Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Management Science and Information Systems Studies

Final Year Project Report

Integration of a Teledentistry System with an Open Practice Management System for Toothpic

CONFIDENTIAL

Dara Ó Cairbre

March 2018

TRINITY COLLEGE DUBLIN
Management Science and Information Systems Studies
Project Report

TOOTHPIC

Integration of a Teledentistry System with an Open Practice Management System

23rd March 2018

Prepared by: Dara Ó Cairbre ***Supervisor: Gaye Stephens***

DECLARATION

I declare that the work described in this dissertation has been carried out in full compliance with the ethical research requirements of the School of Computer Science and Statistics.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at: http://www.tcd.ie/calendar

I have also completed the Online Tutorial on avoiding plagiarism 'Ready, Steady, Write', located at
http://tcd-ie.libguides.com/plagiarism/ready-steady-write

I declare that the report being submitted represents my own work and has not been taken from the work of others save where appropriately referenced in the body of the assignment.

Signed:

_____

Dara Ó Cairbre
23 March 2018

ABSTRACT

The objective of this project was to develop proof of concept software for Toothpic that could facilitate the transfer of patient demographic information between their system and OpenDental, a practice management system. The project involved studying the two systems and developing a means to connect them. The software was written along with extensive documentation; and it was designed to be adaptable and open-ended, to allow further development beyond this project's lifecycle. The final software was adequately tested and met all of the requirements. The deliverables constitute a strong framework to pursue this area further.

PREFACE

Toothpic is a teledentistry service who provide virtual dental check-ups for mobile users over a network of dentists across the United States. They are based in Dublin 2.

The project was successful in meeting all the terms of reference specified. The project was initially quite broad, so one of the challenges was to concentrate its objectives. This was achieved successfully and is evident in the terms of reference.

The software was entirely hypothetical (i.e. a production implementation was off limits), and it was limited in its design by conforming to two already established systems. These two factors also posed challenges. However, the challenges were successfully managed, and the project results were positive.

I would like to extend my gratitude to the following people: Asst. Professor Gaye Stephens for her expert guidance and continued support; Shane Owens and Deirdre van Wolvelaere from Toothpic for all of their support and generosity; Jack Toner and Jack Berrill for their help in setting up the project; and, Asst. Professor Aideen Keaney for all of her assistance.

TOOTHPIC
Practice Management System Integration

23rd March 2018

TABLE OF CONTENTS

APPENDICES

1 INTRODUCTION AND SUMMARY

This chapter introduces the client, outlines the project background and identifies the terms of reference. The approach to the project is summarised, and so too are the remaining chapters.

1.1 The Client

Toothpic is a Dublin based start-up founded in 2012. They operate a teledentistry service on a mobile platform. They have a network of teledentists covering the fifty United States and have completed over 5,000 virtual consultations so far.

Shane Owens, Chief Technology Officer of Toothpic was the liaison for this project.

The client's teledentistry service begins with the user filling out a case submission (a small number of questions and six photographs of their mouth). In less than twenty-four hours, a teledental assessor within the Toothpic network is assigned to review the case and deliver a report to the user. The report outlines any visually-identifiable conditions the assessor has deduced and provides recommendations to the user. In certain circumstances, the assessor may recommend the user visits a physical dentist for further dental care.

For business context: consumers utilise the service as a time saver as the app can rule out unnecessary trips to the dentist. Meanwhile, teledentists (any US-licensed dentist) join the network as assessors to earn additional income, in a similar manner to the many other gig economy-based apps/services. The user and the assessor reviewing their case have no pre-existing relationship (i.e. the teledentist is not necessarily the user's regular dentist), and they do not communicate directly. This entire process takes place on a proprietary system which Toothpic have developed.

This is the extent of the client's service at present.

1.2 Project Background

This project came about as a means to expand the client's service.

Telehealth has been identified by the client as a major growth area. They pioneered this field when they founded their company five years ago. However, with the field growing, so too will the impact of major competitors. Their service is at risk of being flattened by established companies who plan to introduce their own teledentistry services.

In order to survive, the client needs to vertically integrate by expanding their reach beyond mobile case consultations. Expansion into the physical practice is one of the ways they can do this.

The client's initial goal is to partner with dental practices whom they could "refer" their users to. In order to do this, they would need a means to integrate with internal practice management systems (PMS) within the practice.

At present, the client would like to see if they can facilitate an exchange of patient demographic information between their systems and a PMS. Hence, this is the broad scope of this project.

In another sector, this would be a simpler task, however, the following factors were cited as complicating the client's efforts:

- There are a number of different PMSs to integrate with.
- Each of these systems have proprietary APIs, some of which are only accessible at a cost.
- There are many different standards which define the transfer of patient information yet there is little consensus on which standard to use.
- The exchange of patient information is tightly regulated by HIPAA (Health Insurance Portability and Accountability Act) which can increase complexity when building new systems.
- Interoperability between eHealth systems is a major issue in the United States. For context: in 2009 President Obama invested over USD 27 billion towards eHealth interoperability, and still cited it as the greatest disappointment of his healthcare legislation (Kliff, 2017).

HIPAA is a US regulation which governs electronic health information transactions. While this report will not dive into detail regarding HIPAA, the client's systems must conform to its regulations.

It was decided that the project would consist of facilitating an exchange of patient demographic information between the client's system and a selected PMS, which is OpenDental.

1.3 Terms of Reference

The project terms of reference are as follows:

1. Identify Toothpic's requirements for integration and describe their data model.
2. Identify and describe the OpenDental model within the bounds of Toothpic's integration.
3. Describe the domain and context of the OpenDental model.
4. Implement a proof-of-concept integration for data transmission between Toothpic and OpenDental as per requirements and models outlined above.
5. Provide implementation documentation to assist Toothpic with future integration.

1.4 Approach

The approach to this project is slightly different to other systems development projects. The focus was not on implementing production-use software, but rather on building a foundation for future systems to grow. The approach has been carefully formulated, and a number of factors have been taken into account when planning. Firstly, that the client is a small, new company – this has made it difficult for them to invest the resources necessary to undertake a project such as this. Secondly, that the study of interoperability within eHealth (and more specifically, Telehealth) is quite vast and complex. Thus, it has been difficult for the client to build interoperability into their systems. However, interoperability is a goal for them going forward, as part of their strategy to integrate into the practice.

Although the project's end result includes a piece of software that facilitates the client's API (Application Programming Interface) to communicate with OpenDental, it is understood by all parties that this is not expected to be a final solution.

The project is centred around the practicality of developing a proof of concept implementation; however, the approach has focused on how the client can achieve greater interoperability as they grow. This is highlighted in the recommendations at the end, but also in the work completed and the documentation attached. This perspective is furnished by the student's research of interoperability within eHealth.

Thus, this project should serve to provide the client with a springboard to integrate into the practice and advise on how they can build interoperable systems in the future. The included software, source code, and documentation should allow Toothpic to hit the ground running when extending their functionality into the practice.

1.5 Report Summary

- Chapter 2: System Overview provides an overview of the system. The overview is largely from a business perspective and includes a set of requirements. This chapter includes a description of the system's technical environment and a description of the system itself. It also includes an overview of the approach to documentation.
- Chapter 3: Description of Work Done details all of the work completed during the project lifecycle. It opens with the Software Development Method (SDM) and summarises each of the development phases: requirements, design, development, and testing. It also includes an overview of the difficulties encountered.
- Chapter 4: Recommendations gives a brief conclusion and outlines recommendations which the client could implement to grow their integration into the practice.

2 SYSTEM OVERVIEW

This chapter describes the system in business terms. It begins with a business overview of the system, followed by the business requirements. A system description and a technical overview are presented. Finally, the approach to system documentation is outlined.

2.1 Business Overview

The motivation behind this system is presented in the project background (Section 1.2, p. 1). The primary objectives of this system are to enable the exchange of patient demographic information in two situations:

a. information flowing from Toothpic to the dental practice,
b. information flowing from the dental practice to Toothpic.

The system developed is a plugin for the PMS OpenDental. The plugin interfaces between the client's API and OpenDental's plugin environment. It can theoretically be used in any dental practice that uses OpenDental for their PMS, although it has not been implemented in a production environment.

2.2 Business Requirements

Broadly speaking, the business requirements are presented here:

- To import/export patient demographic information that is held within the client's database to/from OpenDental's database.
- To interface with the client's existing API.
- To interface with OpenDental's Practice Management Software.
- To facilitate straightforward deployment of the OpenDental client (i.e. a plugin that can be easily installed and set up in OpenDental).
- To facilitate usage of the OpenDental client by non-technically advanced users (i.e. the design should be simple and self-explanatory).
- To securely connect and communicate with the client's API over the Internet (i.e. communication only through their HTTPS endpoint).
- To follow the client's authentication protocol for the API.

2.3 System Description

The system is self-contained within a plugin developed for the OpenDental software. Once deployed and installed in OpenDental, the practice can begin to communicate with the client's API. Instructions on how to set up the plugin in OpenDental are included in the developer manual (Appendix C, p. C-1).

The main user of the system is referred to as the "dental practice" however there can be any number of users within the dental practice who utilise the system. Each user will have their

own identity and can authenticate with the client's API through the plugin. A system overview diagram is provided in Figure 2.3.1.

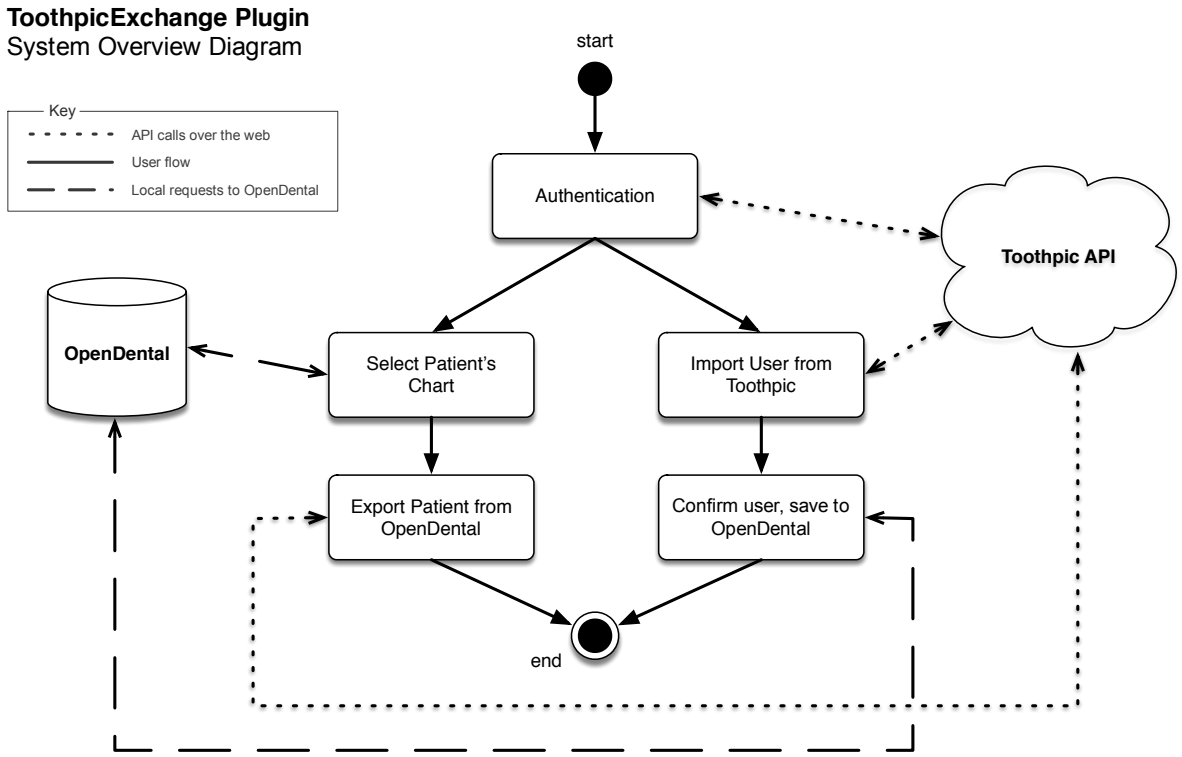


Figure 2.3.1 – System Overview Diagram

Authentication

In order to access the client's API, and in order to deliver the correct data to an authorised viewer, the practice must authenticate themselves through the plugin upon launching it. Their credentials (username and password) are supplied by the client. The process follows the client's authentication protocol and is based on a system of tokens granted by the API. The software has been developed so that authentication plays a key role in the software backend, even though it is only visible on the frontend through the login screen.

For the end user, they merely enter their details upon launch and press login. This can be seen in Figure 2.3.2.

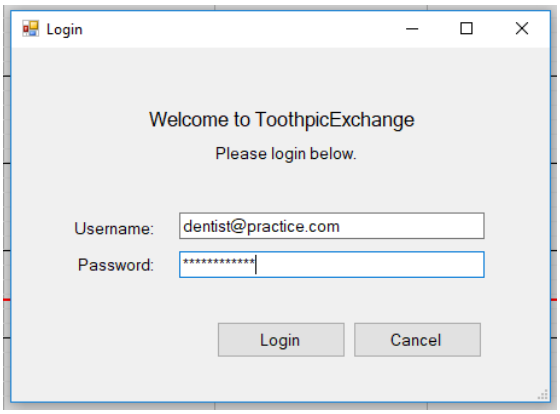


Figure 2.3.2 – Login Screen

Once the practice has authenticated themselves (i.e. logged in), they are provided with two main functions. These can be seen in Figure 2.3.3.

1. Import Patient Info
2. Export Patient Info

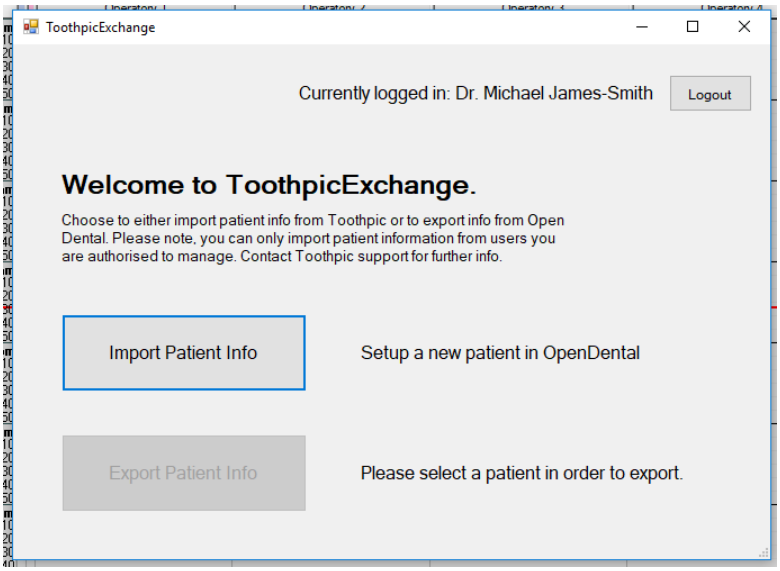


Figure 2.3.3 – The plugin “homepage”

Importing Patient Info

The practice is presented with a list of Toothpic users which they are authorised to view. The plugin makes a call to the client’s API using the authentication tokens, and the API returns the list of users. The list does not contain all users, it only contains users who are in the process of being handed over from Toothpic to the practice. This means the list, in general, is often relatively short.

There is a search box at the top of the list which the practice can use to filter the results by name or email. This is visible in Figure 2.3.4.

The practice clicks on the user they’d like to import. They are then presented with an option to add additional information about the user. The plugin will always check for duplicate users before importing.

Exporting Patient Info

The practice “selects” a patient as they would normally in OpenDental (i.e. by opening their chart). The option to export is disabled until a patient has been selected. When the Export Patient button is clicked, a request is sent to Toothpic’s API which verifies the user does not currently exist in Toothpic’s system. Providing the user does not already exist, the patient is successfully imported into Toothpic’s system. This functionality is visible in Figure 2.3.5.

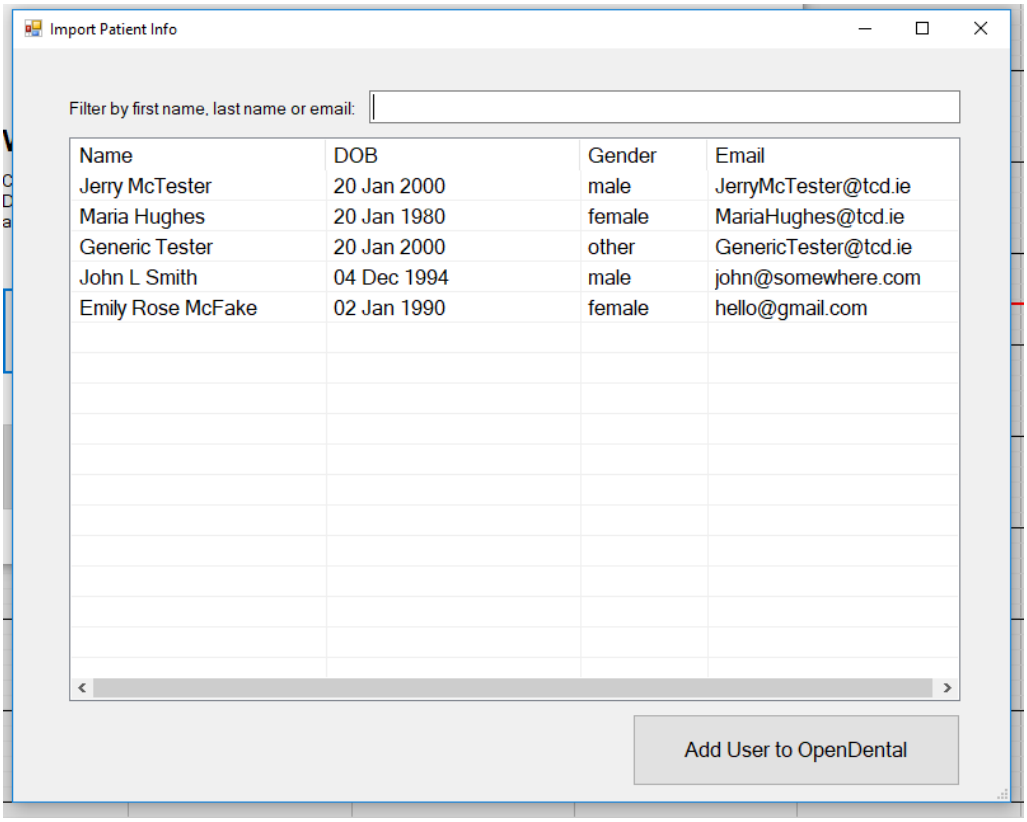


Figure 2.3.4 – Import Screen

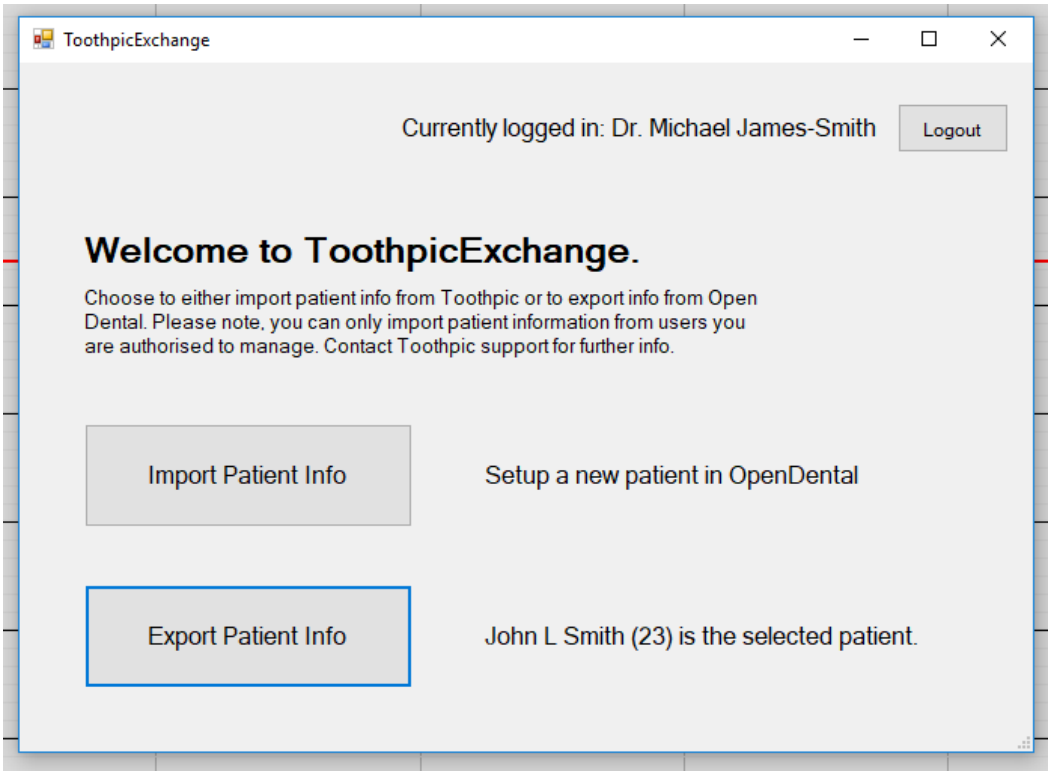


Figure 2.3.5 – Homepage when a user is ready to export

2.4 Technical Environment

The plugin interfaces with both OpenDental and the client's API, hence the technical environment is largely dictated by these two factors.

OpenDental is a free Practice Management System used in dental practices across the United States. Their source code is written in C# .NET and runs on Windows. The software first launched in 2003. Although technically not "Open Source", their source code is free to download and modify under the GNU General Public License (Free Software Foundation, 2007).

The client operates a RESTful (**Re**presentational **S**tate **T**ransfer) API over HTTPS (HyperText Transfer Protocol, Secure) with authentication using the JSON Web Token (JWT) standard. The plugin operates by sending requests to the client's API over the Internet and parsing the response. The response is mapped to OpenDental's data interface. The subsection "API Integration" (Section 3.3, p. 17) contains a more detailed explanation of the client's API.

The plugin is written in C# .NET with Windows Forms just like OpenDental's source code. A Visual Studio environment was used to develop, test and debug the plugin with OpenDental.

OpenDental utilises MySQL for its database, however, the plugin does not interact directly with the database. The plugin interfaces through a separate framework developed by OpenDental.

A traditional OpenDental setup in the practice would consist of a local server, running OpenDental and MySQL. There would then be any number of OpenDental clients networked to talk back with the server. The plugin developed is intended to be installed on any OpenDental client connected to the Internet and can thus be used by any staff member within the practice. It is not required to be installed on the server.

The steps to set up OpenDental and to install the plugin are detailed in the developer manual. A list of dependencies is also included in the developer manual (Appendix C, p. C-1).

The project was built on C# 7.2, .NET 4.5 and OpenDental 17.2.

2.5 System Documentation

For this project, documentation was considered equally as important as the software itself. Because the software was not implemented into a production environment, it was generally understood that it would be adapted and modified as the client expands their service. Hence, the plugin source code has been heavily documented using Visual Studio's C# XML documentation language (Microsoft, 2015). XML tags are used in every class, method, and property to accurately explain their role and function.

The end result is not a static developer manual or an API table, but rather it is a living document. The documentation is compiled automatically with each build of the software (using

the XML tags) and is packaged into a comprehensive HTML website which is simple to navigate. This means that not only will it be simple to understand the code that has already been written to a third party, but as the software grows, so too can the documentation.

The documentation was compiled by DocFx.console within Visual Studio. Further to the API documentation, the manual also contains static content such as instructions on how to set up OpenDental and the plugin. Details on how to configure and update the documentation are included in the developer manual (Appendix C, p. C-1).

The HTML developer manual and a PDF version have been included on the attached DVD. The documentation is best experienced in a web browser or PDF viewer. However, it has also been reproduced in Appendix C (p. C-1).

3 DESCRIPTION OF WORK DONE

This chapter describes the work completed over the project lifecycle. A version of the incremental model was chosen for the system development methodology. The reason for this choice was because development was a single person task with recurring feedback from the client; the model needed to be a fast and dynamic process. This model proved to be successful, and there was a functioning version of the system within the first increment.

The incremental model started with building a rudimentary version of the system. Once this was completed, a new increment was developed which added functionality to the system. This process was repeated until the final system was developed. Each increment contained four phases: requirements, design, development, and testing. The “implementation” phase of the model was omitted as it was not possible to implement a production system.

The plugin was developed over four increments. However, for simplicity the sections that follow summarise each of the four *phases*. Further detail on the entire process is included in Appendix D (p. D-1).

3.1 Requirements

The requirements phase of this project began with eliciting the business motivations for such a system. The original goals were quite broad, and the initial project outline (Appendix A, p. A-1) was ambiguous. A meeting was held with Shane Owens on 23 November 2017 to elicit these motivations. Following this meeting, investigation was undertaken to propose how a system such as this could actually be implemented.

The investigation focused on Practice Management Systems and Electronic Health Record standardisation and interoperability. The relevant summaries have been included in Appendix H (p. H-1).

A rudimentary proposal of the system was discussed at a meeting held with Shane Owens, Jack Berrill (Chief Operating Officer), Jack Toner (Product Director) and Project Supervisor Gaye Stephens on 13 December 2017. From this meeting, the terms of reference (Section 1.3, p. 2) were identified. It was also established that OpenDental would be the software of choice to interface with because their source code is open and freely accessible. The interim report (Appendix B, p. B-1) summarises events up to this point.

At this stage, there was still uncertainty as to how a system such as this would operate, and what the requirements would be. In order to clarify, scenarios were drawn up to broadly envisage how and why the system would be used. These are included in the project’s design documentation (Appendix D, p. D-2).

Scenarios assisted the client in understanding the business logic behind the system. From these scenarios, it was deduced that the system would largely consist of a piece of software operated within a dental practice that would talk to Toothpic’s API. This allowed requirements

to be made more specific, as they could now be formed around an end-user (the dental practice).

Business requirements were drawn up from this, and use-cases followed. Once the high-level requirements were settled on, a full list of system requirements was drawn up. These are all included in the project's design documentation (Appendix D, p. D-2). All of this documentation facilitated a common level of understanding about the scope and the context of the system under discussion.

Broadly speaking, the highest-level requirement was to develop a plugin that mapped users in Toothpic's system to patients in OpenDental's system and allowed the exchange of their information. The plugin was to be used on any OpenDental client within the dental practice by dentists, administrators, and other practice staff.

3.2 Design

The design process entailed designing a system that would meet all of the system requirements. A by-product was that it also facilitated an understanding of the domain and context of both OpenDental and Toothpic's systems. There were three overarching phases to the design process:

- Initial investigation
- Data modelling
- System functionality

Initial Investigation:

The first step was to conduct investigation into how the requirements could be met. Bearing in mind the project background, consideration was taken into how the system developed could be interoperable, and easily expanded upon in the future. Three key areas of investigation were designated:

- OpenDental's software
- Toothpic's API
- Patient information transfer standards

OpenDental:

In terms of OpenDental, their source code (OpenDental, 2017) was downloaded and investigated. OpenDental have a very small amount of documentation, so their source code was physically examined alongside examples of other plugins and middleware. This took a considerable amount of time considering it is an extensive package.

OpenDental presented three potential ways to interface with Toothpic's API. Each of these was investigated:

- Via a custom plugin based on their plugin framework
- Via HL7 v2 messaging
- Via HL7 FHIR

OpenDental have a robust plugin framework, and many other plugins have been developed. Plugins can directly manipulate the OpenDental source code or they can extend functionality.

HL7 (Health Level 7) is a standards-developing organisation. HL7 v2 (version 2) and HL7 FHIR (Fast Healthcare Interoperability Resources; pronounced “fire”) are two standards for communicating patient information. A summary is presented in the “standards” section below.

Toothpic:

Toothpic’s API was investigated in a similar manner to OpenDental’s. Toothpic have no API documentation, and no access could be authorized to the backend (due to Toothpic’s HIPAA compliance) so a proxy server was set up to manually examine the API calls to/from the Toothpic mobile app. Once all of the API calls were enumerated, a data model was drawn up for which a local test API server was configured to emulate.

Toothpic’s authentication protocol was also investigated and it was discussed with Shane Owens at a meeting on 8 February 2018.

Standards:

In light of the above investigations, research was then conducted into the variety of standards that concern communication of patient information between systems. While many standards exist, the two investigated in detail were standards that OpenDental conform to – HL7 v2, and HL7 FHIR.

HL7 version 2 was initially created in 1989. It is a messaging format – an overview of HL7 v2 is presented in Appendix H (p. H-1). In summary, it is an outdated format that is still utilised in many healthcare systems. OpenDental does have HL7 compatibility, so a system could theoretically have been developed around it. However, the HL7 module in OpenDental is neither user-friendly to set up nor to use, so it would not have adhered to the ease-of-use business requirement.

HL7 FHIR is a modern, RESTful standard built around the concept of interoperability. See Appendix H (p. H-4) for an overview of FHIR; see subsection “API Integration” in Section 3.3 (p. 17) for a detailed explanation of a RESTful model.

FHIR would be a perfect match for Toothpic’s RESTful API. Unfortunately, OpenDental does not currently support the entire FHIR framework. As of the time of writing, OpenDental’s FHIR functionality only facilitates appointment-based information.

By process of elimination, it was decided a custom plugin would work best for this project. Although it would have been desirable to use either of the HL7 models from an interoperability point of view; their lack of support in OpenDental meant the full system requirements would not have been met.

Data Modelling

With the initial investigation completed and a decision that a custom plugin was to be developed in OpenDental, the next step was to model the data structures.

The Toothpic user data model is detailed in Table 3.2.1. It maps perfectly to the OpenDental model with two exceptions:

- OpenDental only record a middle initial (instead of a full middle name).
- OpenDental record a Social Security Number (SSN) whereas Toothpic do not.

The system was designed to handle both of these exceptions. The issue of the SSN was raised at a meeting with Shane Owens on 8 February 2018. Although it is possible to uniquely identify a user by their names, date of birth, zip code and gender; it is advisable in a clinical system to utilise a unique identifier such as an SSN for data transfer. This ensures patient information does not get mismatched. The client does in certain cases record the user's insurance identifier, although OpenDental do not use this identifier so it was unusable. Nonetheless, the plugin was designed to check name and date of birth conflicts before information exchange.

Table 3.2.1 – The Toothpic User Data Model

Attributes
First Name
Middle Name
Last Name
Gender
Date of Birth
Zip Code
<Proprietary ID numbers>

Class diagrams were drawn up to design how the Toothpic user model would be represented in the plugin, and the mapping between a Toothpic user and an OpenDental patient was outlined. Figure 3.2.1 shows a simplified view of ToothpicUser and OpenDental's Patient. Both inherit methods from System.Object (the C# Object class) which means their values can be compared. This diagram follows the Unified Modelling Language (UML). The class name is written in bold, then its attributes (properties) followed by its methods (functions).

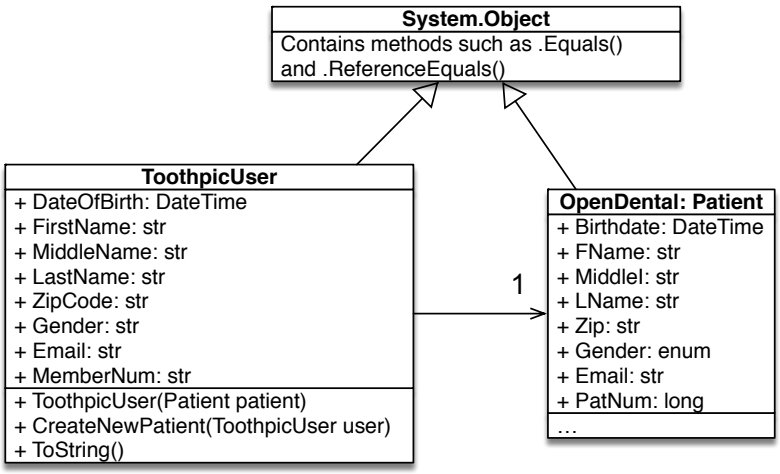


Figure 3.2.1 – A simplified Class Diagram

System Functionality

With the data models and the technical environment specified, the next step was designing the system's functionality and activity flows. Two activity diagrams were developed to design both the import and the export functionality. These diagrams also follow UML and are presented in Figure 3.2.2 and Figure 3.2.3.

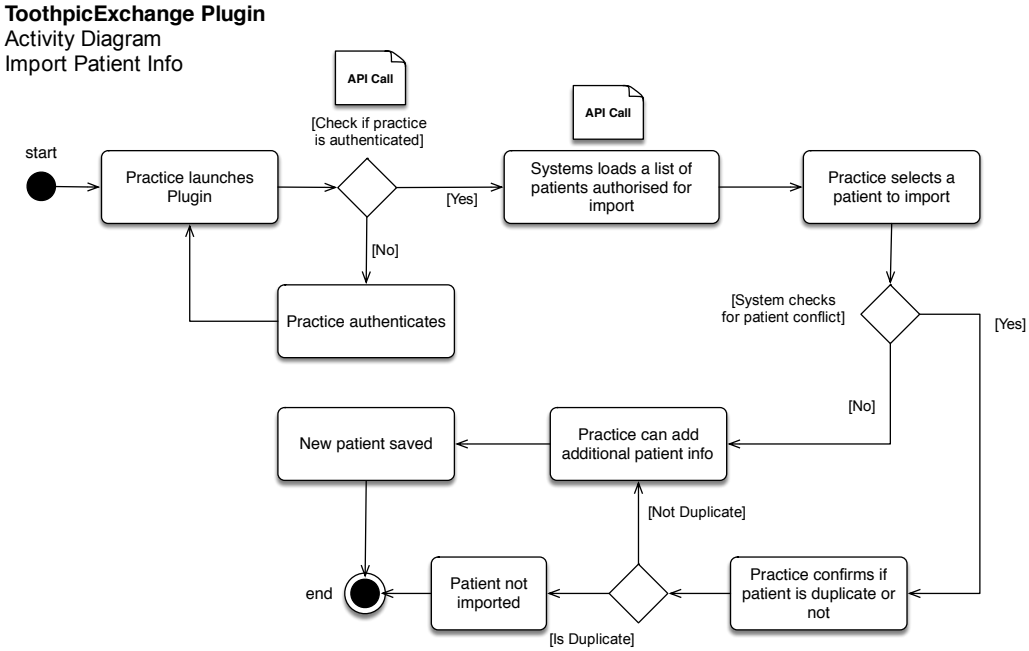


Figure 3.2.2 – The Import Function Activity Diagram

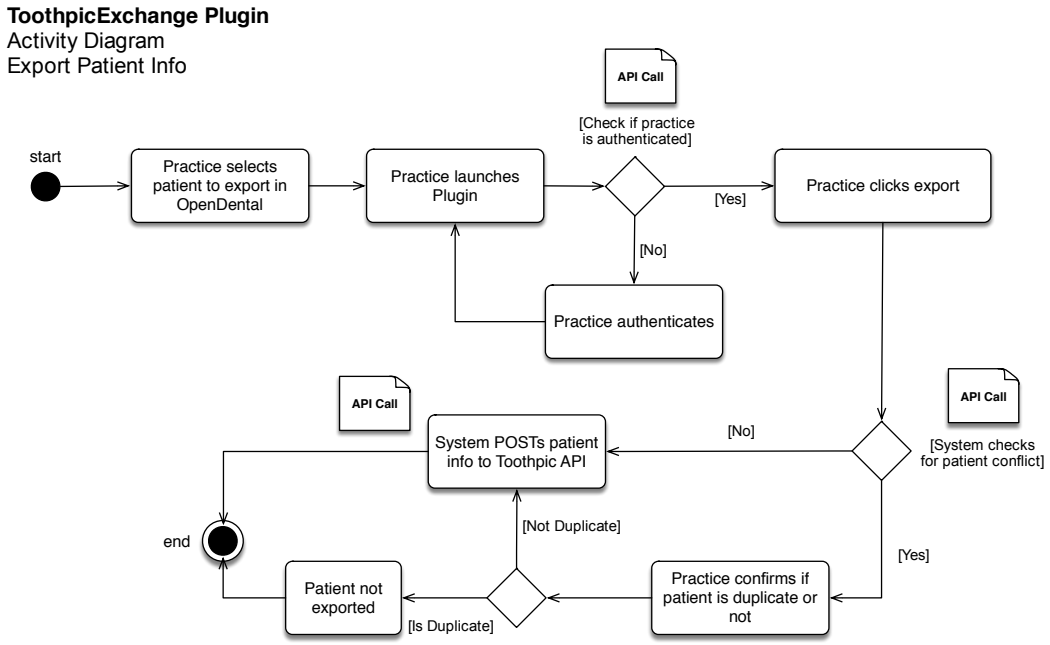


Figure 3.2.3 – The Export Function Activity Diagram

These activity diagrams outline the four main screens in the plugin:

- Authentication (login)
- Homepage
- Import Screen (list of Toothpic users)
- Patient Edit Screen (to allow the practice to add additional info)

The patient edit screen was already included in the OpenDental software, so the remaining three were to be designed from scratch. A dedicated Patient Export Screen was not required because once the “Export Button” was pressed, a message box would inform the user whether the export was successful, unsuccessful or whether there was a patient conflict.

Ideally, the design process would have entailed some interaction with end-users either through discussion or interviews. Unfortunately, this was infeasible for two reasons: firstly, because all of OpenDental’s customers are located in the United States; secondly, because Toothpic have only partnered with assessors thus far, and not actual practices. In order to combat this, the functionality was designed to loosely resemble OpenDental’s software.

The OpenDental user interface was inspected, and while the plugin interface was generally designed to match this, it was intentionally made to look slightly different from the OpenDental experience. The reason for this was to give end-users context to the plugin and to heighten awareness that they were interacting with a separate system when using the Toothpic plugin. The plugin still functions in a manner that OpenDental users would be familiar with.

The final design consideration within the system functionality stage was authentication. Authentication and security are often regarded as an after-thought; however, it was important that every facet of the system’s design considered authentication.

In a clinical setting, it is imperative that authentication procedures are followed correctly. This is to ensure that Personally Identifiable Information (PII) and Protected Health Information (PHI) do not become compromised or fall into the wrong hands. Even though the presence of authentication is only visible in the login screen, it plays an important role behind the scenes. Special consideration was given to how the system was designed so that PII or PHI would not be visible or accessible by third parties. For example, the classes and methods are designed so that they are not accessible by another plugin in OpenDental. Further to this, OpenDental itself cannot see either authentication credentials or PII/PHI until the plugin discloses them.

3.3 Development

The development phases constituted the greatest amount of time in each system increment and made up for the greatest change in software between increments. Therefore, the development section below is discussed by increment. The focus points for each of the four increments were:

1. Forms and User Interface
2. Classes and logic
3. API Interaction
4. Authentication

The plugin was developed with C# in Visual Studio. C# is an object-oriented language, which means almost every data structure is represented by an object. Each object has its own inherent attributes (e.g. properties, data) and methods (e.g. functions). This allows great flexibility as objects can be passed around without needing to know how they are structured or how they work. This concept is referred to as encapsulation (Lee, 2005).

For the client's future development, this means the framework for their user model is already set up and configured in the plugin, and it should be relatively simple for them to add new methods and increase functionality. It also allows greater security, as it means data is not left exposed or accessible to parts of the software that shouldn't see it.

Forms and User Interface

The first increment entailed configuring all of the forms for the plugin. User interaction exclusively takes place through forms, and the framework used was Windows Forms (the same as OpenDental). Windows Forms can be designed and configured using the designer in Visual Studio.

The three forms (authentication, homepage, import) were set up in the designer, and their event handling was programmed. Event handling is how the software reacts to button presses, text box entry etc. Their data flow logic was next developed, this concerned how the forms passed data on to each other and to the software. This was all developed using testing variables which were drafted from the data models.

The plugin needs an entry point from OpenDental (i.e. there is a button on the toolbar), so this was developed last.

Once this was all done, the software was built, run in a debug version of OpenDental, and tested. Following testing, various bugs were eliminated and efficiencies were improved. No major design changes were required at this point.

Classes and Logic

With the forms set up, the next increment focused on developing each of the classes for the plugin. The relevant classes are listed in Table 3.3.1.

Table 3.3.1 – The Fundamental Classes

Class	Description
ToothpicUser	Represents a Toothpic user as an object (e.g. name, email, gender etc.)
ToothpicAPI	Represents a connection to the API as an object (e.g. the API URL, endpoint etc.)
UserAuth	Represents an authentication object (e.g. auth token, user ID etc)

Each of their attributes were developed, and so too were their methods. The concept of encapsulation is seen as particularly important for the UserAuth object. The UserAuth object's attributes contain all of the details necessary to make an authenticated API call. However, these attributes are kept private inside the ToothpicAPI class. Thus, a ToothpicAPI object can be passed around to make authenticated API calls but it never exposes the information it contains. This is also more secure than hardcoding or storing a username/password locally in the ToothpicAPI object.

The ToothpicUser class was developed to map to both an OpenDental "patient" object and to Toothpic's API. Thus, it contains all the methods necessary to convert user information between the various systems.

More detailed class diagrams and modelling can be found in Appendix D (p. D-1). The diagrams were found to be useful in understanding both the scope and domain of this system, and of the OpenDental/Toothpic systems. Hence, they should remain relevant beyond this project's lifecycle.

With the classes developed, the testing variables used for the forms could now be replaced with actual objects. This was heavily tested, and the design of the classes was tweaked.

API Integration

The objective of the next increment was to get the plugin communicating with the client's API. It was not possible to connect to the actual API because access is restricted to production service only. The API is strictly controlled due to the client's HIPAA compliance. In order to work around this, a local test API server was set up to emulate the client's API. Having a local server enabled greater control and experimentation with the API, as functionality only need be

built and simulated locally. Json-server (Typicode, 2017) was used to emulate the API locally, and the data models from the design phase were used to configure it.

The client's API is RESTful, which is an architectural model based around the concept of requests, responses, and resources (W3C, 2004). A *request* for a *resource* is sent by the plugin to the server, and the server returns the appropriate *response*. The response can contain the resource itself, or some other information. This all takes place over the HTTP(S) protocol. In this case, the server's responses are delivered in the JSON format (JavaScript Object Notation) which is an array of key-value pairs (IETF, 2014). A key-value pair is simply an identifier matched with a value (e.g. "authToken" = "1234" is a key-value pair identified by "authToken" taking on the value of "1234").

Each type of request has a particular method; the four most relevant are listed in Table 3.3.2, using the CRUD framework (Create, Read, Update, Delete) to illustrate. The resource is normally some type of object, in this case not only patients but also authentication objects, etc. Each of these objects/resources translates to an endpoint. The endpoint is accessible via a standard HTTP URL (Uniform Resource Locator) (e.g. https://api.toothpic.com/users/). The client's main API endpoints are listed in

Table 3.3.3. The two most relevant endpoints were "users" and "auth".

Table 3.3.2 – Explanation of REST methods

REST Methods	**Description**
POST	Create a resource
GET	Read a resource
PUT	Update a resource
DELETE	Delete a resource

Table 3.3.3 – The client's API endpoints

Endpoints	**Description**
users	All of the user demographic info.
auth	Assorted authentication and identification tokens.
question_answers	Additional information available about the user.
case_report	The framework for user case reports.

RestSharp (RestSharp, 2018) is a package that enables software written in C# to communicate with a RESTful API. Although C# does include functionality to connect to APIs, it is quite complex to set up and use. As a result, RestSharp was used as it simplified the code required to connect to the API. This means that additional API calls can be added with ease as the software expands functionality-wise. Other than RestSharp, the plugin does not rely on

any other packages. It was intentional to have a low number of dependencies in order to simplify software maintenance.

The API methods within the plugin immediately convert the JSON responses into one of the local objects such as ToothpicUser. For example, this ensures that when a “user” is requested from the API, it is converted directly into a ToothpicUser object locally (instead of an array of string-based user properties). This means it can take one or two lines of code to request a user from Toothpic’s API, and immediately run various local methods on it (e.g. convert to OpenDental patient, verify information, calculate age etc.). The time-consuming work of parsing and converting has all been taken care of.

This concluded the API development required. Following this, static objects were replaced with actual data from the test API, and the mapping could be tested. Some additional methods were added to the classes to facilitate easier interaction with the API.

Authentication

The final increment entailed the development of authentication into the plugin. Although authentication was already designed into the software, the reason for this was to allow greater creativity in the earlier development processes. As the final increment, it also permitted the entire code to be reviewed and altered so that it met with authentication and security requirements.
The client’s authentication process requires fetching a number of tokens from the API. It is based on the JSON Web Token (JWT) standard (IETF, 2015).

When the practice logs in initially with their username/password, the API returns a set of unique, one-time-use tokens. The plugin then uses these tokens for all subsequent authenticated API calls. The API will not respond to unauthenticated calls (i.e. without tokens). This ensures that the end-user’s username and password are never stored locally. It also allows the tokens to be revoked at any time if they become compromised.

The plugin fetches the tokens initially and stores them in a single UserAuth object locally. As discussed in the “Classes and Logic” sub-section above (Section 3.3, p. 17), the concept of encapsulation protects all of the information contained in the UserAuth object. The sequence diagram in Figure 3.3.1 visualises the process.

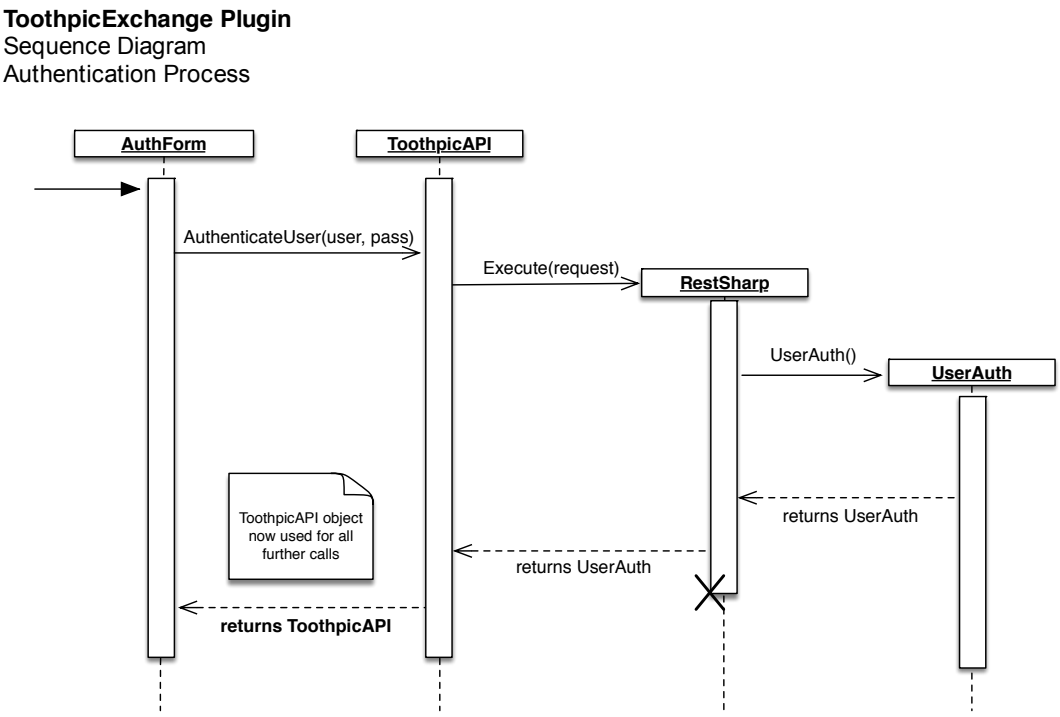


Figure 3.3.1 – Sequence Diagram for Authentication

At this point, the design of some classes was altered (e.g. many properties were changed from a public declaration to a private one). The process of retrieving authentication tokens from the API was developed, and the user login flow was developed.

This concluded the development process of the system.

3.4 Testing

The system was tested at each increment to ensure it met the requirements. In order to achieve this, adequate verification and validation mechanisms were put in place.

Verification revolves around the idea that the software operates the way it was designed to, and validation is concerned with the software meeting the system requirements (see requirements in Appendix D, p. D-5).

Verification

Unit testing was undertaken for each increment. An example of the type of tests are as follows:

- Event handling (e.g. the correct event is triggered when a button is clicked).
- Mapping (e.g. the user's date of birth was translated correctly).
- Authentication (e.g. certain functions could only be triggered once the user was authenticated).

Validation

Functionality testing was undertaken for the final system, this ensured that the plugin offered all of the functions that were set out in the system requirements.

Usability testing (e.g. the user interaction) would have been desirable. Unfortunately, it was infeasible to undertake this with actual users, as they were inaccessible. Instead, a more informal usability testing was followed whereby the user interface was tested in comparison to OpenDental's user interface.

The full testing documentation is included in Appendix G (p. G-1).

3.5 Difficulties Encountered

There were three noteworthy difficulties. Two were expected, one of them was not.

The first difficulty stemmed from the requirement to interface with two existing systems (Toothpic and OpenDental). This was expected and is a challenge for many middleware systems. The design process was quite restricted in that all of the data models and methods had to map to existing functionality in Toothpic or OpenDental. Further to this, programming what would have been a relatively simple task for a modern web app (developed in say Node.js or Ruby on Rails) was made more complicated for being written in C# Windows Forms. In order to address these challenges, extensive investigation was conducted into how the two systems worked. This involved repeatedly using the systems and tracking the background activity with various proxy and debug tools (see Section 3.2, p. 11). A greater understanding of the systems, combined with the action of describing their models and their context (see Appendix D, D-1) facilitated a conceptual mapping to be determined whereby the systems could be bridged. What resulted was a plugin with a very solid, although straightforward, framework written in C# which Toothpic can utilise as they expand their API functionality.

The second challenge was the extensiveness of the OpenDental package. This was also expected when the terms of reference were formed. OpenDental has a very wide scope of functionality, and it took a significant amount of time to understand how it works and how a plugin could interface with it. Once again, this challenge was overcome through extensive investigation into the system. OpenDental operate a forum, although cumbersome to explore, which contained much useful information for familiarising a new user with the system. They also publish video webinars, often with walkthroughs of the software. These were valuable in understanding how practitioners actually use the system. These resources coupled with actual usage of the system, lent practical insight into what functionality the new plugin should offer, and how it should achieve this.

The third challenge was unexpected but controlled. This was the lack of developer documentation for both OpenDental and Toothpic. In order to combat this, OpenDental and a variety of third-party plugins had to be examined through their source code. Test plugins were developed with a lot of trial and error until the OpenDental plugin interface was understood. Toothpic's API was similarly examined by means of a proxy server as detailed in the first

challenge. Both cases were time-consuming and could have been made more efficient with documentation. The major risks these posed were mostly concerned with time management. However, tasks were prioritised, and the project was delivered on time. This is why an emphasis was placed on creating living, usable documentation for this system, so the system can be easily understood and utilised in the future.

4 RECOMMENDATIONS

This chapter summarises a brief conclusion and outlines recommendations to the client for growing their system.

4.1 Conclusion

This project was born from a broad goal that was to be explored as part of the client's business strategy going forward. This goal was concentrated to a set of requirements that were feasible to achieve within the project's timeframe and resources. This is exhibited in the project terms of reference (Section 1.3, p. 2) and the requirements documentation (Appendix D, p. D-2).

The system was requested to be a "proof-of-concept", which it most certainly is. Although it could not be implemented in a production environment and was not required to, the software is ready to be deployed with minimal configuration. The project source code (Appendix F, p. F-1) and the testing documentation (Appendix G, p. G-1) demonstrate this.

The domain and context of the two systems were sought to be understood. This is also highlighted in the design documentation (Appendix D, p. D-1) and in the review on eHealth standards (Appendix H, p. H-1)

Finally, the included documentation provides details on the development of the system and its current functionality. It contains all of the information required to further develop the system and presents the information in an easy-to-consume and easy-to-evolve manner. This is exhibited in the included HTML developer manual, but also in Appendix C (p. C-1).

Shane Owens was presented one of the final increment prototypes at a meeting on 8 March 2018, and the feedback was positive.

In summary, the software written constitutes a sound framework for further development. With its object-oriented design, secure implementation and extensive documentation, it will serve as a springboard for future extensibility.

4.2 Recommendations

Documentation

The lack of documentation took up a significant amount of time in understanding both OpenDental and Toothpic's systems. If documentation is prepared and maintained in line with development (as was the case for this project), then it can only ensure future evolution can occur more rapidly. A conscious effort was made to document this process not only with the software but also with associated design and requirements documentation. Hopefully the documentation will continue to evolve as this strand of the client's business does too.

Unique Identifiers

As discussed in “Data Modelling” (Section 3.2, p. 13), the client currently does not record a user’s Social Security Number. This means there is a slim chance that patient information could get mismatched in the event of a same name and birthdate within a specified zip code.

It is recommended that the client begin to collect a unique identifier such as a Social Security Number (SSN) for when an exchange of patient information takes place.

The SSN can remove much of the uncertainty surrounding patients with similar identification. A user would not necessarily need to provide an SSN when signing up initially, however it could be collected if and when the user is being handed over physically to a practice.

Interoperability

This is no easy feat to achieve, and it is on the top of every healthcare organisation’s wish list; however, building a more interoperable system will help the client integrate into practices on a much larger scale. This report cannot prescribe a solution to achieve interoperability. However, it can suggest areas which the client should consider pursuing further.

Data Modelling

The client currently uses a proprietary data model for their system. It is recommended that the client investigates OpenEHR as a means to structure their data models. OpenEHR is a consortium of eHealth professionals working to define standardised clinical archetypes. Their approach is community driven, so each archetype has to go through several stages of approval before being published. This ensures that eHealth systems are truly working off the same set of reliable data models. Many of OpenEHR’s archetypes are still in the draft phase, especially the dental archetypes. However, as it is a community endeavour, the client could define their own archetypes for consideration in an OpenEHR release.

OpenEHR has two ambitions: the first of which is interoperability; however, their second ambition is to support a migration strategy using their technology-agnostic model. This means, that regardless of the data transfer protocol being used (e.g. HL7 v2 or FHIR etc.), it ensures the data models at the heart of systems can still be understood and communicated with external actors. A migration strategy allows an organisation to easily migrate information between technologies, and a solution such as OpenEHR would support this.

Standardised Transfer Protocols

Although the software developed could not integrate any of the eHealth standards discussed in Section 3.2 (p. 11), it is only a matter of time before OpenDental (and other providers) introduce support for the HL7 FHIR standard. It is recommended that the client pursues integrating the FHIR standard into their own API. As outlined in Appendix H (p. H-4), it is a RESTful API that would integrate quite easily with the client’s API. The only reason it could not be used for this project was because OpenDental did not fully support it.

Because FHIR is a generic standard, the client could restructure their entire API to be FHIR compliant. This would mean instead of mapping their system to a FHIR output, their system could natively operate on FHIR. This would enable a standardised interface which means a system such as the one developed here could be implemented in a number of hours.

The review in Appendix H (p. H-4) also discusses SMART on FHIR. This is an amalgamation of two programmes focused on creating plug and play medical apps that are almost 100% interoperable. Toothpic could develop their practice integration service as a “SMART” app which would slot into a SMART-compliant PMS in a manner of minutes.

In spite of the promises both FHIR and SMART offer of interoperability, it is worth noting that both specifications are still very much in the development stage. FHIR is currently still a draft standard and will perhaps remain that way for a number of years. Further to this, neither SMART or FHIR would address the data modelling issue that OpenEHR does.

It is recommended that the client investigate how they model their information internally (i.e. with OpenEHR), and then investigate the possibility of implementing FHIR and/or SMART into their API. From a technical perspective, FHIR and SMART would seem to be promising endeavours; however, if the client’s data models are limited to work only in FHIR then they will not be able to migrate to a new system if and/or when FHIR becomes outdated.

HL7 v2 has also been discussed and while it does market itself as an interoperable standard, it too suffers from the data modelling problem. Developing a generic HL7 v2 module would not be programmatically difficult to implement, however its return would probably not be great in the long run. The review in Appendix H (p. H-4) provides further detail.

Coding Systems and Nomenclature

OpenDental utilise standardised coding systems and standardised nomenclature in their software. This is also the case for most other PMSs and Electronic Patient Record (EPR) systems; even OpenEHR supports them. These codes can be used to identify pharmaceuticals, medical conditions, diseases and clinical procedures. Some of the systems configured in OpenDental include SNOMED-CT, ICD-10 and the American Dental Association’s CDT (Current Dental Terminology) – further details on these are available in Appendix H (p. H-6).

Such standardised systems would allow the client to exchange a greater amount of valuable patient information in a systematic, efficient way. The client’s current nomenclature is self-developed. However, it is recommended that this is either mapped to, or replaced with a system of standardised nomenclature.

A Venn diagram in Figure 4.2.1 illustrates the overlap between the Toothpic and OpenDental data models. The size of the ellipses is representative of how much information they can hold on their patients. The shaded area (demographic info) depicts the number of elements that could directly be mapped to each other, and hence was the level of integration achieved by this project.

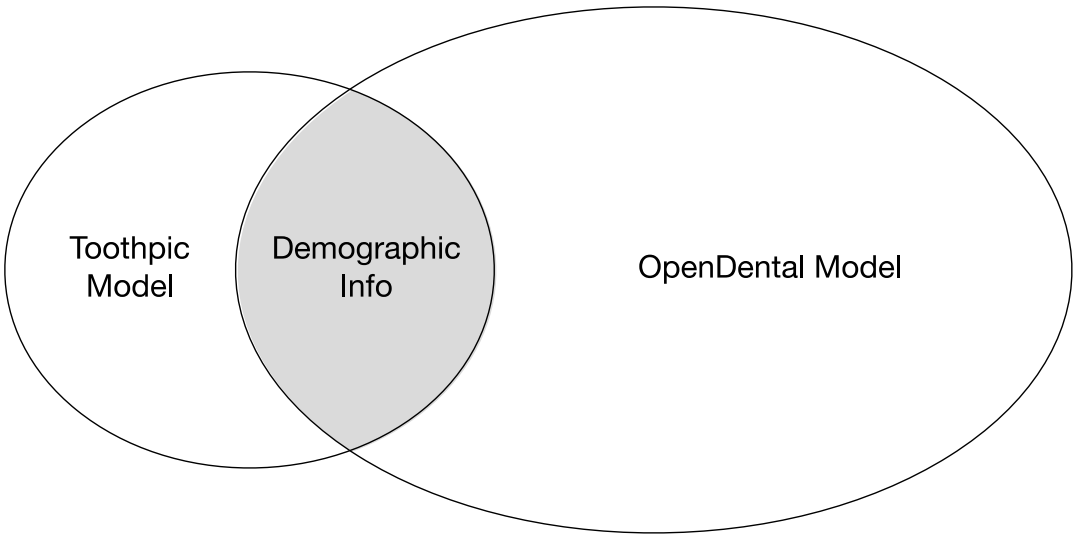


Figure 4.2.1 – The Two Systems' Overlap

In order to grow the overlap, the client needs to address the issue of interoperability. Not only within their own system, but within the wider eHealth community. As teledentistry is a new concept, pioneered by the client, it is recommended that they become involved in the definition phases of many of the open standards discussed (OpenEHR, FHIR and SMART).

Greater interoperability means better services and better decision making. It is by no means a simple task to achieve, and this project merely chips away a small piece of a much larger issue. Although, if the project findings and deliverables are used to the best they can, then the client can truly begin to establish themselves as an interoperable pioneer in the teledentistry sector.

A. ORIGINAL PROJECT OUTLINE

Client:	Teledentistry Firm (toothpic.com / OralEye)
Project:	Integrating a Teledentistry Firm's System with Electronic Health Records (EHR) Providers
Location:	40 – 41 Dame Street, Dublin 2
Client Contact:	Shane Owens, CTO, shane@oraleye.com
Dept. Contact:	TBC

Client Background

OralEye is a start-up based in Ireland with a network of 159 dentists across the United States. OralEye is the first teledentistry platform to connect patients with dentists for virtual check-ups and examinations. OralEye operate a consumer app *toothpic* which allows patients to send photographs of their teeth for review within OralEye's network of dental practitioners.

Project Background

As of recent, OralEye/toothpic operate a proprietary internal system which requires a significant amount of data to be manually input and extracted – for both the patient and dentist. This project entails developing a protocol that would integrate OralEye's system with common Electronic Health Record (EHR) providers and suppliers. This has two potential business use cases - a patient could have their personal data and dental history directly imported into the app, without having to manually input. Similarly, OralEye could package and export data for direct integration with dental practitioners' internal Practice Management Software.

Client Requirement

The client requires a report comparing a variety of EHR suppliers/systems in order to develop a standard for integration. The top two suppliers have been named as Epic and Dentrix, however, this project would entail researching other suppliers, compliance legislation, data protection etc.

The client also requires the development of a physical system/protocol which would implement this integration. Thus, following completion of the project, OralEye/toothpic could begin to integrate their system with a leading EHR supplier and/or Practice Management Software.

The client would require testing and validation of said integration.

What is involved for the student?

The student must research and recommend one or more EHR suppliers for the client to integrate their systems with. The student must document and develop a protocol for integrating with this supplier. The student then must implement a system for integrating the client's systems with said supplier(s) and must test the implementation.

B. INTERIM REPORT

Project: Proof-of-concept integration of a Teledentistry Firm's system with an Open Electronic Health Record Provider
Client: Toothpic / OralEye
Student: Dara Ó Cairbre
Supervisor: Gaye Stephens

Review of Background and Work to Date

Toothpic is a teledentistry service who provide virtual check-ups for users over a network of dentists across the United States. With the growth of telehealth over recent years, and a projected increase of usage in the future, Toothpic is investigating how their teledentistry platform can be integrated into physical dental practices.

Toothpic is interested in how their service can both send and extract rudimentary patient information from dental practitioners within their network in an efficient and seamless manner. The primary method in which this will be facilitated is through Electronic Health Record (EHR) exchange.

Work to date includes preliminary research on the eHealth and EHR industry. Summary as follows:

- EHR implementations vary hugely across practitioners, and interoperability is a major issue. Most practitioners use different EHR providers which operate proprietary software that is not interoperable.
- There are a number of EHR providers Toothpic could integrate with, however OpenDental is the only provider whose API and data model are freely accessible.
- Hence, it has been agreed that OpenDental will be the chosen provider for this project.
- The Health Level 7 (HL7) standards define a messaging protocol for transfer of EHR.

Terms of Reference

1. Identify Toothpic's requirements for integration and describe their data model.
2. Identify and describe the OpenDental model within the bounds of Toothpic's integration.
3. Describe the domain and context of the OpenDental model.
4. Implement a proof-of-concept integration for data transmission between Toothpic and OpenDental as per requirements and models outlined above.
5. Provide implementation documentation to assist Toothpic with future integration.

Notes:
The student's code cannot touch Toothpic's production code (due to HIPAA compliance). Therefore, it can only be implemented in a test environment.
The contents of the project may need to be confidential. Details from the client to follow.

Further Work

Hilary Term, Week 1 - 3 (Fri 2nd February):

Student will establish requirements and draw up models for Toothpic and for OpenDental. Student will investigate possible methods of implementation and will decide on a method.

Hillary Term, Week 4 - 8:

Student will implement EHR integration with OpenDental as per chosen method.

Hillary Term, Week 8 - 10:

Report write-up.

Conclusions

As it stands, the client has zero EHR integration. Both the student and the client are aware of how tricky EHR solutions are to implement, and that there is no one size fits all solution. The aim of this project is to present a proof-of-concept Open EHR implementation which facilitates basic patient data transfer, and to scope the context of OpenDental and Toothpic's systems. Full integration with OpenDental is not guaranteed by completion of this project, however it will (ideally) prove that it is possible. The documentation will also assist Toothpic in developing future implementations whether with OpenDental or another EHR provider.

C. DEVELOPER MANUAL

The developer manual has been reproduced below. However, it is best experienced on the HTML website in a web browser, or the attached PDF in a PDF viewer. The manual's API documentation has not been included here due to its size. It is available on the website or the PDF.

Welcome to the ToothpicExchange manual.

This HTML website contains the relevant information to setup, to use, and to further develop the ToothpicExchange system which has been created for Toothpic/OralEye Ltd.

This documentation has been created with DocFx (console version 2.23). It contains static information alongside dynamic content generated from ToothpicExchange's source code. The source code generated content utilises C# XML Documentation Comments

How to use this website:

There are two main sections.

The system overview is best for setting up and configuring the plugin. The API documentation may seem daunting at first, however it is only necessary for development of the source code.

System Overview

This section contains general information about the system. Including:

1. General Introduction
2. How to Setup the Plugin
3. How to use the Plugin
4. How to setup the developer environment

The information in this section is static content contained in the project's markdown files (.md)

API Documentation

This section contains a comprehensive reference of the source code. It details the classes, their methods, properties and more. It can be used in tandem with the source code, and it can be updated as the source code changes.

Much of this content is created from the XML tages within the source code. In order to update this content, the XML tags should be edited in the source code and the project rebuilt.

With DocFx.Console configured in Visual Studio, the site will update itself everytime the project is built.

The HTML files are output to a `_site` folder within the project.

Much of the content contained in this section is also visible within Visual Studio's *IntelliSense*. This is extremely convenient as the relevant documentation appears in Visual Studio as you type, there is no need to open a separate reference.

It should be noted that DocFx currently do not support publishing of `private` defined methods or attributes. Thus, these will not appear on the HTML site however they are still documented in Visual Studio through XML comments, and they will show up in IntelliSense.

Introduction

ToothpicExchange is a system which allows the exchange of patient demographic information between Toothpic's API and a custom plugin for OpenDental

Information

- Written in C# (7.2) .NET (4.5) and Windows Forms
- Written with OpenDental 17.2 source code and plugin framework

Plugin Installation

Click here for instructions on how to install the OpenDental plugin.

Dependencies

- Microsoft .NET version 4.5
- C# version 7.2
- RestSharp version 105.2.3
- OpenDental 17.2
- (Visual Studio 2017 or other IDE)
- Json-Server 0.12.1 for test environment
- DocFx.console version 2.23 for documentation

Folder Contents

```
- Project source code
- Documentation
- Compiled DLL ready to go
- Screenshots
- OpenDental source code
```

Plugin Setup

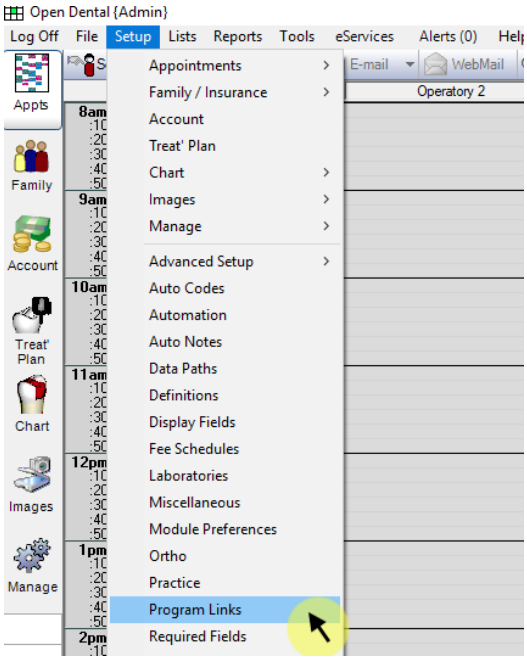
Source Code

The source code is included with the project. This can be altered, however in order to install the plugin in OpenDental, it must be compiled as a `.dll` file. The file *must* be called `ToothpicExchange.dll`. If no modifications are required, a pre-compiled version is included.

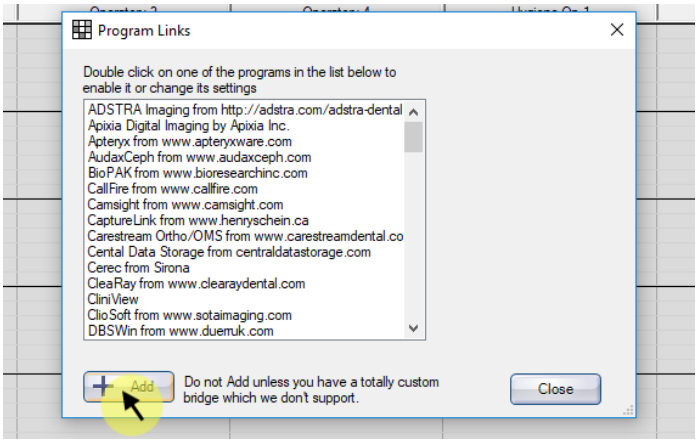
Setup in Open Dental

1. If not already compiled, the sourcecode must be compiled into a file called `ToothpicExchange.dll`
2. ToothpicExchange.dll must be copied to OpenDental's folder e.g `C:\Program Files\Open Dental\`

3. In the OpenDental main menu, click `Setup` > `Program Links`.



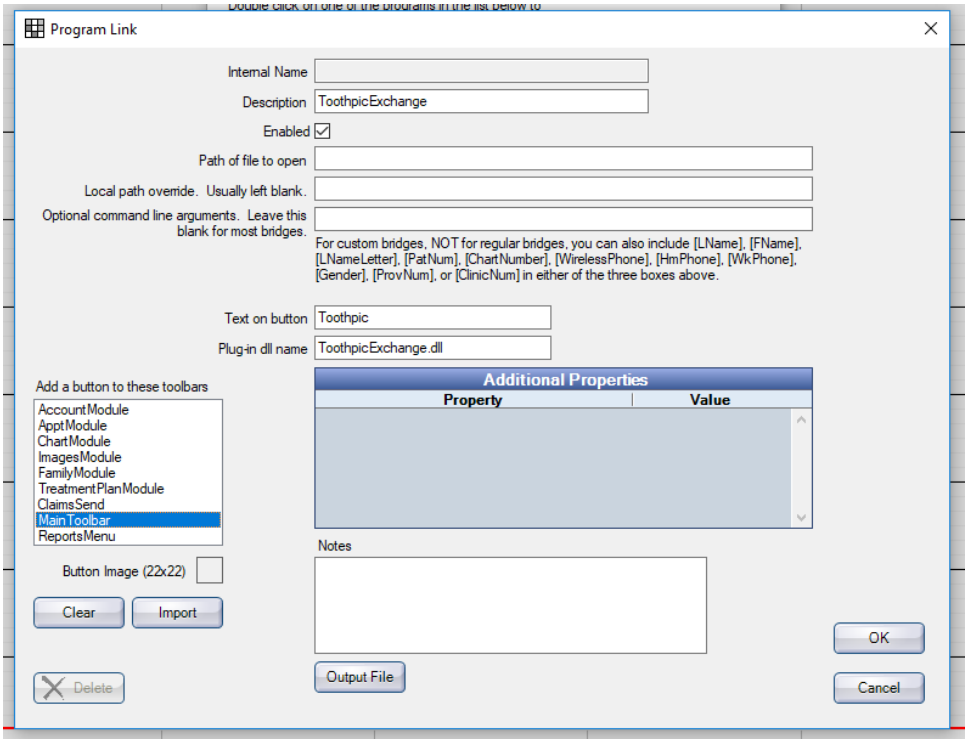
4. A new window will open with a list of Program Links already established, click the `Add` button in the bottom left hand corner.



5. A new window will open to enter the plugin settings. Input the following:

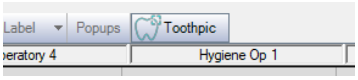
```
Description:        ToothpicExchange
Enabled:            [Checked]
Text on Button:     Toothpic
Plug-in dll name:   ToothpicExchange.dll
```

Ensure the option `Main Toolbar` is selected in the section `Add a button to these toolbars`.



6. Save the settings.

7. The Toothpic plugin will now be added to the Main Toolbar in OpenDental.



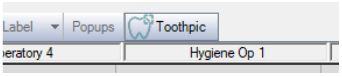
For more information, consult the OpenDental plugin framework page.

Using the Plugin

The plugin is relatively straightforward. It is accessed through the OpenDental toolbar which is accessible in every module. The plugin offers functionality to import or export patient information to/from OpenDental and Toothpic.

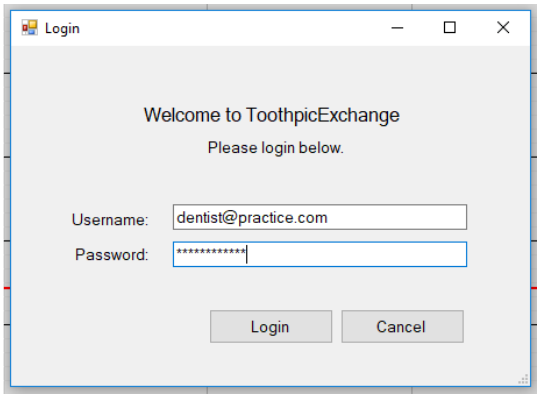
Opening the Plugin

Once the plugin has been installed, simply click the Toothpic button in the main toolbar of OpenDental.



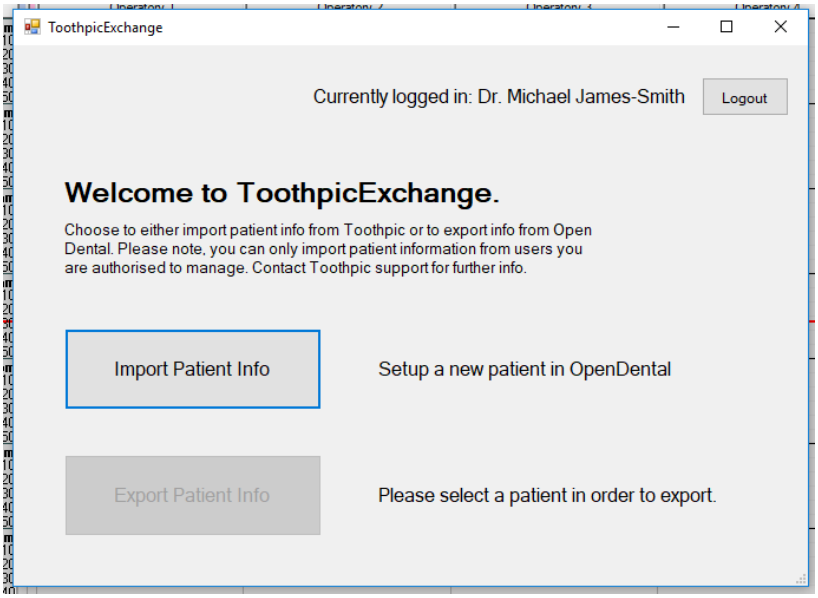
Logging in

The plugin will prompt the practice to login with their Toothpic `username` and `password`.



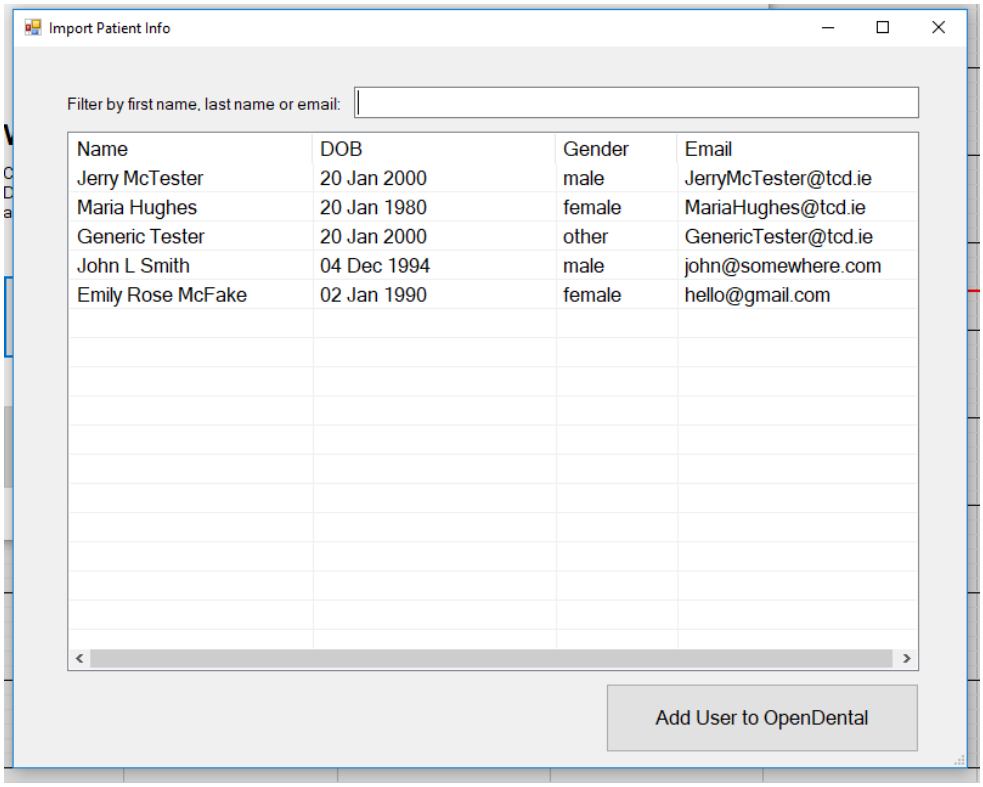
Homepage

Once the practice has been authenticated, ToothpicExchange will launch. This is where the practice can either import or export patient information.

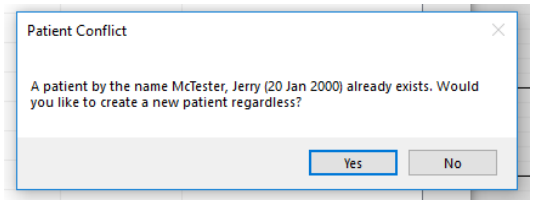


Importing Info

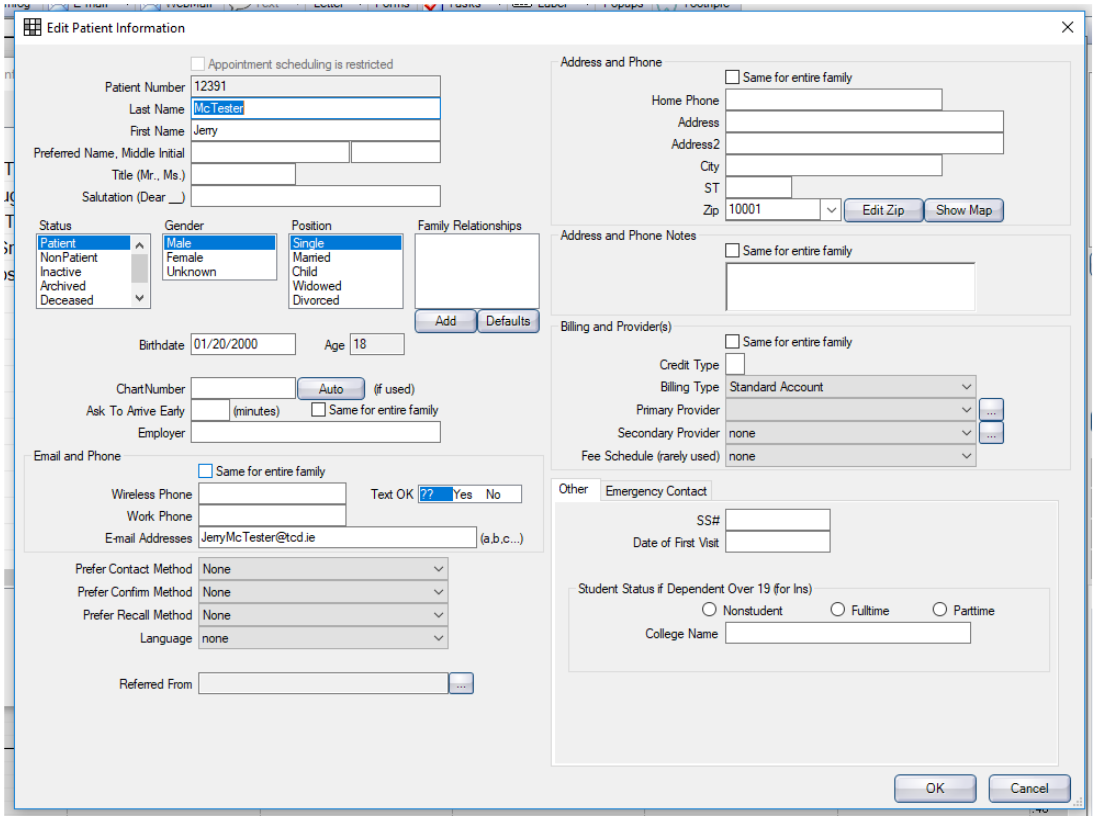
In order to import patient information, simply click the `Import Patient Info` button to begin. A new window will open with a list of users available to import. These users are loaded from Toothpic, and thus only users who the practice are authorised to view will appear on the list. There is a search box at the top of the form to filter users by first name, last name or by email address.



Select the user to import, and press `Add User to OpenDental`. If a user with the same name and date of birth already exists in OpenDental, the software will warn about a patient conflict. The practice may review this information and choose to proceed if they are not duplicates.

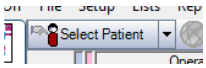


The patient edit form in OpenDental will now launch, and the practice may insert additional information about the patient. To save the information, click `OK`, otherwise `Cancel`.

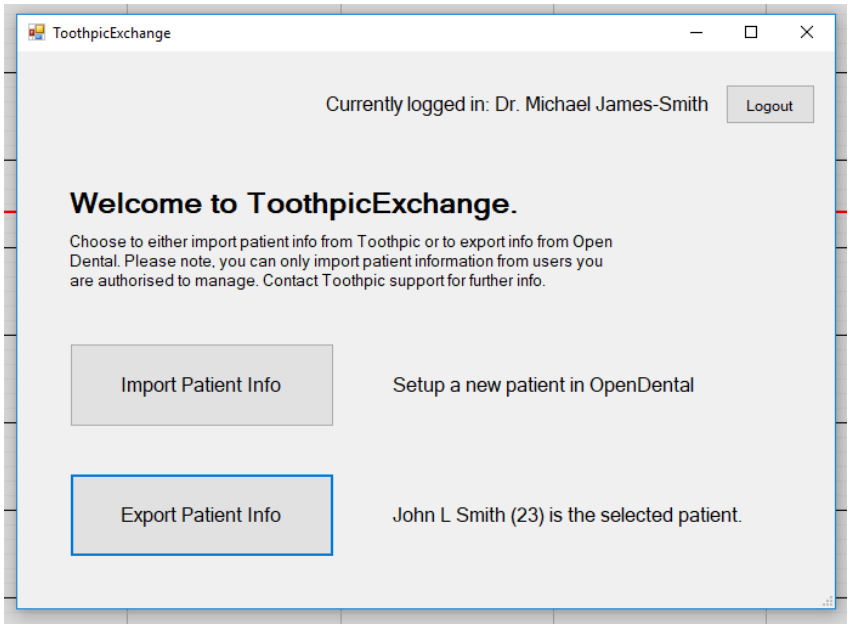


Exporting Info

Return to the homepage of the ToothpicExchange plugin. A user must be **selected** in OpenDental before proceeding with an export. To select a patient, the practice must click the `Select Patient` button in the top left hand corner.



Once a patient has been selected, the option to `Export Patient Info` will now become available. Click it to export.



This will send the patient to Toothpic's API and will advise the practice whether this was successful or not.

Setting up the environment

The technical environment to set up for the plugin can be quite complex. It is fairly straightforward to load the plugin directly to an instance of OpenDental (e.g. the trial version). These instructions outline that process. However, in order to setup a development environment you must follow the steps below.

Visual Studio Installation

It is advisable to install Visual Studio IDE 2017 with a minimum .NET 4.5 and C# 7.2

Once this is done, the root `ToothpicExchange folder (the folder directly containing the` ToothpicExchange.sln` file) should be copied or set up in a new folder. This new folder will contain all of the plugin source code.

OpenDental Developer Installation

The OpenDental source code must now be downloaded. This is a two step process.

1. The first step involves downloading the trial version of OpenDental and installing it. This installs all of OpenDental's additional dependencies and sets up the MySQL server locally. Follow the OpenDental instructions for advice on how to install OpenDental locally.
2. The next step is to actually download the source code. This can only be done through OpenDental's version control system, Subversion. Please consult this webpage for instructions how to download and set up OpenDental's source code. It is advised to store the OpenDental source code in the same HEAD folder as the ToothpicExchange folder.

Now, open the `ToothpicExchange.sln` solution file to begin setting up the technical environment.

In the Visual Studio Solution Explorer (top right hand side) there should be three projects:

- OpenDental
- OpenDentBusiness
- ToothpicExchange

1. It is likely that the two OpenDental packages will be indicated as "missing". If that is the case, you must right click and remove them from the solution.
2. Next, you must right click on the solution and click `Add` > `Existing Project`

   Browse to the following and add them: `../OpenDental/OpenDental.csproj` and `../OpenDentBusiness/OpenDentBusiness.csproj`
3. Next, right click on the ToothpicExchange project in Solution Explorer and click `Add` > `Reference`.

   On the right hand side click `Projects` > `Solution` and enable both OpenDental and OpenDentBusiness.
4. Now right click on the ToothpicExchange project in Solution Explorer and click `Properties`.

   Select `Build Events` on the left hand side, and edit the "Post-Build event command line". This should reference to a batch file called CopyDllToOd.bat located in the project folder.

   You will need to alter the absolute path in the properties so that it points to the correct file. This file takes care of transferring Visual Studio's DLL build of the plugin into the required folder in the OpenDental debug bin.
5. Now, open the `CopyDllToOd.bat` file in the project folder, and edit the absolute path to match the OpenDental `bin/debug` folder.
6. Right click on the OpenDental project in Solution Explorer and set it as the startup project.

7. Finally, try build the project and see if it runs. It could take a while initially.

For troubleshooting, and further info, consult the OpenDental plugin setup page

Setting up JSON-Server

1. Setup a folder to contain the server files and navigate to it in the Command Prompt.
2. Json-server runs on Node.js, so you will need to install the Node.js package manager (npm).

   Navigate here to download and install npm
3. Now you can utilise npm to install json-server. Run the following in the Command Prompt:

   ```
   npm install -g json-server
   ```
4. Create a `db.json` file, there is a template included in the project folder.
5. Run the following command:

   ```
   json-server db.json
   ```

   This will start a new local server operating on port 3000. It will respond to GET requests for any objects contained in the `db.json` file, and it will add any new objects to the `db.json` with a POST request.

For further information, consult the json-server github.

Product Documentation

Navigate to the DocFx website for information on how to download and setup the latest version of DocFx.

This project utilises the DocFx.console variant which can be downloaded in Visual Studio through the NuGet Package Manager.

There are `docfx.json` and `docfxPDF.json` files in the project directory. These files are used to specify the DocFx configuration. It is recommended these are not changed, unless you are familiar with the DocFx environment

Configuring the Plugin

There are a few minor configuration items to tackle before deploying the plugin.

API Base Url

In the `APISettings.settings` file contained within the project folder, there is a property labelled `ToothpicAPIBaseURI`. This value is the API's base URL.

This property is referenced in `ToothpicAPI.cs`. The `APISettings.settings` file is compiled when the project is built, so this setting must be configured before building and deploying the plugin.

API Endpoints

In `ToothpicAPI.cs` there are two private variables `authEndpoint` and `userEndpoint`. These can be changed as necessary.

Currently there is no other configuration required.

D. DESIGN AND TECHNICAL DOCUMENTATION

Systems Development Methodology

The incremental model was followed for systems development. This consisted of four major increments, with each increment containing four phases. The four phases were requirements, design, development, and testing. Each increment resulted in a working piece of software, with functionality added to each increment.

“Implementation” and/or “maintenance” are normally included in the incremental model, however, these were left out as the software was not deployed to a production environment.

The four major increments focused on the following areas of functionality:

1. Forms and User Interface
2. Classes and Logic
3. API Integration
4. Authentication

Figure D.1 illustrates the addition of functionality over the project lifecycle. The grey boxes portray the increment, and each of the subsequent white boxes portray the individual phases within the increment. The overlap of phases is indicative of the intertwining, dynamic development process.

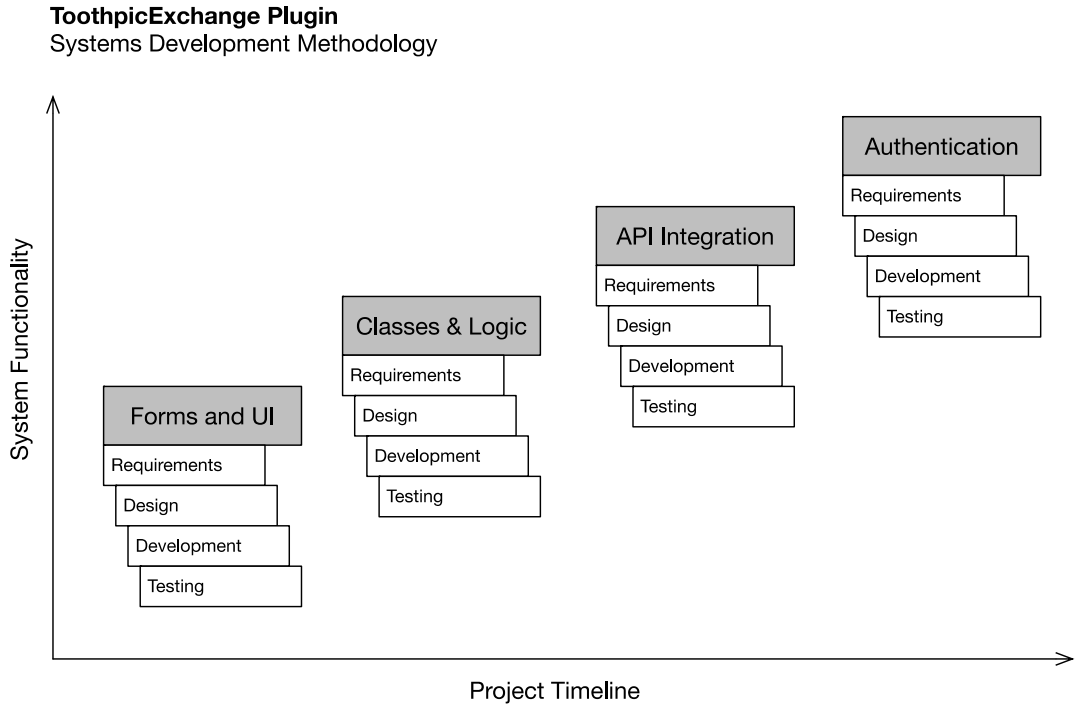


Figure D.1 – The Systems Development Methodology

Software Dependencies

Microsoft .NET version 4.5
C# version 7.2
RestSharp version 105.2.3 (RestSharp, 2018)
OpenDental 17.2 (OpenDental, 2017)

(Visual Studio 2017 or other IDE)
Json-Server 0.12.1 for test environment (Typicode, 2017)
DocFx.console version 2.23 for documentation (DocFx, 2018)

Coding Standards

Conforms to C# coding conventions (Microsoft, 2015)

Scenarios

These are to give the client an idea of how the system might be used in a real-world environment. They are not indicative of the system's final functionality.

Actors:

- Customer (user)
- Teledental Assessor
- Physical dental practice
- Toothpic
- OpenDental
- ToothpicExchange (System under Discussion)

Systems:

- Toothpic
- OpenDental
- ToothpicExchange

Scenario 1:

Purpose:	Scenario which describes the intake of patient information into a dental practice.
Individuals:	Jane, a typical user of Toothpic The teledental assessor The physical dental practice

Jane has not visited the dentist in over a year, however she is now experiencing pain in her mouth. She does not have the time to make an appointment, so she downloads Toothpic, creates an account and completes a case consultation. The case consists of her demographic

description, a summary of her general health, a description of the problem she is having and a set of six photographs of her teeth.

Jane's case is sent to a Toothpic teledental assessor and reviewed. The assessor marks up some of the photographs. The assessor notices that some of Jane's wisdom teeth are coming in and includes this in the consultation report. The assessor suggests that Jane pay a visit to a physical dentist who can further explore the problem and administer dental procedures. The assessor packages all of the case information and exchanges this with a dental practice on the Toothpic network which is in Jane's vicinity.

The practice loads Jane's data into their system and begins reviewing the information. They organise an appointment with Jane to have her wisdom teeth removed and the procedure is noted on Jane's record. This information is then synchronised back to Toothpic who can maintain a comprehensive history of Jane's dental health. This information will facilitate future clinical decision making and will allow Jane to keep track of her dental health remotely and independently in an efficient manner.

Scenario 2

Purpose:	Scenario which describes the extraction of patient information from the dental practice.
Individuals:	Marcus, a patient Marcus's physical dentist, who is on the Toothpic network

Marcus has been visiting his dentist regularly for a number of years now. His dentist is on the Toothpic network and suggests that Marcus start completing consultations via Toothpic. This saves time for Marcus.

Instead of Marcus signing up for Toothpic himself and having to enter the information manually – his dentist seamlessly exchanges the information with Toothpic and an account is set up for Marcus. Marcus is emailed a link from Toothpic to complete the signup process and is instructed how to complete regular consultations.

As a result, Toothpic now has a record of Marcus's previous dental procedures, his dental health history and any other pertinent health information. Marcus can keep track of his dental health through the Toothpic app which will enhance the service his physical dentist is already delivering.

Note:
These scenarios are indicative of how a system such as this could be fully integrated into the practice, however this would require many changes to Toothpic's internal systems and their API. The issue at hand only tackles the exchange of patient *demographic* information (i.e. name, date of birth, email etc.)

Use Cases:

This section contains a UML use-case diagram in Figure D.2 showing the scope and context of the system. Below this is the system's high-level use-cases which demonstrate the functionality to be achieved.

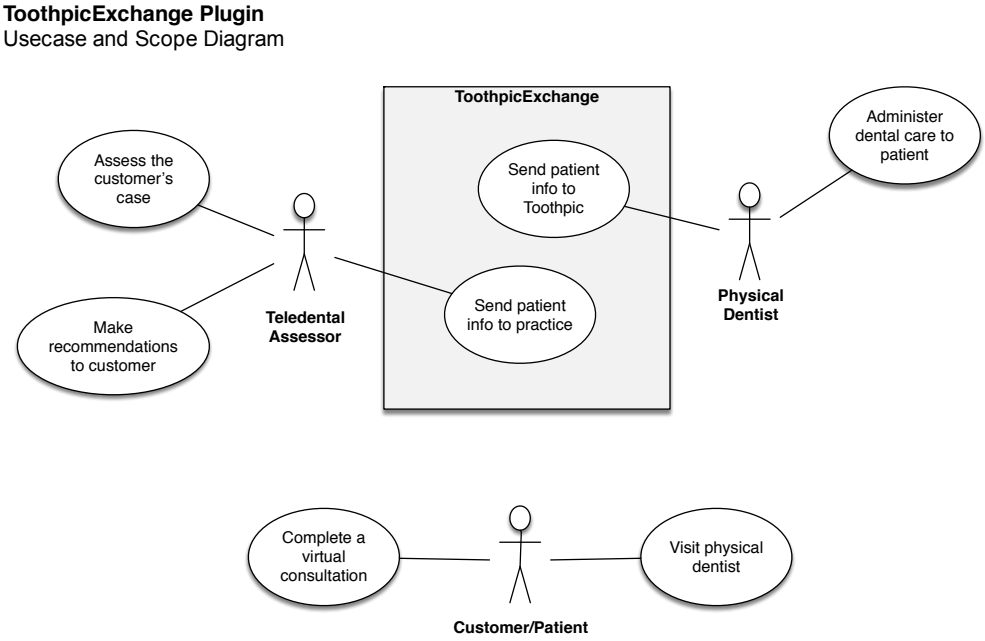


Figure D.2 – Use-case and Scope Diagram

Business Level Use-cases:

Use Case: **1 – Send patient information to the practice**
Scope: Toothpic
Level: Business
Primary Actor: Assessor/Toothpic

1. Assessor informs customer they should visit a dentist
2. Customer chooses a local practice on the Toothpic network
3. Toothpic informs the practice of new patient
4. Practice requests the customer's information from Toothpic
5. Toothpic deliver the customer's information
6. Practice delivers healthcare to customer
7. Practice sends Toothpic updated patient information

Steps 1 – 3 are outside the scope of this project.

Use Case: **2 – Send patient information to Toothpic**
Scope: OpenDental
Level: Business
Primary Actor: Dental Practice

1. Practice informs customer they should be set up with Toothpic

2. Practice sends patient information to Toothpic
3. Toothpic contacts customer to set up account

Step 3 is outside the scope of this project.

System Level Use-cases:

Use Case: **3 – Send patient information to the practice**
Scope: OpenDental
Level: System
Primary Actor: Dental Practice

1. Practice launches OpenDental
2. Practice launches plugin
3. Practice authenticates with username and password
4. Practice requests a list of users
5. Toothpic returns a list of users authorised for access
6. Practice selects user to import
7. Practice adds additional information about user
8. Practice saves the user into OpenDental

Use Case: **4 – Send patient information to the practice**
Scope: OpenDental
Level: System
Primary Actor: Dental Practice

1. Practice launches OpenDental
2. Practice selects a user (via their chart)
3. Practice launches plugin
4. Practice authenticates with username and password
5. Practice clicks “Export Patient”
6. Patient information is sent to and imported by Toothpic

Business Requirements

These are the requirements used to design the system functionality and are used in the validation phase of testing.

1. Must interface with Toothpic’s existing API.
2. Must interface with OpenDental’s Practice Management Software.
3. Must import patient demographic information presented from Toothpic’s API to OpenDental’s database.
4. Must export patient demographic information from OpenDental’s database to Toothpic’s API.
5. The OpenDental client must be deployable in a straightforward manner (i.e. a plugin that can be easily installed and set up in OpenDental).
6. The OpenDental client should be usable by non-technically advanced users.
7. The OpenDental client’s user interface should be simple and self-explanatory.

8. The OpenDental client's user interface should operate in a similar manner to OpenDental's other functionality.
9. Must securely connect and communicate with Toothpic API over the Internet through HTTPS.
10. Must follow Toothpic's authentication protocol for their API.
11. Must ensure the contents of the OpenDental client are not accessible by other plugins.

System Requirements

These are lower level requirements and are used in both the validation and verification phases of testing.

1. Should GET a list of users from Toothpic's API.
2. Should GET an individual user from Toothpic's API by their user ID.
3. Should POST a user to Toothpic's API.
4. The API calls should return the relevant C# object, not an array.
5. The API POST calls should be supplied a relevant C# object, not an array.
6. Should be able to map a ToothpicUser to an OpenDental Patient and vice versa.
7. Should fetch a patient from OpenDental by patient number, name or date of birth.
8. Entire functionality should be packaged into one executable.
9. Should include a configuration file for altering the API URL.
10. No functionality should be enabled until the user has successfully authenticated.
11. The homepage should contain an ImportUser button, an ExportPatient button and a button to log out.
12. The ImportUser button should open a new screen which contains a list of available users to import.
13. The ImportUser screen should contain a search box to filter the users.
14. The ImportUser screen should contain a scrollable list.
15. The ImportUser list should contain the user's name, date of birth, gender and email address.
16. The system should advise whether the patient import/export processes were successful or not.
17. The system should check that duplicate patients do not already exist when importing into OpenDental.
18. The system should query Toothpic's API to check duplicate patients do not exist when exporting to Toothpic's API.
19. The authentication tokens are to be stored in an authentication object held privately.
20. The API object should contain the authentication object privately.
21. The authenticated API object should be passed into each function requiring an API call.
22. The user objects should not be accessible outside the plugin.

Activity Diagrams

These UML activity diagrams show the dynamics of the system and illustrate the flow from one activity to another.

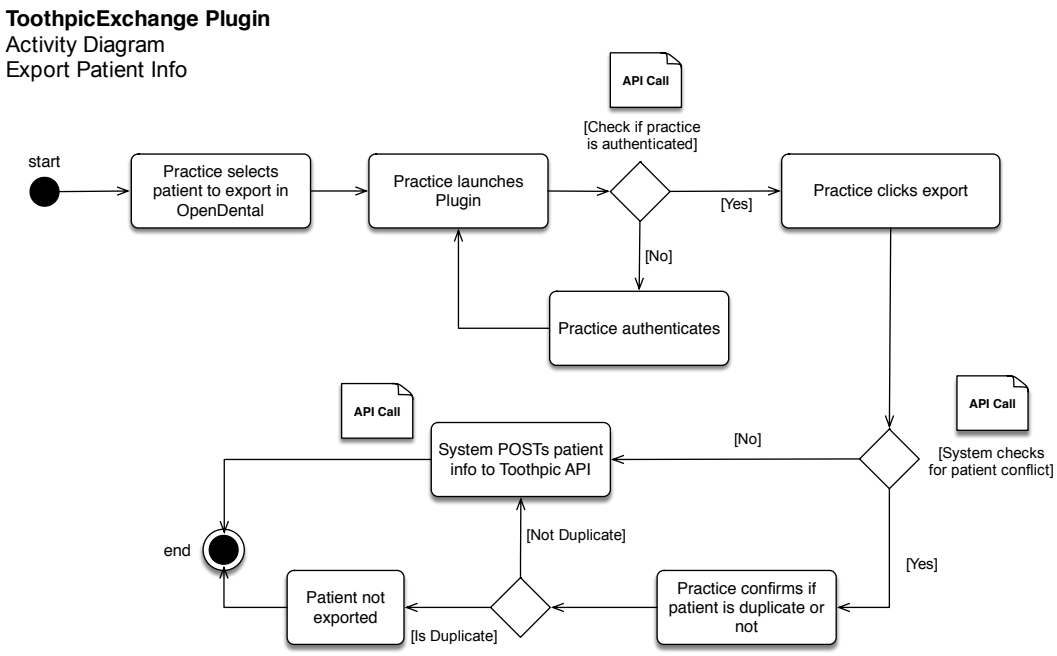


Figure D.3 – The Export Patient Activity Diagram

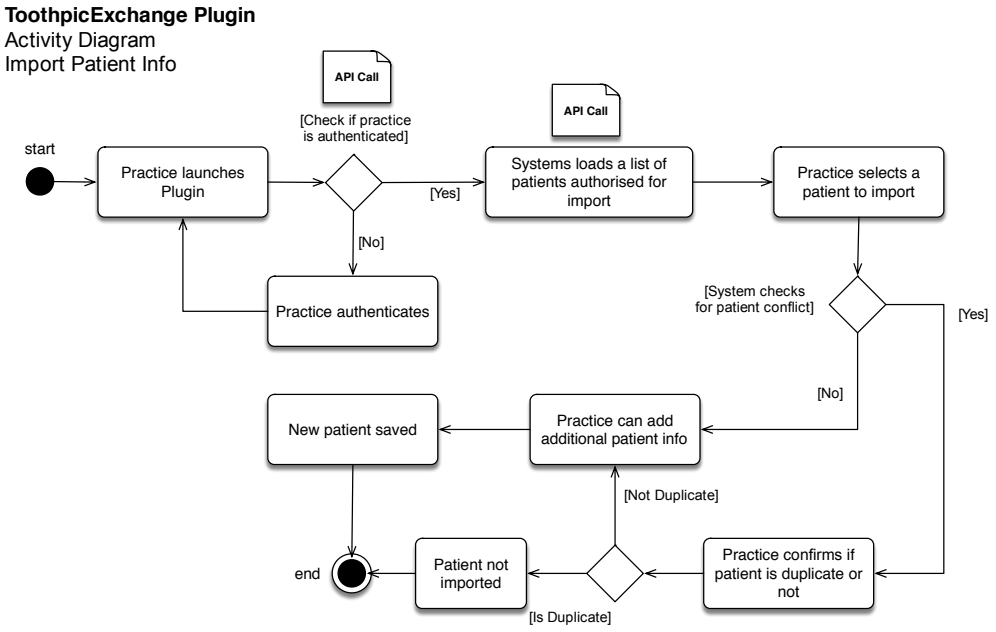


Figure D.4 – The Import Patient Activity Diagram

Class Diagrams

ToothpicExchange Plugin
Class Diagram

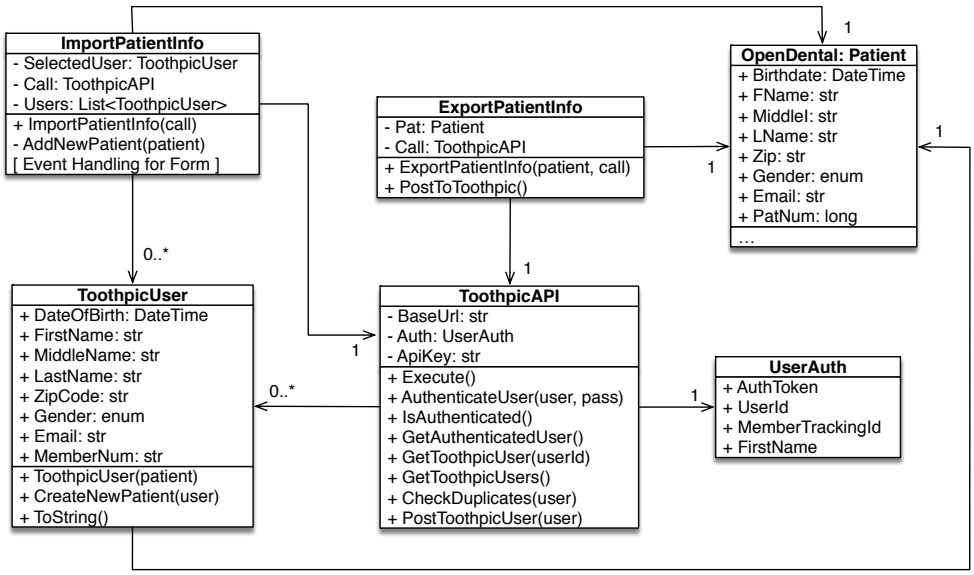


Figure D.5 – Class Diagram of the primary classes

Data Models

Figure D.6 illustrates the ToothpicUser vs OpenDental data models. Figure D.7 depicts a more horizontal view of Toothpic's existing model. Much of Toothpic's information is valuable but cannot be easily integrated into OpenDental without standardisation. Figure D.8 depicts a horizontal (summarised) view of OpenDental's model. Again, there is a huge amount of valuable information here but Toothpic have no means to ingest it (currently).

ToothpicExchange Plugin
High-Level Class Diagram

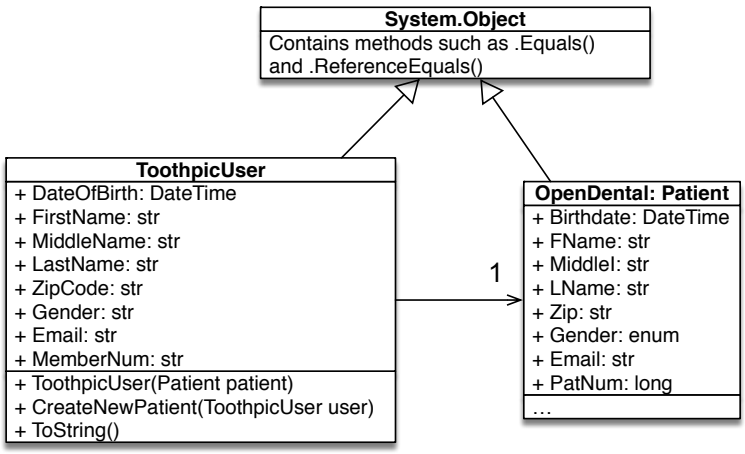


Figure D.6 – A High-Level Class Diagram

ToothpicExchange Plugin
The Toothpic Logical Data Model

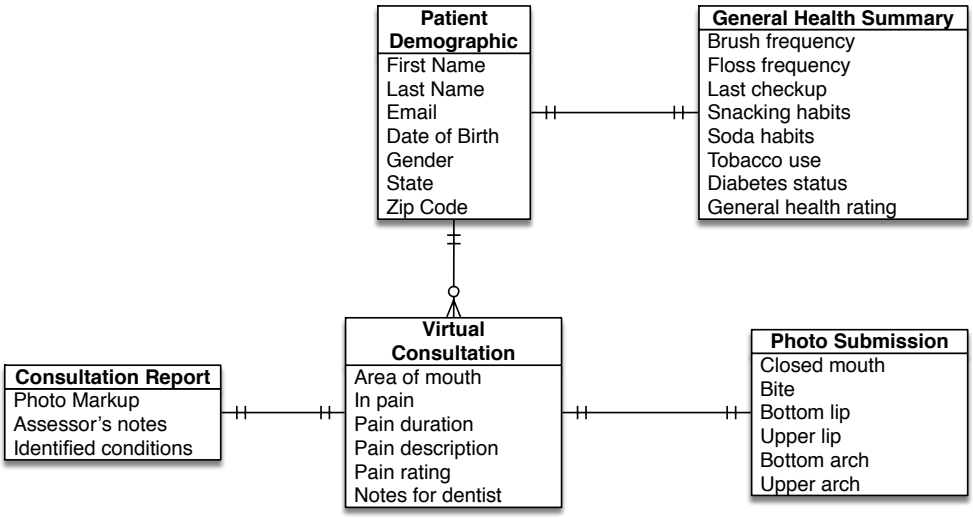


Figure D.7 – A Horizontal View of Toothpic's Model

ToothpicExchange Plugin
OpenDental Logical Data Model

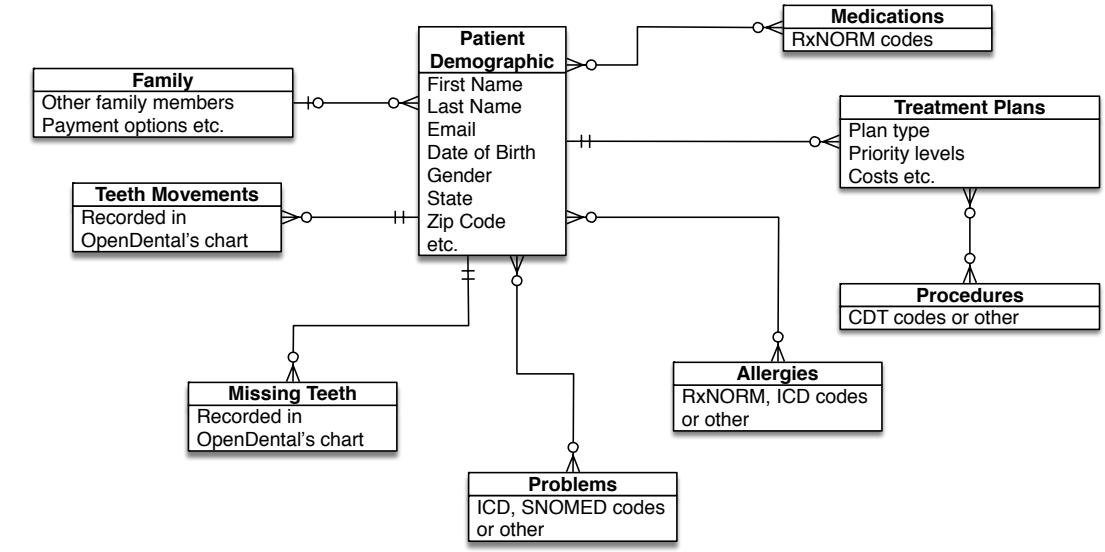


Figure D.8 – A Horizontal View of OpenDental's Model

Figure D.7 and Figure D.8 are entity relationship diagrams using the Crow's Foot notation. The small icons connecting objects indicate the multiplicity.

E. SCREENSHOTS

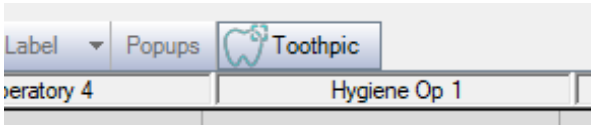


Figure E.1 – The toolbar button to launch the plugin.

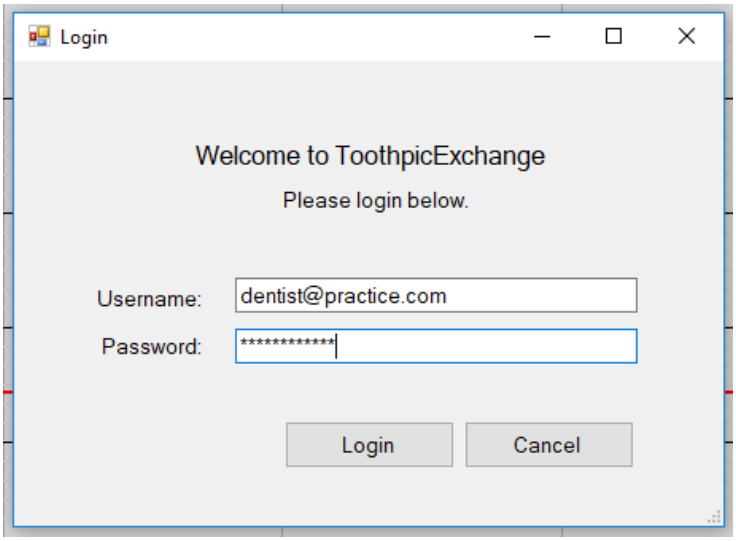


Figure E.2 – The login/authentication screen for the practice.

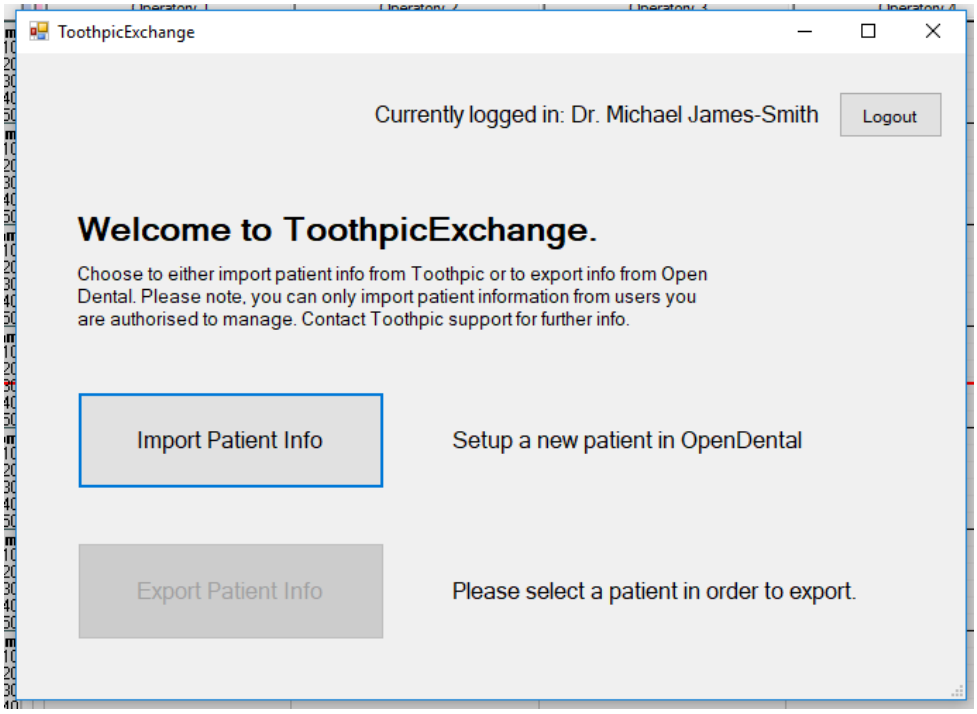


Figure E.3 – The homepage if the practice has not “selected” a patient in OpenDental.

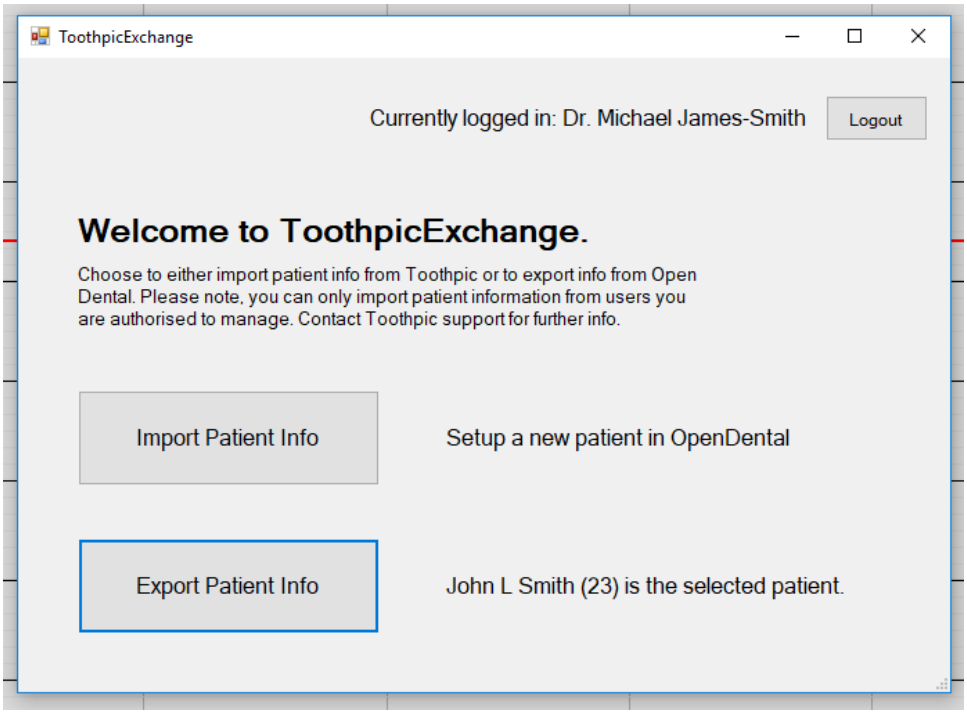


Figure E.4 – The homepage when the practice has “selected” the user in OpenDental.

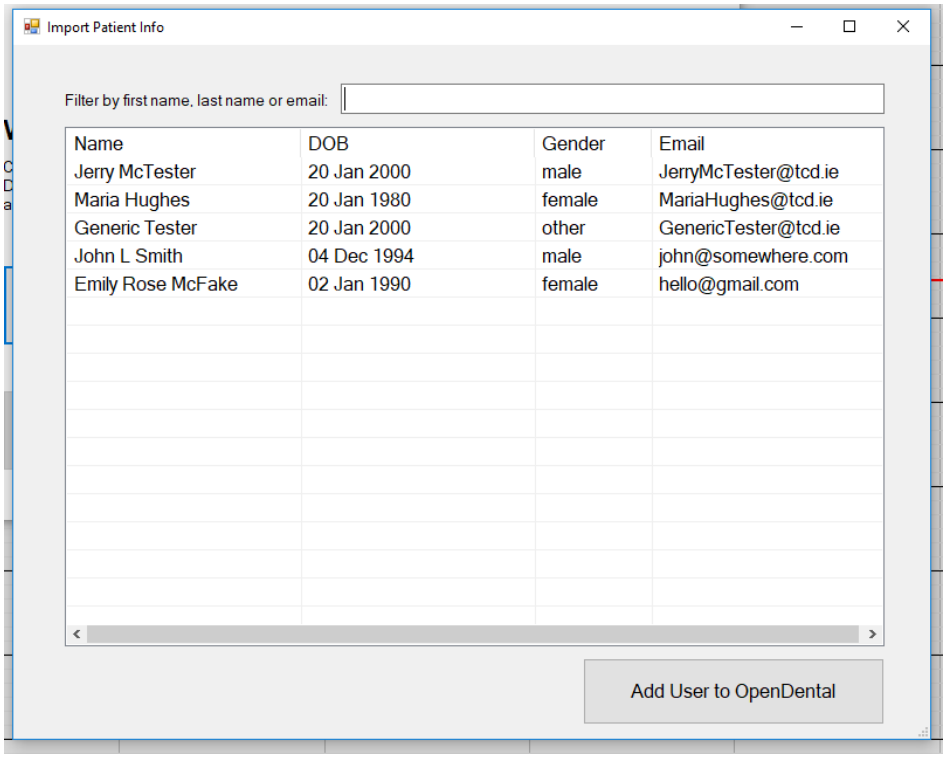


Figure E.5 – The import screen fetching a list of users from Toothpic’s API

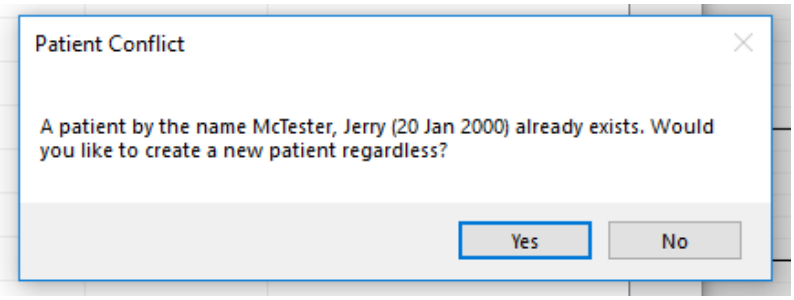


Figure E.6 – A patient conflict warning.

F. SOURCE CODE

Included is a sample of the ToothpicUser, the UserAuth and the ToothpicAPI classes. The full source code is available on the attached DVD.

ToothpicUser.cs:

```
1.  using System;
2.  using OpenDentBusiness;
3.  using RestSharp.Deserializers;
4.
5.  namespace ToothpicExchange
6.  {
7.      /// <summary>
8.      /// This class contains the user model as defined by Toothpic's API.
9.      /// It contains methods to convert a Toothpic User to an OpenDental Patient, an
    d vice versa.
10.     /// </summary>
11.     public class ToothpicUser
12.     {
13.         /// <summary>
14.         /// DateOfBirth mapped to Toothpic API as "dob".
15.         /// </summary>
16.         [DeserializeAs(Name = "dob")]
17.         public DateTime DateOfBirth { get; set; }
18.
19.         /// <summary>
20.         /// FirstName mapped to Toothpic API as "first_name".
21.         /// </summary>
22.         public string FirstName { get; set; }
23.
24.         /// <summary>
25.         /// MiddleName mapped to Toothpic API as "middle_name".
26.         /// </summary>
27.         public string MiddleName { get; set; }
28.
29.         /// <summary>
30.         /// LastName mapped to Toothpic API as "last_name".
31.         /// </summary>
32.         public string LastName { get; set; }
33.
34.         /// <summary>
35.         /// ZipCode mapped to Toothpic API as "zip_code".
36.         /// </summary>
37.         public string ZipCode { get; set; }
38.
39.         /// <summary>
40.         /// Gender, recorded as a string from the Toothpic API ("male", "female", "
    other") which maps to OpenDental's enum PatientGender (male, female, unknown).
41.         /// </summary>
42.         public string Gender { get; set; }
43.
44.         /// <summary>
45.         /// Email address from Toothpic API.
46.         /// </summary>
47.         public string Email { get; set; }
48.
49.         /// <summary>
50.         /// Member Number from Toothpic API, not currently used in OpenDental.
51.         /// </summary>
52.         public string MemberNumber { get; set; }
```

```
53.
54.         /// <summary>
55.         /// Contains all of the user attributes. ToothpicUser(Patient pat) is recom
    mended.
56.         /// </summary>
57.         public ToothpicUser()
58.         {
59.             DateOfBirth = new DateTime(0);
60.             FirstName = "";
61.             MiddleName = "";
62.             LastName = "";
63.             ZipCode = "";
64.             Gender = "other";
65.             Email = "";
66.             MemberNumber = "";
67.         }
68.
69.         /// <summary>
70.         /// Maps an OpenDental patient to a Toothpic user.
71.         /// </summary>
72.         /// <param name="Pat">An OpenDental patient</param>
73.         public ToothpicUser(Patient Pat)
74.         {
75.             DateOfBirth = Pat.Birthdate;
76.             FirstName = Pat.FName;
77.             MiddleName = Pat.MiddleI;
78.             LastName = Pat.LName;
79.             ZipCode = Pat.Zip;
80.             Gender = (Pat.Gender == PatientGender.Male ? "male" : (Pat.Gender == Pa
    tientGender.Female ? "female" : "other"));
81.             Email = Pat.Email;
82.             MemberNumber = null;
83.         }
84.
85.         /// <summary>
86.         /// Modified from OpenDental's Patients.CreateNewPatient
87.         /// !important: assumes the user information has already been validated.
88.         /// It creates a new patient from the current instance of ToothpicUser and
    adds them to the database.
89.         /// </summary>
90.         /// <remarks>
91.         /// This is a complex method which currently conforms to OpenDental 17.2 bu
    t should be carefully monitored.
92.         /// This method is used in ImportPatientInfo.AddNewPatient
93.         /// It does the majority of the mapping from a ToothpicUser to an OpenDenta
    l patient.
94.         /// </remarks>
95.         /// <returns>A new patient object</returns>
96.         public Patient CreateNewPatient()
97.         {
98.             // makes a record in the log where this patient originated from
99.             var securityLogMsg = "Patient created from ToothpicExchange.";
100.
101.                 var patient = new Patient();
102.
103.                 // PT name must be longer than 1 character
104.                 // These 2 blocks simply format the PT's name
105.                 if (LastName.Length > 1)
106.                     patient.LName = LastName.Substring(0, 1).ToUpper() + LastNam
    e.Substring(1);
107.                 if (FirstName.Length > 1)
108.                     patient.FName = FirstName.Substring(0, 1).ToUpper() + FirstN
    ame.Substring(1);
109.                 if (MiddleName != null)
110.                 {
111.                     if (MiddleName.Length > 0)
```

```
                            patient.MiddleI = MiddleName.Substring(0, 1).ToUpper() + MiddleName.Substring(1);
                    }

                    // This is the mapping from ToothpicUser to Patient
                    patient.Birthdate = DateOfBirth;
                    // Assumes the user is not deceased
                    patient.PatStatus = PatientStatus.Patient;
                    patient.BillingType = PrefC.GetLong(PrefName.PracticeDefaultBillType);
                    patient.Gender = (Gender == "male" ? PatientGender.Male : (Gender == "female" ? PatientGender.Female : PatientGender.Unknown));
                    patient.Email = Email;
                    patient.Zip = ZipCode;

                    // Inserts the PT into the DB
                    // "false" indicates we do not currently have a Primary Key for the PT
                    Patients.Insert(patient, false);
                    SecurityLogs.MakeLogEntry(Permissions.PatientCreate, patient.PatNum, securityLogMsg, LogSources.None);

                    // Final clean up methods
                    var custRef = new CustReference();
                    custRef.PatNum = patient.PatNum;
                    CustReferences.Insert(custRef);
                    var PatOld = patient.Copy();
                    patient.Guarantor = patient.PatNum;
                    Patients.Update(patient, PatOld);

                    // Returns the newly created PT
                    return patient;

                }

                /// <summary>
                /// Simply formats ToothpicUser as a string.
                /// </summary>
                /// <returns>A string with each item separated by a line break.</returns>
                override public string ToString()
                {
                    return string.Format("{0}\n{1}\n{2}\n{3}\n{4}\n{5}\n{6}", FirstName, MiddleName, LastName, ZipCode, Gender, Email, MemberNumber);
                }
            }
        }
```

UserAuth.cs:

```
1.  using System;
2.
3.  namespace ToothpicExchange
4.  {
5.      /// <summary>
6.      /// This class contains the Auth object which is used for authenticating API ca
    lls.
7.      /// </summary>
8.      class UserAuth
9.      {
10.         /// <summary>
11.         /// The Auth token used for API calls. Returned from API once user is authe
    nticated.
12.         /// </summary>
13.         public string AuthToken { get; set; }
14.
15.         /// <summary>
16.         /// The authenticated user's ID. Returned from API once user is authenticat
    ed
17.         /// </summary>
18.         public string UserId { get; set; }
19.
20.         /// <summary>
21.         /// The member_tracking_id returned from API once user is authenticated.
22.         /// </summary>
23.         public string MemberTrackingId { get; set; }
24.
25.         public UserAuth()
26.         {
27.             AuthToken = "";
28.             UserId = "";
29.             MemberTrackingId = "";
30.         }
31.
32.         public override string ToString()
33.         {
34.             return string.Format("AuthToken: {0}\nUserId: {1}\nMemberTrackingId: {2
    }", AuthToken, UserId, MemberTrackingId);
35.         }
36.
37.     }
38. }
```

ToothpicAPI.cs:

```
1.  using System;
2.  using System.Collections.Generic;
3.  using RestSharp;
4.
5.  namespace ToothpicExchange
6.  {
7.      /// <summary>
8.      /// ToothpicAPI is the class responsible for handling Toothpic's REST API. All
    of its attributed are private. It holds the API's URL and the UserAuth object.
9.      /// </summary>
10.     public class ToothpicAPI
11.     {
12.         /// <summary>
13.         /// This is the API's BaseURL. It is changed in the APISettings.settings co
    nfig file before build.
14.         /// </summary>
15.         private string BaseUrl = APISettings.Default.ToothpicAPIBaseURI;
16.
17.         /// <summary>
18.         /// This is the API's authentication object.
19.         /// </summary>
20.         private UserAuth Auth;
21.
22.         /// <summary>
23.         /// The API key for this client (i.e Open Dental).
24.         /// </summary>
25.         private string apiKey = "1234";
26.
27.         /// <summary>
28.         /// The API endpoint for authentication.
29.         /// </summary>
30.         private string authEndpoint = "/auth/";
31.
32.         /// <summary>
33.         /// The API endpoint for users.
34.         /// </summary>
35.         private string userEndpoint = "/users/";
36.
37.         /// <summary>
38.         /// Contains the BaseUrl, Authentication object and an API key.
39.         /// </summary>
40.         public ToothpicAPI()
41.         {
42.             BaseUrl = APISettings.Default.ToothpicAPIBaseURI;
43.             Auth = null;
44.             apiKey = "";
45.         }
46.
47.         /// <summary>
48.         /// This method executes a REST request.
49.         /// It is from the RestSharp class http://restsharp.org
50.         /// See RestSharp documentation for more info.
51.         /// It makes a request and adds the necessary headers for authentication et
    c.
52.         /// </summary>
53.         /// <typeparam name="T">Can be any type, normally ToothpicUser</typeparam>

54.         /// <param name="request">Any RestSharp request object</param>
55.         /// <returns>Returns the object specified, normally ToothpicUser.</returns>

56.         public T Execute<T>(RestRequest request) where T : new()
57.         {
58.             // Sets up a new client.
```

```
59.             var client = new RestClient();
60.             client.BaseUrl = new System.Uri(BaseUrl);
61.
62.             // Adds authentication headers to the request.
63.             request.AddHeader("x-signature", Auth.AuthToken);
64.             request.AddHeader("x-auth-token", Auth.AuthToken);
65.             request.AddHeader("x-api-key", apiKey);
66.
67.             // Makes the request.
68.             var response = client.Execute<T>(request);
69.
70.             // Throws an exception if there are any errors.
71.             if (response.ErrorException != null)
72.             {
73.                 const string message = "Error retrieving API response.";
74.                 ApplicationException toothpicAPIException = new ApplicationExceptio
    n(message, response.ErrorException);
75.                 throw toothpicAPIException;
76.             }
77.
78.             // Returns the relevant object.
79.             return response.Data;
80.         }
81.
82.         /// <summary>
83.         /// Calls the API to authenticate the user. Returns a UserAuth object if su
    ccessful, otherwise returns null.
84.         /// </summary>
85.         /// <param name="username">The user's username (or email)</param>
86.         /// <param name="password">The user's password</param>
87.         /// <returns>Returns itself if authentication was successful, otherwise nul
    l.</returns>
88.         public ToothpicAPI AuthenticateUser(string username, string password)
89.         {
90.             // The auth_type corresponding to Toothpic API
91.             var authType = "practice_user";
92.
93.             // Sets up a new POST request
94.             var request = new RestRequest(Method.POST);
95.
96.             // This is the API endpoint, can easily be changed
97.             request.Resource = authEndpoint;
98.
99.             // Specifies content-type: application/json
100.                request.RequestFormat = DataFormat.Json;
101.
102.                // The JSON body
103.                request.AddBody(new
104.                {
105.                    email = username,
106.                    auth_type = authType,
107.                    password = password
108.                });
109.
110.                /*
111.                 *  // DEBUG:
112.                 *
113.                 *  Auth = new UserAuth
114.                 *  {
115.                 *      AuthToken = "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzY2h
    lbWEiOiJ0b290aHBpYyIsInJvbGVzIjpbInJlZ3VsYXIiXSwidHlwIjoiand0IiwiaXNzIjoiR3JpYW5naH
    JhZkRlRmlhY2FpbCIsImlhdCI6MTUxOTkzNDIyOCwicmVzb3VyY2UiOiJVc2VyIn0.q5nEBhAYLB5QpaGYe
    rbJBIuEQYeuTHs1mXt13Jm-DJg",
116.                 *      UserId = "1111",
117.                 *      MemberTrackingId = "aa0fc553181fe5ee66c0973c5868cc16c3035
    955"
```

```
118.                     *  };
119.                     *
120.                     * if (username == "hello")
121.                     *    return this;
122.                     * else
123.                     *    return null;
124.                     *
125.                     */
126.
127.                    // Attempts to POST and returns object as necessary
128.                    try
129.                    {
130.                        Auth = Execute<UserAuth>(request);
131.                        return this;
132.                    }
133.                    catch (ApplicationException)
134.                    {
135.                        return null;
136.                    }
137.
138.                }
139.
140.                /// <summary>
141.                /// A boolean indicating whether the API object is authenticated or
    not.
142.                /// </summary>
143.                /// <returns></returns>
144.                public bool IsAuthenticated()
145.                {
146.                    return (Auth != null);
147.                }
148.
149.                /// <summary>
150.                /// Fetches the current authorised user (by their user ID). Primaril
    y used in LaunchToothpicExchange.cs.
151.                /// </summary>
152.                /// <returns>The currently authenticated user as a ToothpicUser obje
    ct, otherwise null.</returns>
153.                public ToothpicUser GetAuthenticatedUser()
154.                {
155.                    if (Auth != null)
156.                        return GetToothpicUser(Auth.UserId);
157.                    else
158.                        return null;
159.                }
160.
161.                /// <summary>
162.                /// This method fetches a user from the Toothpic API by a specified
    User ID.
163.                /// </summary>
164.                /// <param name="userId">A Toothpic User ID.</param>
165.                /// <returns>The specified ToothpicUser.</returns>
166.                public ToothpicUser GetToothpicUser(string userId)
167.                {
168.                    // Sets up a new GET request
169.                    var request = new RestRequest();
170.
171.                    // this is the API endpoint, can easily be changed
172.                    request.Resource = userEndpoint + userId;
173.
174.                    // Attempts to GET and returns null if necessary
175.                    try
176.                    {
177.                        return Execute<ToothpicUser>(request);
178.                    }
179.                    catch (ApplicationException)
```

```
180.                     {
181.                         return null;
182.                     }
183.                 }
184.
185.                 /// <summary>
186.                 /// This method fetches a list of Toothpic users from the API.
187.                 /// </summary>
188.                 /// <returns>A List of Toothpic Users, null otherwise.</returns>
189.                 public List<ToothpicUser> GetToothpicUsers()
190.                 {
191.                     // Sets up a new GET request
192.                     var request = new RestRequest();
193.
194.                     // this is the API endpoint, can easily be changed
195.                     request.Resource = userEndpoint;
196.
197.                     // if necessary, request.RootElement maybe required, see RestSha
    rp documentation for more info
198.
199.                     // Attempts to GET and returns null if necessary
200.                     try
201.                     {
202.                         return Execute<List<ToothpicUser>>(request);
203.                     }
204.                     catch (ApplicationException)
205.                     {
206.                         return null;
207.                     }
208.                 }
209.
210.                 /// <summary>
211.                 /// Fetches a list of users and checks for duplictes.
212.                 /// </summary>
213.                 /// <param name="searchUser">The user to be checked.</param>
214.                 /// <returns></returns>
215.                 public bool CheckDuplicateUser(ToothpicUser searchUser)
216.                 {
217.                     var users = GetToothpicUsers();
218.
219.                     if(users != null)
220.                     {
221.                         foreach (var user in users)
222.                         {
223.                             if (searchUser.Equals(user))
224.                                 return true;
225.                         }
226.                     }
227.
228.                     return false;
229.                 }
230.
231.                 /// <summary>
232.                 /// This method POSTs a Toothpic User to the API.
233.                 /// The JSON formatting is defined within the method.
234.                 /// </summary>
235.                 /// <param name="user">Any ToothpicUser.</param>
236.                 /// <returns>A boolean whether it POSTed successfully or not</return
    s>
237.                 public bool PostToothpicUser(ToothpicUser user)
238.                 {
239.                     // Sets up a new POST request
240.                     var request = new RestRequest(Method.POST);
241.
242.                     // This is the API endpoint, can easily be changed
243.                     request.Resource = userEndpoint;
```

```
244.
245.                     // Specifies content-type: application/json
246.                     request.RequestFormat = DataFormat.Json;
247.
248.                     // The JSON body
249.                     request.AddBody(new
250.                     {
251.                         dob = user.DateOfBirth.ToString("yyyy-MM-dd"),
252.                         email = user.Email,
253.                         first_name = user.FirstName,
254.                         middle_name = user.MiddleName,
255.                         last_name = user.LastName,
256.                         gender = user.Gender,
257.                         zip_code = user.ZipCode
258.                     });
259.
260.                     // Attempts to POST and returns a boolean as necessary
261.                     try
262.                     {
263.                         var response = Execute<ToothpicUser>(request);
264.                         return true;
265.                     }
266.                     catch (ApplicationException)
267.                     {
268.                         return false;
269.                     }
270.                 }
271.             }
272.         }
```

G. TESTING DOCUMENTATION

Verification Testing

Unit Testing

Table G-1 outlines the different types of unit tests applied to the software. Unit tests were conducted at the end of each increment to ensure the software functioned correctly.

Table G-1 – Different Types of Unit Testing

Test Type	Description
User Input	The forms requiring user input were tested to ensure they could handle wrong input.
Event Handling	All forms were tested to ensure the buttons and actions lead to the correct events being triggered.
Data Validation	Any data being imported/exported was validated to be in the correct format and that duplicates were flagged by the system.
Mapping	Tested to ensure the system mapped the right elements to each other and that it worked with missing data.
API Errors	Tested under different scenarios that the API was not accessible or if there were errors (e.g. no Internet connectivity)
Authentication Errors	Tested the system responded correctly to cases where the user was or was not authenticated properly. Tested unauthorised information was not exposed.
Null Reference Errors	Tested the system did not throw any exceptions for null objects.

Validation Testing

Functionality Testing

With the system and business requirements in hand, the software was tested to satisfy each requirement. The system checked every requirement off the list.

Usability Testing

Although the traditional method of usability testing was not possible, the system was tested so that it responded to common usability requirements. E.g. using the tab key to jump through forms, using the Escape key, or being able to press the form close button without crashing the system.

H. REVIEW OF E-HEALTH STANDARDS

HL7 Version 2

Background:

HL7 (Health Level 7) is a standards-developing organisation. Their standards are used for transfer of clinical or administrative data in many healthcare contexts.

Their version 2 standard was developed in 1989 and has been in use and maintained since then. It is popular in many healthcare organisations as a messaging specification.

The standard defines types of messages that can be used to transfer data. The messages are quite rudimentary in their construction, although their interpretation can be complex.

Despite the fact that HL7 define the actual structure of the message, there is no real consensus on what information actually goes inside the message, or how it should be represented. HL7 defines where the data should go, and roughly what encoding/format it should be in. However, many of the fields can be interpreted as ambiguous. Further to this, it is not a binding standard, as certain elements of the form can be omitted, and proprietary systems can even write in their own fields (these are called z-segments).

For example, when OpenDental ingests a HL7 message, it only reads the fields it has been programmed to read. Therefore, important information can be left by the wayside.

In summary, a specification such as HL7 v2 does not truly achieve interoperability. It can allow systems to communicate in a given way, but it does not guarantee understanding between the two systems.

Specification:

A HL7 message is just a text document that uses lines and delimiters to structure data. The standard defines different types of messages, each of them has a three-letter code. For example, an ADT message is a patient demographic message.

A message is made up of segments. Each new line represents a new segment. A particular message type has a specification for what segments it can contain. Each message segment also begins with a three-letter code (different from the message code). For example, the message header segment uses the code MSH.

A segment is made up of fields. The order of the fields is important as they do not have identifiers. The pipe character “|” is used as a delimiter between fields, and the caret character “^” is used as a delimiter to separate components within the field. For example, an “address” field could contain a number of components such as house number, street, county, etc.

Types of Messages:

The most relevant to Toothpic would be the ADT (Patient Demographics Message) and the PPR (Patient Problem Message).

The ADT message contains all of the patient demographics; however, it can also include information such as patient allergies, procedures that have taken place, and various observations on the patient (e.g. blood pressure, or various conditions). If Toothpic had a means to ingest this information, then they could gain a wider perspective on the patient's dental health.

The PPR message is a shorter message used to update a patient's record when they experience a new problem.

The majority of HL7 messages use some kind of systematised nomenclature or a coding system to represent medical and dental concepts. For example, SNOMED-CT is a system of standardised clinical terminology. A condition such as diabetes would be represented by a specific code, which eliminates ambiguity when exchanging patient information. The American Dental Association also manage CDT (Current Dental Terminology) to represent a number of dental concepts. Further detail on coding systems is available on page H-6.

Relevance in OpenDental:

OpenDental supports the HL7 standard, but not entirely. It natively supports a variety of HL7 v2.6 messages (including ADT and PPR), however additional specifications would need to be configured in the software if required.

OpenDental also supports a variety of coding systems, however there is no standard. Coding systems and standards are configured on a practice-by-practice basis.

Relevance to Toothpic:

Theoretically, Toothpic could partner with a practice and use HL7 v2 to exchange patient information. However, it would require configuration both within the practice and within Toothpic to ensure that both systems were speaking the same language.

Example ADT Message from OpenDental:

```
MSH|^~\&|^OtherSoftware.OIDroot^|OtherSoftware|^2.16.840.1.113883.3.4337.14
86.####^HL7
|OpenDental|20141015103243||ADT^A04^ADT_A01|b72247f821034398923d6b9834b0f16
8|P |2.6||||AL

EVN||20141015103243||01

PID|1|12345|12345^5^M11^&2.16.840.1.113883.3.4337.1486.####.2&^PI
~98765^1^M10^&OtherSoftware.PatientOID&^PI|98765|Smith^John^L^^Mr.||1994120
4|M||2106-3^White^CDCREC~2186-5^^CDCREC|421 N Main St^Apt
7^Salem^OR^97302^^^^^^^^^ ^^^^^^Emergency Contact: Jane
```

```
Smith\.br\Relationship: Wife\.br\Phone: (503)555-1234
||^PRN^PH^^^503^5551234~^PRN^Internet^john@somewhere.com
~^PRN^CP^^^503^5556789|^WPN^PH^^^503^3635432||M|||123456789

PV1|1|O|Clinic
1^^^^^C||||2.16.840.1.113883.3.4337.1486.####.3.123^Abbott^Sarah^L^^DrAbbot
t
~OtherSoftware.ProviderOID.987^Abbott^Sarah^L^^DrAbbott||||Oregon State
University ^^^^^S|||||||1

GT1|1|56789^2^M11^&2.16.840.1.113883.3.4337.1486.####.2&^PI
~54321^5^M10^&OtherSoftware.PatientOID&^PI|Smith^Jane^W^^Mrs.||421 N Main
St ^Apt
7^Salem^OR^97302|^PRN^PH^^^503^5551234~^PRN^Internet^jane@somewhere.com
~^PRN^CP^^^503^5556788|^WPN^PH^^^503^3631234|19941221|F|||987654321

OBX|1||745678^Albuterol Metered Dose Inhaler^RXNORM|||||||R

OBX|2||541349^^RXNORM|||||||R

AL1|1|DA|1191^Aspirin^RXNORM

 AL1|2|DA|70618^^RXNORM

PR1|1||D1351^Sealant per
tooth^CD2^^^^2014^^Sealant||20141015103243||||||||||10
|||1234^^OtherSoftware.ProcedureOID

PR1|2||D0150^^CD2^^^^2014||20141015103243||||||||||

PR1|3||D2394^^CD2^^^^2014||20141015103243||||||||||3&MODL
```

Explanation:

The three letter codes in bold represent the start of a segment. A line break represents the end of a segment. The pipe "|" is used as a field separator and the caret "^" to delineate "components" within a field.

MSH = Message Header
EVN = Event
PID = Patient ID
PV1 = Patient Visit (summarises details of the patient's visit to the healthcare facility, such as the healthcare provider)
GT1 = Guarantor (the patient's guarantor)
OBX = Observation (in this example, the observations are two medications the patient is taking)
AL1 = Allergies (in this case, DA = Drug Allergy)
PR1 = Procedures (using D codes from ADA's CDT)

This message uses RxNORM codes for pharmaceuticals, and D codes from ADA's CDT

HL7 FHIR

Background:

HL7 (Health Level 7) is a standards-developing organisation. Their standards are used for transfer of clinical or administrative data in many healthcare contexts.

FHIR (Fast Healthcare Interoperability Resources; pronounced “FIRE”) was initially published in 2014. It is still in the draft phase but has seen usage across a wide variety of healthcare scenarios.

FHIR is a modern RESTful standard that operates over HTTP. As with any RESTful model, the foundation of the model is based on resources. FHIR have designed a set of resources, and many of them will continue to be developed as the standard remains in the draft phase.

Examples of resources include: patients, practitioners, medications, observations etc.

Where FHIR begins to address the issue of interoperability is through their profile model. FHIR profiles define the semantics of the information being transferred.

Consider the example of blood pressure, which is a frequently cited problem in interoperability studies. Blood pressure can be recorded and presented in many different ways. If two organisations are exchanging blood pressure readings, and they measure them using different methods, then the information is useless. In order to address this, a profile could be sent alongside the blood pressure reading. The profile informs the receiving organisation which specification the reading is conforming to, and how it should be interpreted. Thus, even if the two systems do not natively record the value in the same way, at least they can interpret the variations.

This is a major advancement over HL7 v2, because it shows that the specification understands not all systems speak the same language.

Notwithstanding, it is still pertinent to remember that FHIR is a means to address the *transmission* of information between eHealth systems. There are many other ways to define how information should actually be structured and recorded. Not all systems should be modelled to emulate the resources and the profiles which FHIR have developed themselves. However, the beauty of the standard is that each system can implement their own version of FHIR, and if the profiles are correctly configured, a system can still communicate with other systems that do not necessarily use the same profiles.

Specification:

The transmission specification is a RESTful API that can deliver JSON or XML response. The data models are defined by the resource models developed by HL7. The profiles concern the semantics of the information and are also produced by HL7, however custom profiles can be used.

The FHIR website (http://build.fhir.org) contains a large amount of user-friendly information regarding the standard.

SMART

Beyond HL7 FHIR, there is an additional development called SMART (Substitutable Medical Apps, Reusable Technology) which is an effort to standardise medical apps that can plug and play across different eHealth systems. Currently, there is a lot of activity surrounding SMART on FHIR, which is a movement to create an ecosystem of interoperable apps that operate within and across varying clinical health systems.

Relevance in OpenDental:

Currently, OpenDental only offer very limited FHIR functionality. It is mostly restricted to appointment-based and administrative information (as opposed to clinical information). However, as the standard gains tractions, hopefully OpenDental will improve their support.

Relevance to Toothpic:

Toothpic should look into FHIR, and potentially SMART on FHIR to see how they could build an interoperable service. Theoretically, Toothpic could not be limited to an app on the mobile consumer app stores, but could also be an “app” on a “healthcare app store”. Dentists could plug and play Toothpic’s functionality directly into their own Practice Management System if it supported SMART on FHIR.

In fact, during the course of this project, Apple released a significant update to their native Health app which implemented HL7 FHIR. It allows users to access real-time, meaningful health information supplied directly from a health institution (Apple, 2018).

The caveat with HL7 FHIR and with SMART is that they are very much still in the development phase. When HL7 were developing their version 3 messaging standard, many organisations invested large amounts of money to migrate their version 2 systems to version 3. The standard proved to be a flop and is rarely used today. Thus, it is appropriate to take recommendations with a grain of salt, especially for standards that have yet to mature.

Coding Systems and Nomenclature

Coding systems were initially developed for statistical and financial applications. However, they are often used for information transfer between healthcare institutions. They are simple and efficient means to convey standardised procedures, medications, conditions, symptoms, observations etc.

There are different types of coding systems for different functions. The four types of systems utilised in OpenDental are:

- Diagnostic codes
- Topographical codes
- Procedural codes
- Pharmaceutical codes

The usage of codes is not exclusive to OpenDental, in fact many of the popular transmission standards (HL7 v2, FHIR etc.) rely on coding systems to exchange information.

Diagnostic Codes:

ICD-9-CM, ICD-10 are the two main standards. ICD stands for “International Statistical Classification of Diseases and Related Health Problems” and is issued by the World Health Organisation (WHO). ICD-10 and ICD-9 are base standards, however many subsequent standards are developed on top of them. They generally vary from country to country. The codes are typically used to classify diseases, conditions and symptoms.

Topographical Codes:

SNOMED-CT (Systematised Nomenclature of Medicine-Clinical Terms) is the frontrunner. Topographical codes are typically used to define parts, locations, disorders, and conditions of the human body. SNOMED-CT is a very extensive system and is used across a wide range of environments.

Pharmaceutical Codes:

RxNORM is the major standard, covering all medications licensed in the US.

Procedural Codes:

CDT (Current Dental Terminology) is issued by the American Dental Association. These codes are prepended with a “D” and represent dental procedures. They are used to describe procedures and are typically used for insurance and financial purposes.

LOINC (Logical Observation Identifiers Names and Codes) is another coding system but is not typically utilised in OpenDental.

OpenEHR

Background

OpenEHR (pronounced “OpenAir”, EHR = Electronic Health Record) is a consortium of healthcare organisations and experts working to define a set of interoperable domain driven standards.

Their approach is on the semantic layer, and all of their models are technology-agnostic. This means their data models and archetypes can be utilised on any platform regardless of the technology.

The programme is totally open, and is an entirely community based effort. They operate a Clinical Knowledge Manager (CKM) on their website which can be used to examine all of their archetypes and data models.

The CKM is also responsible for tracking the development of all models, so one can easily distinguish between models that have been published and approved, or models that are still in the draft phase.

In the draft phase, models can be adjusted as the community sees fit. Anyone is free to comment and suggest developments to any archetypes, which ensures the models have a wide scope of usage, and have been developed with many perspectives in mind.

Relevance to OpenDental

Currently, OpenDental does not either support or model their structures on the OpenEHR archetypes. However, a mapping could easily be developed utilising OpenEHR’s CKM tools. OpenEHR can present data models in any format (JSON, XML etc). Even though a mapping may sound cumbersome, in the long run it could be more beneficial to internally structure information in an OpenEHR model, but continue to map to third party PMSs for the time being.

Relevance to Toothpic

OpenEHR is still in a development phase, so Toothpic could easily get involved and begin to define either dentistry and/or teledentistry concepts for consideration. Not only would this be a good introduction to the eHealth interoperable community, but it could ensure Toothpic are using a reliable, futureproofed data model which is highly extensible and robust.

The OpenEHR website contains a huge number of resources on the models and archetypes they are developing (http://openehr.org).

I. GLOSSARY OF TERMS USED

HIPAA: Health Insurance Portability and Accountability Act

This is legislation in the US which governs (among many things) the transfer of electronic patient information in healthcare.

PMS: Practice Management System

Any software used internally in a dental practice for administration, clinical management etc. In this case, OpenDental is a PMS.

API: Application Programming Interface

This is a generic term for an external interface made available for interacting with other software.

UML: Universal Modelling Language

This is a language and a visual specification for modelling software design.

HTTP: HyperText Transfer Protocol

This is a protocol for data transfer. It is the foundation of the Internet. Most APIs on the Internet operate over HTTP(S).

RESTful: REpresentational State Transfer

This is an architectural model for web services which is based on requests, responses, and resource. Most modern web APIs are RESTful.

JSON: JavaScript Object Notation

A JSON document is just an array of key-value pairs.

Key-Value Pair

A combination of strings representing an identifier and a value (e.g. “key” : “value”)

URL: Universal Resource Locator

This is a unique address for locating a resource on the web.

C#

An object-oriented programming language initially developed by Microsoft. OpenDental and the plugin developed for this project were written in it.

.NET

A Microsoft software framework in Windows.

Visual Studio

An Integrated Development Environment (IDE) primarily used for Windows-based software development.

MySQL

An open-source relational database management system. Utilised by OpenDental but not required by the plugin developed for this project.

HL7

Health Level 7, a standards-developing organisation for healthcare IT.

HL7 v2

A messaging specification for exchange of electronic patient information in healthcare IT.

HL7 FHIR

A modern, RESTful specification for exchange of electronic patient information in healthcare IT.

PII: Personally Identifiable Information

Specified by HIPAA this is any information that uniquely identifies an individual (e.g. Social Security Number in the US)

PHI: Protected Health Information

Also specified by HIPAA this is any health information that can be connected back to an individual and disclose their identity.

REFERENCES

Apple, 2018. *Apple announces effortless solution bringing health records to iPhone.* [Online]
Available at: https://www.apple.com/newsroom/2018/01/apple-announces-effortless-solution-bringing-health-records-to-iPhone/
[Accessed 15 March 2018].

Cockburn, A., 2000. *Writing effective use cases.* 1st Edition ed. Harlow: Addison-Wesley.

DocFx, 2018. *DocFx.* [Online]
Available at: A Documentation Reference Tool for XML and Markdown
[Accessed 15 February 2018].

Free Software Foundation, 2007. *GNU General Public License.* [Online]
Available at: https://www.gnu.org/licenses/gpl-3.0.en.html
[Accessed 15 March 2018].

IETF, 2014. *RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format.* [Online]
Available at: https://tools.ietf.org/html/rfc7159
[Accessed 02 March 2018].

IETF, 2015. *RFC 7519: JSON Web Token (JWT).* [Online]
Available at: https://tools.ietf.org/html/rfc7519
[Accessed 2018 February 01].

Lee, M. S., 2005. *Programming language pragmatics.* 2nd Edition ed. Massachusetts: Morgan Kaufman.

Microsoft, 2015. *C# Coding Conventions.* [Online]
Available at: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions
[Accessed 05 January 2018].

Microsoft, 2015. *XML Documentation Comments for C#.* [Online]
Available at: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/xmldoc/xml-documentation-comments
[Accessed 01 February 2018].

OpenDental, 2017. *OpenDental Plugin Framework.* [Online]
Available at: http://opendental.com/manual/plugins.html
[Accessed 05 January 2018].

OpenDental, 2017. *OpenDental Source Code (17.2).* [Online]
Available at: http://www.opendental.com/manual/sourcecode.html
[Accessed 05 December 2017].

RestSharp, 2018. *Simple REST and HTTP Client for .NET.* [Online]
Available at: http://restsharp.org
[Accessed 20 February 2018].

Typicode, 2017. *json-server on Github (0.12.1).* [Online]
Available at: https://github.com/typicode/json-server
[Accessed 2018 February 02].

W3C, 2004. *Web Services Architecture.* [Online]
Available at: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest
[Accessed 10 March 2018].