagrams

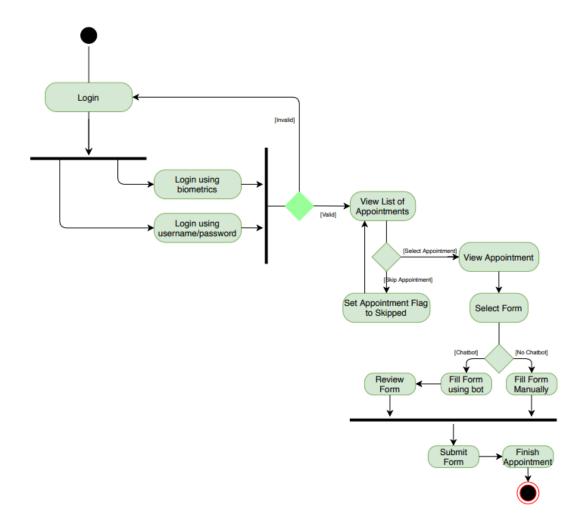


Figure 8: Activity Diagram for Filling a Form

## 2.4 Collaboration Diagrams

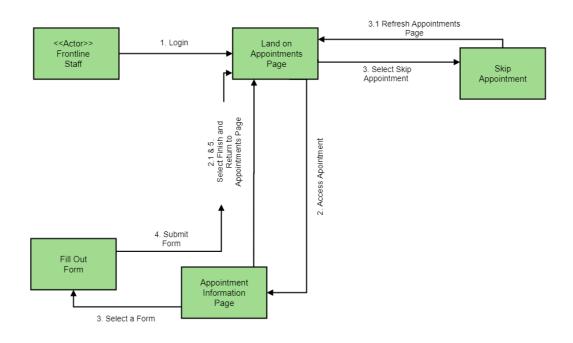


Figure 9: Collaboration Diagram for an Appointment

## 2.5 Class Diagrams

HarrisPharmacy.Data.Entities.Forms namespace				
Form Class	FormField Class	FormWithFields Class	FormSubmission Class	FormFieldSubmission Class
Properties	Properties	Properties	Properties	Properties
FormId	FormFieldId	FormWithFieldsId	FormSubmissionId	FormFieldSubmissionId
Name	FormInputType	FormId	FormName	FormFieldName
Creatorld	FieldName	Form	Userld	FormFieldValue
FormWithFields	UseValues	FormFieldId	FormFieldSubmissions	FormInputType
	Values	FormField		FormSubmission
	FormWithFields			FormSubmissionId

Figure 10: Class Diagram for Forms

# HarrisPharmacy.Data.Entities.Patients namespace

Patient Class	PatientList Class
Properties	Properties
PatientId	PatientListId
FirstName	PatientId
LastName	Userld
Age	StartTime
MentalStatus	EndTime
Sex	Location

Figure 11: Class Diagram for Patients

# HarrisPharmacy.Data.Entities.Appointments namespace

Appointment Class
Properties
AppointmentId
PatientId
Userld
StartTime
EndTime
Location
AppointmentState
DurationInSeconds

Figure 12: Class Diagram for Appointments

#### HarrisPharmacy.App.Models namespace ServerRecipientActivitiesModel ModalModel TherapeuticInterventionModel Class Class Class Properties Properties Properties Message GroupPatientTeaching60Minutes Assessment Title GroupPatientTeaching90Minutes TherapeuticIntervention ConfigModel FormFieldWithValueModel AssessmentModel Class Class Class Properties Properties Properties Token FormField NonFTFAssessment Userld Value FTFAssessment Methods TelephoneElectronicAssessment Equals TelephoneElectronicPlan

Figure 13: Class Diagram for Models

HarrisPharmacy.Data.Services namespace		
AppointmentService Class	FormService Class	PatientInfoService Class
Methods	Methods	Methods
AppointmentService	FormService	PatientInfoService
GetPatientAsync	GetFormsAsync	GetPatientInformationAsync
GetPatientsAsync	GetFormFieldsAsync	
DeleteAsync	GetFormAsync	
SetAppointmentStateFinishedAsync	GetFormFieldAsync	
GetAppointmentAsync	InsertFormFieldAsync	
GetPatientListAsync	InsertFormAsync	
GetPatientListUserAsync	UpdateFormAsync	
GetOpenPatientListUserAsync	UpdateFormAsync	
InsertAsync	DeleteFormAsync	
UpdateAppointmentAsync	DeleteFormFieldAsync	
	UpdateFormFieldAsync	
	GetFormFieldsMultiSelectListAsync	
	CreateFormAsync	
	CreateFormWithFields	
	SubmitFormAsync	
	GetFormSubmissions	
	GetFormSubmission	
	FormExists	

Figure 14: Class Diagram for Services

# HarrisPharmacy.API.Controllers.Home namespace

HomeController Class	DirectLineToken Class	ChatConfig Class
Methods	Properties	Properties
Index	conversationId	Token
	token	Userld
	expires_in	

Figure 15: Class Diagram for API Home

# HarrisPharmacy.API.Controllers.FormsAPI namespace

FormController Class
Fields
_formService
Methods
FormController
GetFormsAsync
GetFormAsync
InsertFormAsync
UpdateFormAsync
DeleteFormAsync

Figure 16: Class Diagram for API Forms

HarrisPharmacy.API.Controllers.FormFieldsAPI namespace

FormFieldController Class
Fields
_formService
Methods
FormFieldController
InsertFormFieldAsync
GetFormFieldsAsync
GetFormFieldsMultiSelectListAsync
GetFormFieldAsync
UpdateFormFieldAsync
DeleteFormFieldAsync

Figure 17: Class Diagram for API Form Fields

## HarrisPharmacy.API.Controllers.AppointmentsAPI namespace

AppointmentsController Class
Fields
_appointmentService
Methods
FormController
DeleteAsync
GetAppointmentAsync
GetPatientListAsync
GetPatientListUserAsync
GetOpenPatientListUserAsync
InsertAsync
UpdateAppointmentAsync
SetAppointmentStateFinishedAsync

Figure 18: Class Diagram for API Appointments

# HarrisPharmacy.API.Controllers.PatientInfoAPI namespace

PatientInfoController Class
Fields
_patientInfoService
Methods
PatientInfoController
GetPatientInformationAsync

Figure 19: Class Diagram for API Patients

#### 2.6 Database Diagrams

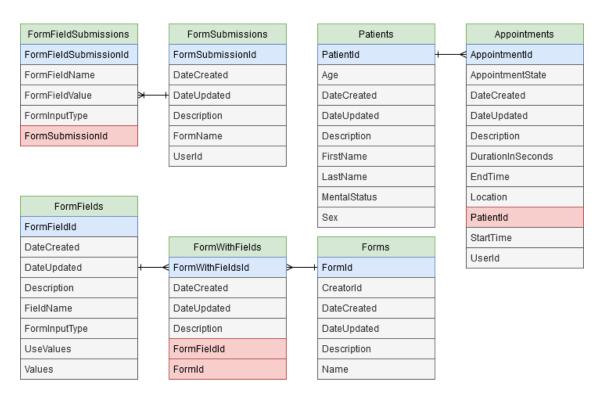


Figure 20: Database Diagram for Application

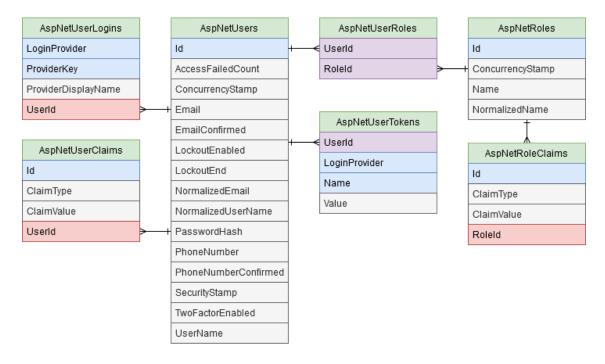


Figure 21: Database Diagram for Authentication

#### 2.7 Use Case

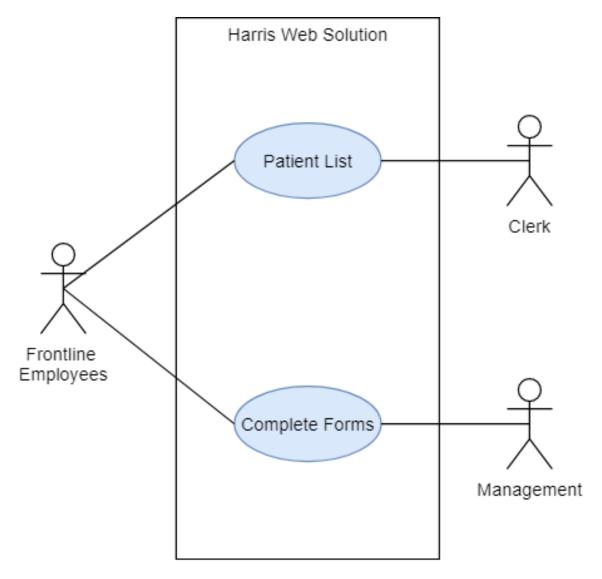


Figure 22: Use Case Diagram

#### 2.8 Technology Stack

- ASP.NET core 3.0
  - The framework used to create the web application, with C.
- Blazor Server
  - Server side web framework used to run ASP.NET at runtime using WebAssembly.
- SignalR
  - SignalR endpoint is created by the Blazor Server that clients connect to use the web application.
- MS SQL Server
  - Used as the database platform.
- Entity Framework core
  - Used by the web application to access data in the MS SQL Server Database.
- Azure
  - Deployment.
- Microsoft Identity Framework
  - Used for the registering and logging in of users.
- Bootstrap
  - Used for front-end user interface.
- FIDO2
  - Used for web authentication with biometrics.

### 3 Code Description

#### 3.1 Azure

Microsoft Azure is a cloud computing service for building, testing, managing, and deploying applications. Azure is utilized in the software project as a way to deploy the project online. This allows the development team, stakeholders, and other interested parties to test and utilize the web application throughout the development process.

#### 3.2 ASAP.NET Core 3.0

ASAP.NET Core 3.0 is an open source web framework developed by Microsoft. ASP.NET Core is a cross-platform tool designed to work across multiple operating systems. It is being utilized in this project as a framework to build a web application with the C language.

#### 3.3 SignalR

SignalR is a software library for Microsoft ASP.NET that allows server-side code to send asynchronous notifications to client-side web applications in real-time.

#### 3.4 Blazor Server

Blazor is an open source web framework designed to run server-side in ASP.NET Core. Blazor allows the web application to utilize the full server capabilities, .NET tooling, and support browsers that don't work with WebAssembly and on resource-constrained devices.

#### 3.5 MS SQL Server

MS SQL Server is a relational database management system developed by Microsoft. In this project it is being utilized as the main database platform.

#### 3.6 Entity Framework Core

Entity Framework Core is an open-source object relational mapping framework. EF Core servers as an object-relational mapper enabling the use of .Net objects and eliminating the need for most data-access code. In this software project, it is being utilized by the web application to access data in the MS SQL database.

#### 3.7 Bootstrap

Bootstrap is a free open-source CSS framework mainly used for responsive mobile-first front-end development. Bootstrap is being used as the front-end user interface for the web application.

#### 3.8 Microsoft Identity Framework

Microsoft Identity Framework is an API that provides user interface login functionality. It manages users, passwords, roles, tokens, emails, and more. It is being used to manage the registration and login of users in this software project.

#### 3.9 FIDO2

FIDO2 is an authentication standard that enables users to leverage common devices to easily authenticate to online services in both mobile and desktop environments. In this project it is being used for its ability to use biometric authentication with web applications and mobile applications so as to eliminate the use of passwords. If FIDO2 is successfully implemented, it may allow for the removal of the other authentication system relying on usernames and passwords.

#### 4 Testing the Code

Regular testing of the code in a software project is required to ensure that there is a standard adhered to throughout the development process. This section describes the testing standards utilized throughout the development process, and why those standards were chosen and their benefits.

There are multiple tests that are repeatedly ran throughout the development process to ensure that the software is working and maintains its integrity. We utilize layered testing, a technique which refers to testing on different levels of the software. Layered testing is testing various levels of the system, from subroutines and modules to large systems.

#### 4.1 Unit Testing

First, we utilize Unit-testing to test individual subroutines and modules. Unittesting is used to validate that each unit of a software is performing properly as designed. Unit tests are important as the help to fix bugs early on in the development cycle.

#### 4.2 Code Reviews

Second, we utilize peer code review. Code reviews performed by other members of the development team allow for errors to be caught before being implemented. The code review process is being implemented in our software development process by pair programming. Pair programming is when a team of two or three members of the development team work on the same subroutines or system. This process allows multiple people to go through the implementation and the design of the code, and possibly catch errors that may be missed in some of the other testing methods.

#### 4.3 Regression Testing

Third, we utilize regression testing to ensure that none of the recent changes to the program negatively impact existing features. Regression testing focuses on ensuring that the old code is still working properly after new code has been implemented.

#### 4.4 Integration Testing

Last, we utilize a method called Integration-testing, which combines individual modules and units, before testing them as a group. This test exists to ensure that the relationships, interfaces, and interactions between the units are performing properly.