**H48W 35 Computing: Software Development  
Graded Unit 2**

**Project Stage: Solution Planning**

**Solution Planning Report**

**Student name: Daria Vekic**

**Student number: 586661**

**Date: February / March 2023**

Table of Contents

[Overview 1](#_Toc129552927)

[Introduction 1](#_Toc129552928)

[Section 1 – Business Model 3](#_Toc129552929)

[The Use Case Model 3](#_Toc129552930)

[Top Level Use Case Diagram 3](#_Toc129552931)

[Fully Dressed Use Case Diagrams 4](#_Toc129552932)

[Use Case 2.1 – Creating a New Case 5](#_Toc129552933)

[Use Case 2.2 – Searching Cases 6](#_Toc129552934)

[Use Case 2.3 – View Caseload 9](#_Toc129552935)

[Prioritisation 12](#_Toc129552936)

[Fully Dressed Use Case Descriptions 13](#_Toc129552937)

[Use Case: Manage Cases (1) 16](#_Toc129552938)

[Use Case: Create New Case (1.1) 18](#_Toc129552939)

[Use Case: Enter Client Details (1.1.1) 19](#_Toc129552940)

[Use Case: Enter Solicitor Details (1.1.2) 20](#_Toc129552941)

[Use Case: Enter Case Details (1.1.3) 21](#_Toc129552942)

[Use Case: Assign Reference Number (1.1.6) 22](#_Toc129552943)

[Use Case: Search Cases (1.2) 23](#_Toc129552944)

[Use Case: Record Billable Activities (1.2.2) 24](#_Toc129552945)

[Use Case: Calculate Monies Owed (1.2.6) 25](#_Toc129552946)

[Use Case: Delete Case (1.2.3) 26](#_Toc129552947)

[Use Case: View Caseload (1.3) 27](#_Toc129552948)

[Use Case: Log In (2) 28](#_Toc129552949)

[Use Case: Generate Report (3) 29](#_Toc129552950)

[Data Validation 31](#_Toc129552951)

[Error Messages 38](#_Toc129552952)

[The Domain Model 39](#_Toc129552953)

[Natural Language Analysis 39](#_Toc129552954)

[Potential Entities 41](#_Toc129552955)

[Potential Behaviours 53](#_Toc129552956)

[Client Feedback 64](#_Toc129552957)

[Meeting Number 3 Agenda 65](#_Toc129552958)

[Meeting Number 3 Minutes 67](#_Toc129552959)

[Summary 70](#_Toc129552960)

[Updated Use Case Description: Log In (2) 71](#_Toc129552961)

[Recording Billable Activities – Natural Language Analysis 73](#_Toc129552962)

[Use Case: Add Staff Member (3) 80](#_Toc129552963)

[Conceptual Classes 81](#_Toc129552964)

[Relationships 84](#_Toc129552965)

[Attributes 85](#_Toc129552966)

[Final Conceptual Model 86](#_Toc129552967)

[Responsibilities and Collaborations 86](#_Toc129552968)

[Designing an Object‑Oriented System 88](#_Toc129552969)

[CRC Cards 88](#_Toc129552970)

[Data Binding Model 91](#_Toc129552971)

[Data Structures 92](#_Toc129552972)

[The Static Model 94](#_Toc129552973)

[Initial Class Diagram (1) 94](#_Toc129552974)

[Dependancy 95](#_Toc129552975)

[Package Structure 96](#_Toc129552976)

[Enumeration 96](#_Toc129552977)

[Person – Generalisation and Inheritance 98](#_Toc129552978)

[Initial Class Diagram (2) 100](#_Toc129552979)

[Case – Inheritance or Interface? 103](#_Toc129552980)

[An Abstract Person 103](#_Toc129552981)

[Cases, Clients, and Composition 104](#_Toc129552982)

[Billable Activities and Associations 105](#_Toc129552983)

[Refined Class Diagram 107](#_Toc129552984)

[The Dynamic Model 109](#_Toc129552985)

[Sequence Diagram – Log In 111](#_Toc129552986)

[Basic Log In 111](#_Toc129552987)

[Log In Validation 112](#_Toc129552988)

[New Employee Log In 113](#_Toc129552989)

[Sequence Diagram – Create New Case 115](#_Toc129552990)

[Sequence Diagram – Search Cases 117](#_Toc129552991)

[Activity Diagrams 119](#_Toc129552992)

[Activity Diagram – Log In 119](#_Toc129552993)

[Activity Diagram – Create New Case 122](#_Toc129552994)

[Activity Diagram – Search Cases 123](#_Toc129552995)

[Section 2 – View Model 125](#_Toc129552996)

[User Interface Design 125](#_Toc129552997)

[Wireframes 126](#_Toc129552998)

[Log In 127](#_Toc129552999)

[Home 128](#_Toc129553000)

[Searching 129](#_Toc129553001)

[Recording Billable Activities 130](#_Toc129553002)

[Add New Case 131](#_Toc129553003)

[View Caseload 132](#_Toc129553004)

[Our Staff 133](#_Toc129553005)

[Add New Employee 134](#_Toc129553006)

[User Analysis 135](#_Toc129553007)

[Style Guide 136](#_Toc129553008)

[User Interfaces 139](#_Toc129553009)

[Log In 139](#_Toc129553010)

[Home 140](#_Toc129553011)

[Add New Case 141](#_Toc129553012)

[Searching 142](#_Toc129553013)

[Recording Billable Activities 143](#_Toc129553014)

[View Caseload 144](#_Toc129553015)

[Our Staff 145](#_Toc129553016)

[Add New Employee 146](#_Toc129553017)

[Client Feedback 146](#_Toc129553018)

[Meeting Number 4 Agenda 147](#_Toc129553019)

[Meeting Number 4 Minutes 149](#_Toc129553020)

[Summary 151](#_Toc129553021)

[Data Binding Design 151](#_Toc129553022)

[Data Mapping – Log In 152](#_Toc129553023)

[Data Mapping – Add New Case (Case Details) 153](#_Toc129553024)

[Data Mapping – Add New Case (Client Details) 154](#_Toc129553025)

[Data Mapping – Add New Case (Solicitor Details) 155](#_Toc129553026)

[Data Mapping – Add New Case 156](#_Toc129553027)

[Data Mapping – Recording Billable Activities 157](#_Toc129553028)

[Data Mapping – View Caseload 158](#_Toc129553029)

[Data Mapping – Add New Employee 159](#_Toc129553030)

[Amendments History 160](#_Toc129553031)

[Bibliography 161](#_Toc129553032)

**Table of Figures**

[Figure 1 Top-level use case diagram 4](#_Toc129552866)

[Figure 2 Use case 2.1 Create New Case 5](#_Toc129552867)

[Figure 3 Use case 2.1 Create New Case, refined 6](#_Toc129552868)

[Figure 4 Use Case 2.2 Search Cases 7](#_Toc129552869)

[Figure 5 Use Case 2.2 Search Cases, refined 8](#_Toc129552870)

[Figure 6 Use Case 2.3 View Caseload, refined 9](#_Toc129552871)

[Figure 7 Fully Dressed Use Case Diagram 11](#_Toc129552872)

[Figure 8 Top-level Conceptual Model 40](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552873)

[Figure 9 Refined Conceptual Model 83](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552874)

[Figure 10 Conceptual Classes with Attributes 85](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552875)

[Figure 11 The Final Conceptual Model 86](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552876)

[Figure 12 A Java class is the blueprint of an object 87](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552877)

[Figure 13 Case CRC Card 89](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552878)

[Figure 14 Solicitor - CRC Card 89](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552879)

[Figure 15 Client - CRC Card 90](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552880)

[Figure 16 Activity - CRC Card 91](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552881)

[Figure 17 Initial Class Diagram (1) 94](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552882)

[Figure 18 Case Type - Enumeration 97](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552883)

[Figure 19 Job Title - Enumeration 97](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552884)

[Figure 20 Activity - Enumeration 98](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552885)

[Figure 21 Top Level Class Diagram - Generalisation (Person) 99](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552886)

[Figure 22 Initial Class Diagram (2) 101](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552887)

[Figure 23 Composition - Case and Client 105](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552888)

[Figure 24 BillableActivity - Associations and Composition 106](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552889)

[Figure 25 Refined Class Diagram 108](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552890)

[Figure 26 Sequence Diagram - Log In 111](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552891)

[Figure 27 Sequence Diagram - Log In with Validation 112](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552892)

[Figure 28 Sequence Diagram - New Employee Log In 113](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552893)

[Figure 29 Sequence Diagram - Create New Case 115](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552894)

[Figure 30 Sequence Diagram - Search Cases 117](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552895)

[Figure 31 Sequence Diagram - Search Cases with Delete Option 118](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552896)

[Figure 32 Activity Diagram - Log In 120](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552897)

[Figure 33 Activity Diagram - New Employee Log In 121](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552898)

[Figure 34 Activity Diagram - Add New Case 122](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552899)

[Figure 35 Activity Diagram - Search Cases 124](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552900)

[Figure 36 Log In Wireframe 127](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552901)

[Figure 37 Home Page Wireframe 128](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552902)

[Figure 38 Searching - Wireframe 129](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552903)

[Figure 39 Recording Billable Activities - Wireframe 130](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552904)

[Figure 40 Add New Case - Wireframe 131](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552905)

[Figure 41 View Caseload Wireframe 132](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552906)

[Figure 42 Our Staff - Wireframe 133](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552907)

[Figure 43 Add New Employee - Wireframe 134](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552908)

[Figure 44 GUI Colour Scheme - Contrast Checker 137](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552909)

[Figure 45 GUI Colour Scheme - Readability 138](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552910)

[Figure 46 Log In - User Interface 139](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552911)

[Figure 47 Home - User Interface 140](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552912)

[Figure 48 Add New Case - User Interface 141](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552913)

[Figure 49 Search - User Interface 142](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552914)

[Figure 50 Record Billable Activity - User Interface 143](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552915)

[Figure 51 View Caseload - User Interface 144](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552916)

[Figure 52 Our Staff - User Interface 145](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552917)

[Figure 53 Add New Employee - User Interface 146](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552918)

[Figure 54 Data Binding - Log In 152](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552919)

[Figure 55 Data Binding - Case Details 153](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552920)

[Figure 56 Data Binding - Client Details 154](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552921)

[Figure 57 Data Binding - Solicitor Details 155](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552922)

[Figure 58 Data Binding - Case Details 156](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552923)

[Figure 59 Data Binding - Record Billable Activity 157](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552924)

[Figure 60 Data Binding - View Caseload 158](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552925)

[Figure 61 Data Binding - Add New Employee 159](file:///E:\Graded%20Unit\1.%20Planning%20Phase\1b%20Solution%20Planning\586661\586661%20-%20Planning%20Phase%20Part%20B%20-%20Solution%20Planning%20Report.docx#_Toc129552926)

# Overview

This document relates to the Graded Unit 2 project as part of the HND Computing: Software Development H48W 35. It aims to satisfy Stage 1, Part B of the overall project by providing an analysis of the business model and the view model for the system being developed, along with evidence for the analysis and design of the prototype system. This report will make reference to previous documentation submitted for this project, namely the Action Plan Report.

# Introduction

The contents of this document are intended to guide the project team into the Development phase of this project. In accordance with the chosen software development methodology, the Unified Process, it will **elaborate upon** sections of the Action Plan Report to allow for quality decision making throughout the **analysis and design workflows**. The main workflows that this document evidences are: business modelling, requirements, and, to a greater extent, analysis and design. I have included evidence of client interaction in this report too, to show the client’s feedback on the progression of the analysis and design of the evolving system and to record client feedback. The initial project brief can be found in the Action Plan Report, so I have omitted the brief from this document to avoid unnecessary repetition.

The Action Plan (AP) Report provided initial planning models which were presented to the client for approval, and culminated in setting out what the project team believes are the overall aims of this project along with the resources required to achieve the requirements established in the AP Report. I have not appended this report to the AP for purposes of clarity.

Using this AP Report as our foundations, Section 1 of this report aims to provide an analysis of the **business model**. It starts by offering a **Use Case Model**. A key factor that led to adopting the UP is its agile nature: a basic use case diagram and a refined use case diagram were included in the AP Report, so using this combined with the basic conceptual model in the AP Report, we already have a good starting point for this section. This report will now elaborate on this and produce a use‑case model in full. In other words, a fully dressed use case diagram will be shown and discussed before providing fully dressed use case descriptions.

The business modelling workflow is one of the main workflows of the UP, and a business model cannot be thought of as thorough if it does not devise a **domain model**. Therefore, Section 1 also attempts to address the domain by using the conceptual model in Section 4 of the AP Report as a starting point. As discussed in the AP Report, the chosen development route is a Java project with data structures. Therefore, the problem domain must be approached in an **object‑oriented manner**. In order to do this, Section 1 performs a **natural language analysis** of the use case descriptions, allowing for the identification of several noteworthy **conceptual classes**. It should be stressed here that the inclusion of conceptual classes in a domain model implies that they are not intended to be taken as classes in the software. The conceptual classes are simply another method of helping the project team to understand and analyse the problem domain which the evolving system will address.

To further analyse the problem domain, Section 1 then informally develops the domain model through **Class Responsibility Collaborator (CRC) cards**. The purpose of this activity is to begin to consider what the most important classes identified should be responsible for, and also show how these classes should, or should not, interact with each other. This is an important step in beginning to design the evolving system in an object‑oriented manner. This technique provides a nice link into the **static and dynamic models**, which lead us out of analysis and into the design of the **software**. The static and dynamic models are concerned with the design of the software, with the former expressing how I intend system components will be structured irrespective of time; the latter expressing how I intend for objects in the evolving system will interact with each other.

Where Section 1 analyses the problem domain and begins designing the software, Section 2 leads deeper into the design workflow by addressing the **aesthetics of the solution’s interface**. With the static structure of the system and the system's behaviour at runtime designed, Section 2 moves on to design the **user interface**. There is a discussion on user experience design and Human Computer Interaction, along with a discussion on general accessibility that has been considered before and throughout the design of the system’s aesthetics. The user interface design also takes into consideration a **user analysis**, and relevant client interaction records are included here too.

Finally, to bring all of the design work together, there is then a **data binding model** produced. This is simply to show how everything connects together, so when the system receives data, this data binding model shows where this data is then stored.

# Section 1 – Business Model

This project is focused on developing one application. That is to say that the complexity is no where near that of developing a product for a large‑scale enterprise. As such the analysis of the business model is best done through developing a use case model and a domain model. In planning a solution for the client, the attention should not solely lie on designing the aesthetics of the evolving system; we are still analysing requirements, so there are still activities to be done before we start designing the view model. In analysing the business model, though, some decision‑making can take place. For example, the use case model attempts to detail the logic of the evolving system, so we can consider potential appropriate data structures available to us. The use case model leads nicely into considering the domain model. It provides a link in the form of linguistic analysis, which leads to the consideration of conceptual classes. The conceptual model will then allow us to start considering the solution in an object‑oriented manner, leading into the static and dynamic model of the evolving system. A lot of these activities overlap; such is the nature of an agile methodology. The overall aim of this section is to outline how we are going to satisfy the requirements that have been established.

## The Use Case Model

A use case model is an important aspect of the Unified Process as this is where we pay attention to what the intended user(s) should be able to perform on the new system. This is where we place emphasis on user involvement and the creation of various visual aids can be shared with the client to feed back on whether requirements are missing. There is a false belief that moving into the Elaboration phase of the UP solely involves the design work being carried out. However, it is an agile methodology, and a main key deliverable (as set out in the previously submitted Action Plan report), is a continuation of the vision document, or in this context, the Action Plan report.

I have set out the use case model before analysing the domain model. This is because it allows me to further analyse the requirements of the evolving system, and this will result in a smoother transition into analysing the problem domain. Ultimately, providing a use case model will allow for a more thorough and detailed design, and a well-thought-out design will likely make for a smoother development phase, which all contributes to the success of the project overall.

### Top Level Use Case Diagram

A top‑level use case diagram (Figure 1) which had been refined was previously presented to the client both in the Action Report submitted and also in meeting 2 with the client. The minutes of meeting 2 show that the client approved of this use case diagram and indicated it captured all of the functionality that they wish to be available to their employees in the evolving system. I will therefore use this as our starting point.

Diagram

Description automatically generated

Figure Top-level use case diagram

### Fully Dressed Use Case Diagrams

It is quite obvious from the top‑level use case diagram (Figure 1) that the bulk of the evolving system’s functionality lies in use case number 2 – Manage Cases. The remaining use cases ((1) Log In, (3) Generate Report, and (4) Exit System) are, in my view, self‑explanatory. Therefore, I have focused the refinement of the top‑level use case diagram on the Manage Cases use case because without this use case, the project has no purpose. The use cases "Log In" and "Generate Report" have been addressed in the fully dressed use case descriptions. For purposes of readability, I have included sub‑diagrams to highlight the most important use cases.

#### Use Case 2.1 – Creating a New Case

The first sub‑diagram (Figure 2) zones in on use case number 2.1 in the top‑level use case diagram.

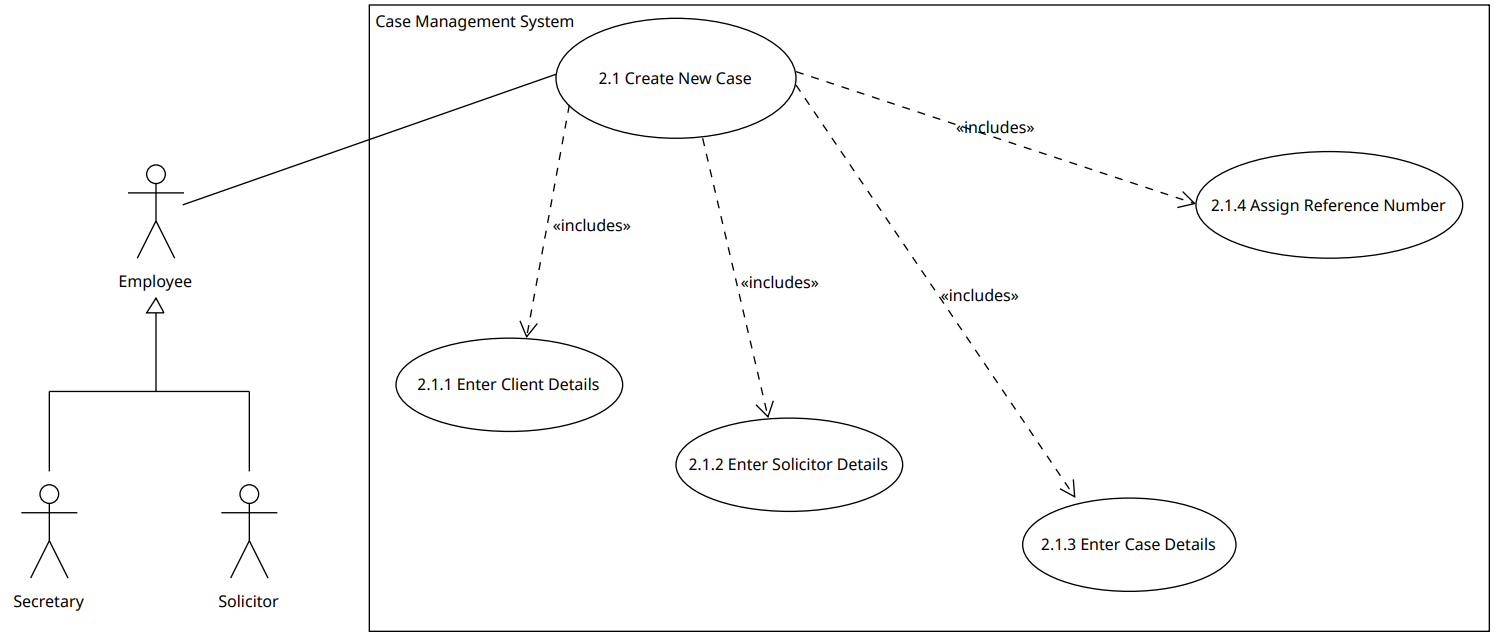


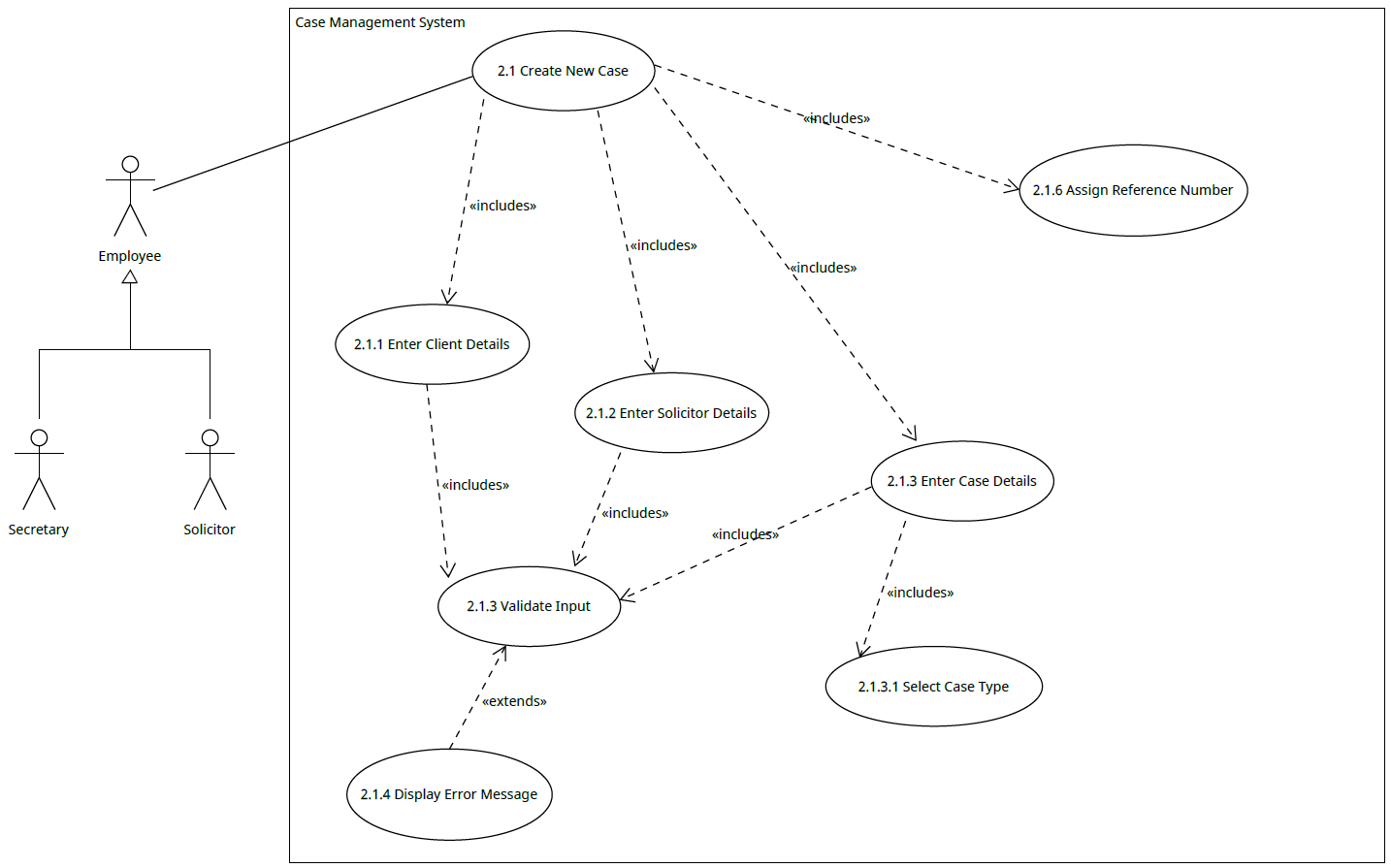
Figure Use case 2.1 Create New Case

In creating this sub‑diagram I have taken into account that a user can be either a Secretary or a Solicitor. This is primarily to show our understanding of the organisation of the business: that the end user of the evolving system is going to be the client’s Secretaries or Solicitors. Each of these users will have available to them the same functionality, hence why I have included a generalisation relationship between these actors – this simply means an Employee will either be a Secretary or a Solicitor. The refined sub‑diagram better communicates that in creating a new case the user will: enter client details, the responsible solicitor’s details, and the relevant case details (2.1.1, 2.1.2, and 2.1.3).

It is possible to devise a **brief** use case description from the above sub‑diagram. The table below provides a simple story of the system achieving the objective of creating a new legal case.

|  |
| --- |
| **Use Case 2.1 – Create New Case** |
| When the Employee needs to add a new case to the system, they must press the “Create New Case” button. The user will then be prompted to: enter the required client details, solicitor details, and case details. A case reference number will then be assigned to this new case. |

However, when we review functional requirement 2 in the Action Plan report, it is clear that all input should be valid. This means the evolving system will have to implement appropriate **validation** algorithms to ensure input is acceptable. It was also established that an important case detail to be stored is the **type of case** (see Non‑Functional Requirement 6.1 in Action Plan report). This results in a further refined use case diagram (Figure 3).



2.1.5 Display Error Message

2.1.4 Validate Input

Figure Use case 2.1 Create New Case, refined

The elaborated sub‑diagram above (Figure 3) now shows that all input will go through the appropriate validation. We can also see from the sub‑diagram that the result of this validation could potentially result in an error message, and that part of entering the case details will require the user to select the type of case (2.1.3.1). Finally, we can see that a case reference number will be assigned when a user creates a new case (2.1.6). This sub‑diagram is a more accurate reflection of Functional Requirement 2 that is detailed in the Action Plan report.

#### Use Case 2.2 – Searching Cases

The search functionality is another critical use case in the evolving system. From this, the user will access much of the essential functionality that will achieve the client’s objective of managing their workload more efficiently. The top‑level use case diagram describes that a user can search for a case by entering the relevant case reference number. It also describes that a user can record billable activities having searched for a case. Finally, it indicates that if a case is to be deleted, the user must search for the case to be deleted.

However, this diagram omits important details. More specifically, it does not describe that all input should go through appropriate validation and it also does not describe that to record a billable activity, the user must specify an activity and enter the time spent on that activity *after* entering the case reference number. I note also that the top‑level diagram describes that a user may delete a case, but in order to delete a case the user will have to enter the case reference number; therefore, *2.2.3 Delete Case* should be an extension of *2.2.1 Enter Case Reference Number* and not *2.2 Search Cases*; similarly, *2.2.1 Enter Case Reference Number* should include *2.2.2 Record Billable Activities*. I have again created a sub‑diagram to reflect this refinement (Figure 4).

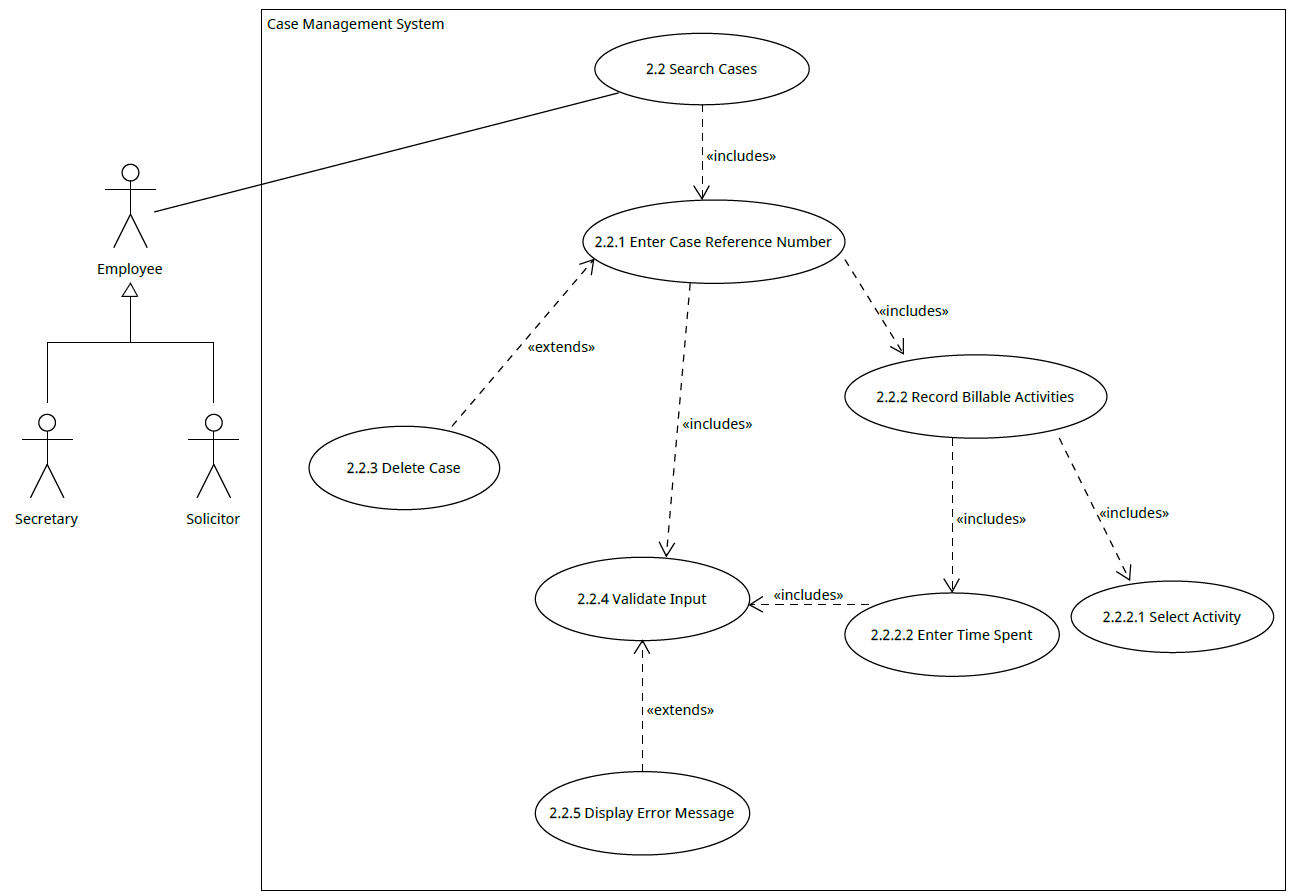


Figure Use Case 2.2 Search Cases

Now, this sub‑diagram reflects that a user can search for a case by entering a case reference number. It also describes that the recording of billable activities is achieved through searching for a case and requires that the user specify the type of activity to be recorded and the time spent on that activity.

A brief use case description that describes the user successfully searching for a case is provided in the table below.

|  |
| --- |
| **Use Case 2.2 – Search Cases** |
| A Solicitor has made progress on a case and needs to update the case on the system to record this. The user can update the case by searching for the case’s unique reference number. On entering the reference number, the user will click into the case and will be able to record the billable activity. The system will then validate and record the user’s input. If a user searches for a case reference number, the option to delete this case will be available. |

If we review the functional requires in the Action Report and the minutes of meeting number 2 with the client, we can see that this refined sub‑diagram does not describe that the system should be able to calculate the total monies owed to the firm based on their solicitor’s hourly rates. This meeting with the client also revealed that they want the evolving system to be intuitive, so I have amended the sub‑diagram (see Figure 5) to describe that if the user presses “Delete”, then a confirmation message should be displayed for the user to confirm they indeed want to delete a case. This will give the user the opportunity to review their decision in case they have simply clicked Delete by accident.

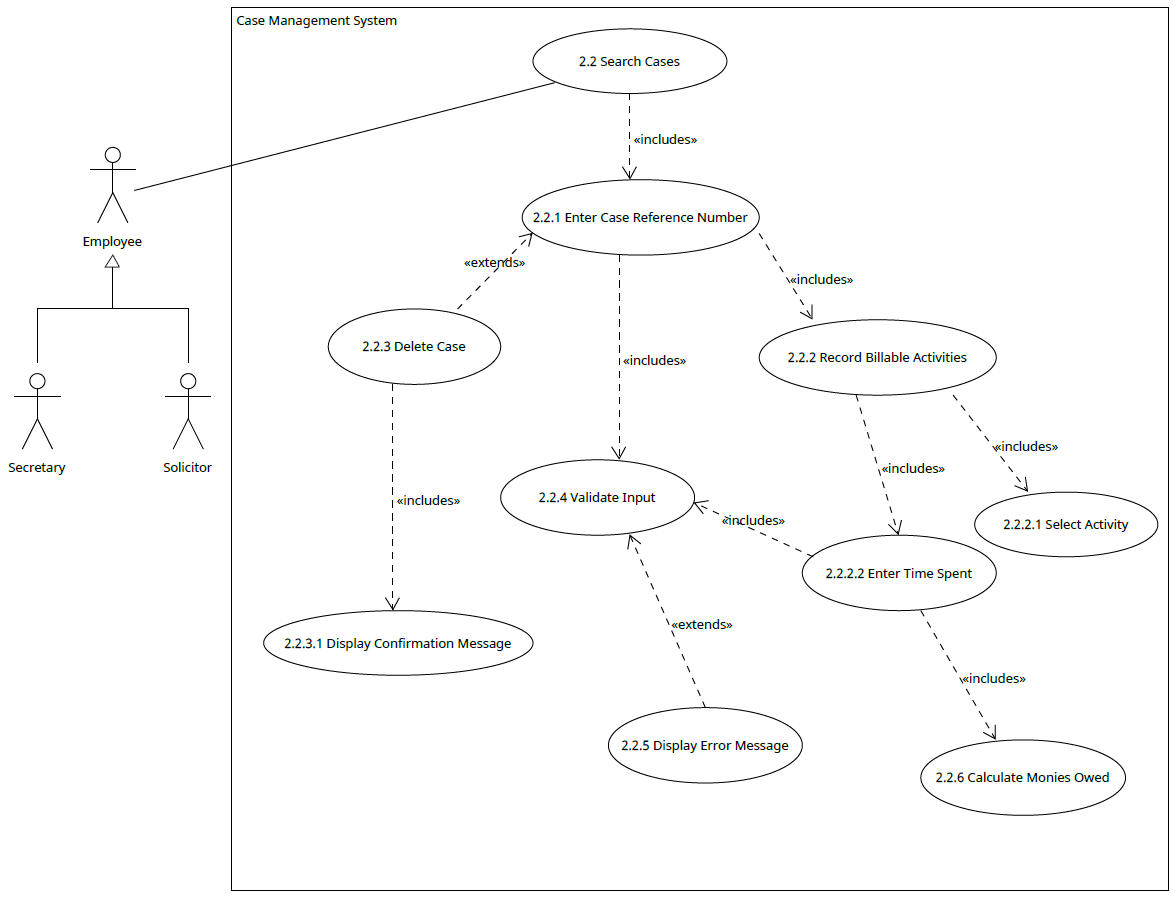


Figure Use Case 2.2 Search Cases, refined

The elaborated sub‑diagram above now provides a more accurate representation of the functionality of the evolving system. It is important for the development team to read this in conjunction with the Action Plan report as the sub‑diagram above assumes that on searching for a case referencing number, the case’s details will be displayed in a structured manner. We can see that on entering a case reference number, the user is then able to record billable activities and the amount of monies owed is tracked by the system based on the time‑spent input. We can also see that all user input is validated, and a helpful, user‑friendly message will be displayed if the Delete button is clicked.

#### Use Case 2.3 – View Caseload

The final use case that completes 2. Manage Cases is *2.3 View Caseload*. From the top‑level use case diagram we can see this is a standalone use case in that it does not include any other use cases. The only sensible refinement here would be to show that in viewing the current caseload (that is, all cases regardless of type), the user should be able to filter their view to all Criminal cases, Immigration cases, or Personal Injury cases. The sub‑diagram below reflects this refinement. This may seem meaningless to the overall system, but this functionality will contribute to managing the firm’s workload more efficiently as it allows the user dealing with an initial enquiry to quickly establish if there is a Solicitor who has availability to take on that case (the maximum number of cases a Solicitor can be responsible for at any one time is 15).

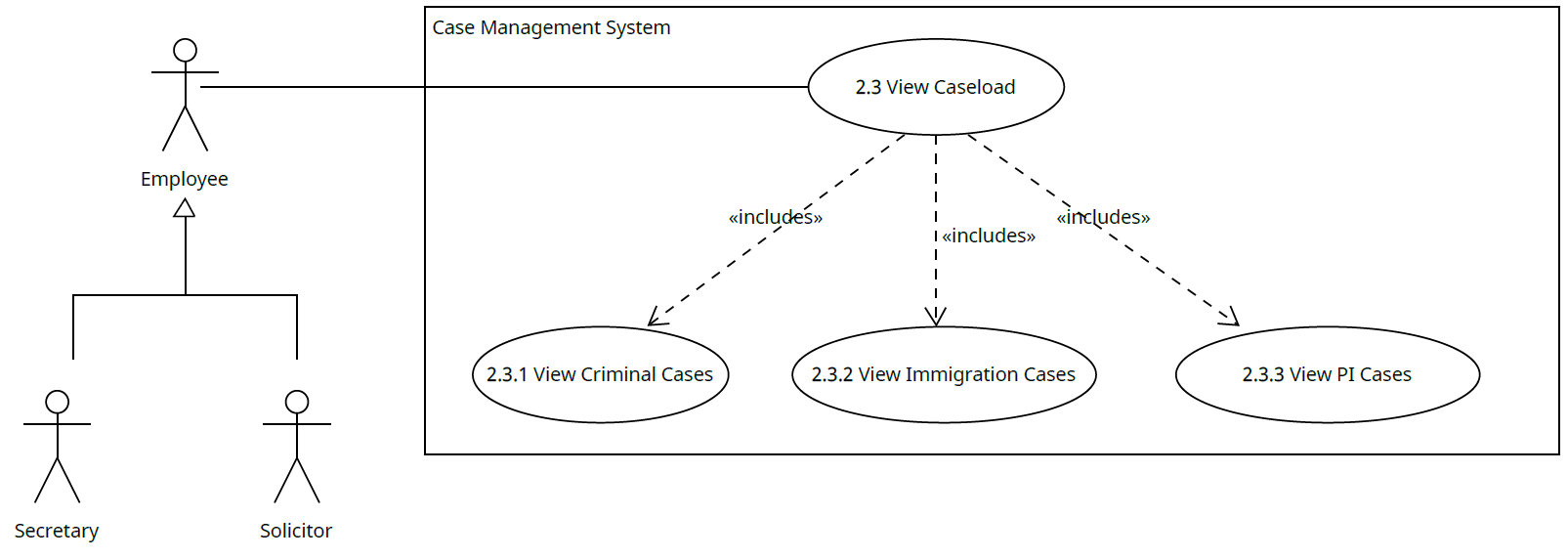


Figure Use Case 2.3 View Caseload, refined

A brief use case description that describes the user successfully viewing all cases stored in the system is provided below.

|  |
| --- |
| **Use Case 2.3 – View Caseload** |
| An Employee has received an enquiry from a potential client and wishes to view all cases the firm is responsible for at this current time to establish whether a Solicitor has availability to take on a new case. The user will press the “View All Cases” button and the system will retrieve and display all cases.  The user may wish to refine their search to all cases by type. Therefore, the user will press either the “View Criminal Cases”, “View Immigration Cases”, or “View PI Cases” button. The system will then retrieve and display all cases by type. |

The fully dressed use case diagram reflecting the aforementioned refinements is provided below (Figure 7). A PDF version of this diagram will be submitted with this report.

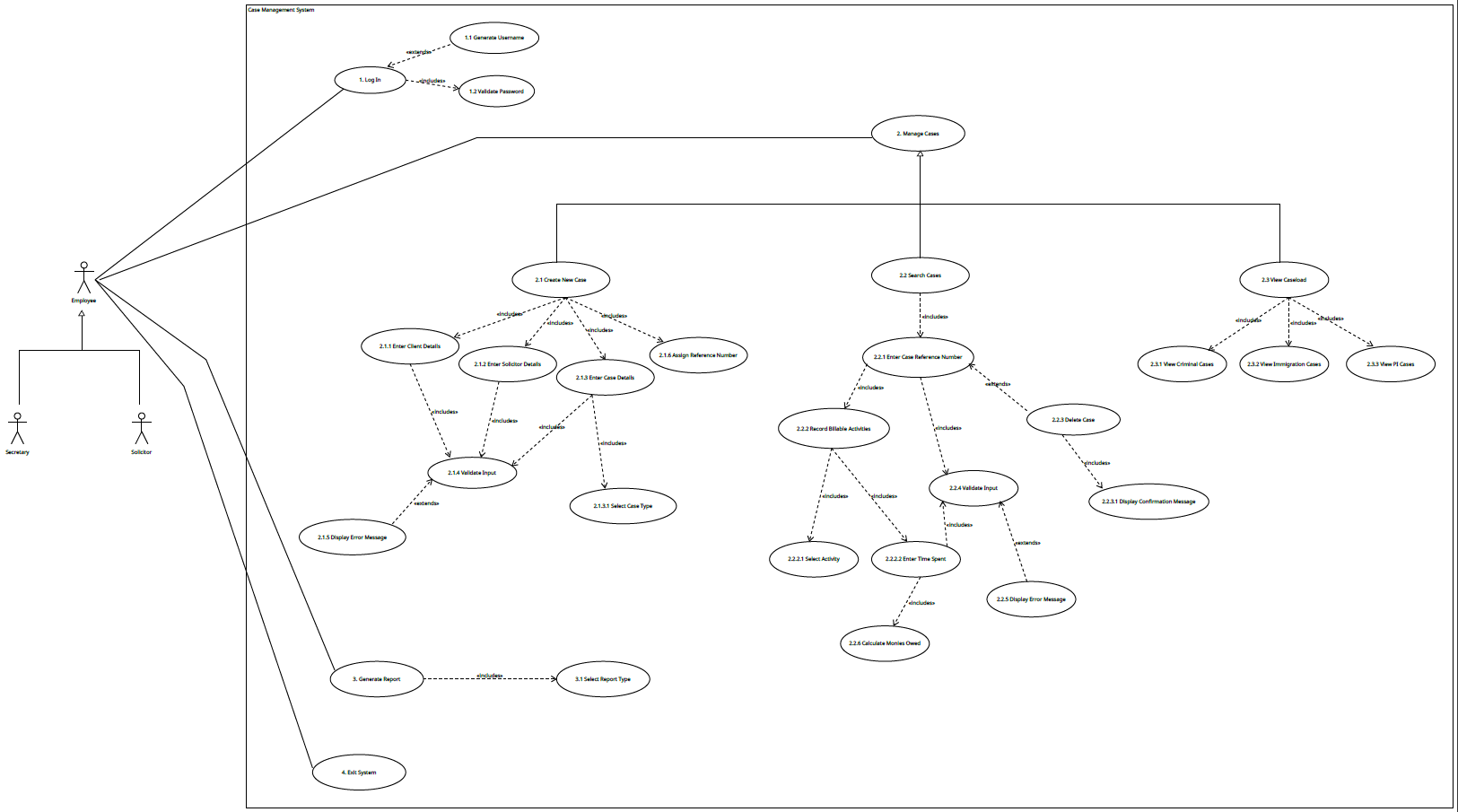


Figure Fully Dressed Use Case Diagram

#### Prioritisation

It is sensible to now review the prioritised list of requirements before we produce fully-dressed use case descriptions. In the Action Plan report, a prioritised list of functional requirements was provided, but now that we have devised a more refined use case diagram with brief use case descriptions, I have decided to iterate over this and re‑evaluate the prioritisation of use cases rather than the list of functional requirements to ensure that the project team’s time and effort is focused on the correct objectives.

In order for the evolving system to have any meaning at all, use case number 2 “Manage Cases” absolutely must be implemented. Without this use case, all the client will be able to do is Log In and Exit System. In order to achieve use case 3 “Generate Reports”, the client must be able to access all of the behaviour contained within “Manage Cases”. Once the system successfully allows the client to “Manage Cases”, then the implementation of “Log In” can take precedence – we must remember that the client is likely to input sensitive and confidential data into the new system, so a secure log in is important. We are then left with use case number 3 “Generate Reports” and 4 “Exit System”. The ability to generate reports will allow users to view relevant data immediately, which will contribute to making the overall management of the firm’s caseload more efficient. It is quite obvious that a user should be able to exit the system and this is not a meaningful requirement, as such I will keep this at the bottom of the prioritised list of use cases. So, in order to make this project worth all stakeholders’ while, I propose the following list of prioritised use cases:

| **Prioritised Use Cases** | |
| --- | --- |
| **Priority** | **Use Case** |
| 1 | Manage Cases (currently use case 2)  This includes 2.1 – 2.3. |
| 2 | Log In (currently use case 1)  This includes the current 1.1 and 1.2 |
| 3 | Generate Reports (currently use case 3) |
| 4 | Exit System (currently use case 4) |

### Fully Dressed Use Case Descriptions

The development of the use case diagrams is simply a technique to analyse the requirements of the evolving system. But, these diagrams are primarily a way of communicating to any stakeholder regardless of technical experience what a given user of the new system will be able to do. To really comprehend the behaviour of the evolving system, the most valuable exercise the project team can do is to “*write text, not diagram or focus on use case relationships*” which often results in wasted time and effort by “*an over‑concern with secondary issues*” like actors and relationships (Larman, 2005). Although strictly speaking not object‑oriented, use case descriptions are a “*requirements analysis tool*” and act as “*a pivotal input into classic OOA/D activities*” (Larman, 2005). However, Larman also describes how many people are visual learners and “*expressing their software design thoughts in a visual language complements their nature…A large percentage of the brain is dedicated to visual or iconic thinking and processing, rather than textual processing…*” (Larman, 2005). With this in mind, the use case descriptions are written in an attempt to describe the logic of the system we are developing, but it is worth remembering that the designing of the system should take all of the activities performed into consideration – we shouldn’t depend on just one activity to design the system.

The following use case descriptions are fully‑dressed and will heavily influence the artifacts that will be generated as part of the Unified Process, for example when detailing input/output events in the design model, or when developing a Test Plan. I have focused on use cases that are integral to the purpose of the evolving system and as such have omitted use case descriptions for use cases that are not explicitly mentioned in the prioritised list above or will result in unnecessary repetition. For example, for use cases such as “Display Error Message” or “Validate Input”, key details are set out under Data Validation. I have also omitted a use case description for “Exit System” use case as I do not believe the Development team will need any elaboration or additional details here. **The numbering of these use case descriptions follows that listed in the prioritised list of use cases, and not the numbering shown in the use case diagram**. I have indicated the use case’s number as shown in the diagram in brackets following the use case name. The coloured highlighting refers to the linguistic analysis contained in The Domain Model. I have detailed below what the detailed use case descriptions in this report are comprised of.

| **Sections of a Use Case Description** | |
| --- | --- |
| **Section** | **Contents and Purpose** |
| Overview Description | **Use Case Name**: the name of the use case as it appears in the use case diagram  **Use Case ID**: Minimises risk of ambiguity by making clear what use case is being referred to. Makes reference to the prioritised list. For example, the use case description for “Create New Case” has an ID of 1.1 which references the prioritised list; its old numbering is 2.1 in the use case diagram.  **Importance Level**: the importance of the use case in question, can be either Very High, High, or Moderate. Will reflect the prioritised list of use cases. For example, use case “Manage Cases” will be Very High.  **Primary Actor**: the primary actor is identified in order to document who this use case will be initiated by / who will be able to achieve this use case. It is the role an external force will play.  **Use Case Type**: will set the type of the use case to show if the use case is essential, or if it is required for extra detail. The nature of the use case.  **Stakeholders and Interests**: a list of the stakeholders who will have an interest in this use case. Stakeholders are not limited to being primary actors; it refers to anyone who will benefit from the system achieving this use case successfully. This is simply an user that may have a need for achieving this use case along with a brief description of how and/or why they have a need for this use case.  **Brief Description**: simply an overview of the objective of the use case. This might include some repetition of the brief use case descriptions. It summarises how the use case is relevant to the overall system or process. It describes the purpose of the use case.  **Trigger**: The event that causes the use case to come into action. This could be a user pressing a button, for example.  **Relationships**: This identifies all of the relevant relationship lines from the use case diagram. |
| Flow of Events | **Normal Flow**: The steps required to achieve the use case. These steps are often concerned with the business processes that lead to the use case goal. They should, in general, be listed hierarchically in the order that they are performed. The normal flow of events is the steps in the process that will result in a successful use case, or in other words what should happen when nothing goes wrong.  **Subflows**: A subflow is essentially part of the normal flow of events, but it provides additional details. It is a more detailed explanation of parts of the normal flow of events.  **Alternate/Exceptional Flows**: Identifies something you may reasonably expect to go wrong; behaviours that might happen. For example, if the wrong data type is entered, an error message is displayed. |

#### Use Case: Manage Cases (1)

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name: Manage Cases (2) | | ID: 1 | Importance Level: Very High |
| Primary Actor: Employee | **Use Case Type:** Essential, Detail | | |
| Stakeholder and Interests:  Solicitor: Wants to organise their workload in a structured and efficient manner.  Secretary: Wants to support the Solicitor in the organising of the firm's caseload. | | | |
| Brief Description: The firm may receive customer enquiries seeking legal advice relating to a variety of matters. If a Solicitor is able to accept a new client, they will have to add a new case to their system. If a Solicitor has made progress on a specific case, they may have to update some details of the case in the system. If an Employee has to draft legal documentation, they may have to look up specific details of a case (for example, a client's name) to accurately fill in legal documents. | | | |
| Trigger: The firm has a heavy workload and need to organise their cases. | | | |
| Relationships:  Association: Employee  Includes:  Extends:  Generalisation: 1.1, 1.2, 1.3 | | | |
| Normal Flow of Events:   1. If a new client enquiry has been received, subflow 1 is followed. 2. If a case has progressed and billable activities have to be recorded, subflow 2 is followed. 3. If a client has changed their details (for example, got a new phone number), subflow 3 is followed. 4. If specific details of a case need to be obtained, subflow 3 is followed. | | | |
| Subflows:  Subflow 1  A Solicitor has decided to take on a new case.  The Solicitor will obtain case and client details.  The Solicitor will email the Secretary with these details.  The Secretary will press "Add New Case" button.  The normal flow of events for use case 1.1 "Create New Case" are followed.  Subflow 2  The Solicitor has made progress on a case and has to record the relevant billable activities that have resulted in this progress.  The Solicitor may ask the Secretary to record these activities on their behalf, or may do it themselves.  Regardless, the user will locate the search bar which will be obvious to the user and present at all times.  The normal flow of events for use case 1.2 "Search Cases" are followed.  Subflow 3  The user will enter the case reference number in the search bar.  The normal flow of events for use case 1.2 "Search Cases" are followed. | | | |
| Alternate/Exceptional Flows:  Appropriate error message is displayed if data is invalid.  If more than one field is invalid, the invalid fields should be outlined in red. | | | |
|  | | | |

#### Use Case: Create New Case (1.1)

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name: Create New Case (2.1) | | ID: 1.1 | Importance Level: Very High |
| Primary Actor: Employee | **Use Case Type:** Essential, Detail | | |
| Stakeholder and Interests:  Solicitor: Wants to record information relating to their caseload in one place.  Secretary: Wants to support the Solicitor in administration tasks. | | | |
| Brief Description: The Solicitor has reviewed a customer enquiry and has decided to create a new case which should be added to the caseload. The Solicitor then instructs the Secretary to add a new case to the system. User must then log in and press the “Add New Case” button. The user will then be prompted to: enter the required client details, solicitor details, and case details. After inputting the details, the Secretary will then press a "Confirm New Case" button. A case reference number will then be assigned to this new case. | | | |
| Trigger: A Secretary has received the Solicitor’s email with relevant case and client details. | | | |
| Relationships:  Association:  Includes: 1.1.1, 1.1.2, 1.1.3, 1.1.6  Extends:  Generalisation: | | | |
| Normal Flow of Events:   1. Solicitor requests for a new case to be added to the system. 2. Solicitor obtains client details. 3. Solicitor emails the case and client details to Secretary. 4. Secretary will log in to system if not already logged in. 5. Secretary will press "Add New Case" button. 6. System will prompt user to enter case, client, and Solicitor details. 7. Secretary inputs case, client, and Solicitor details. 8. Secretary must then press "Confirm New Case" button. 9. A unique reference number will be assigned to this new case. 10. A confirmation message will be displayed to the user confirming the case title and its reference number. 11. A case is successfully added to the system's storage. | | | |
| Subflows:  Step 8 will trigger appropriate validation for case, client, and Solicitor details. Validation rules can be found in the following section, under Data Validation.  Step 10 will trigger generation of a unique reference number in the format of 3 digits followed by a dash followed by the first three letters of the case type, for example:  001-CRI | | | |
| Alternate/Exceptional Flows:  Appropriate error message is displayed if data is invalid.  If more than one field is invalid, the invalid fields should be outlined in red. | | | |
|  | | | |

#### Use Case: Enter Client Details (1.1.1)

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name: Enter Client Details (2.1.1) | | ID: 1.1.1 | Importance Level: Very High |
| Primary Actor: Employee | **Use Case Type:** Essential, Detail | | |
| Stakeholder and Interests:  Solicitor: Needs accurate client details to know exactly who this case relates to.  Secretary: Wants to support the Solicitor in administration tasks; entering wrong client details could result in data breach. | | | |
| Brief Description: In order to successfully create a new case to be stored in the system, valid client details have to be entered. If the firm don't specify the required client details, they will not know who the case relates to. | | | |
| Trigger: Add New Case button has been pressed. | | | |
| Relationships:  Association:  Includes: 1.1.4  Extends:  Generalisation: | | | |
| Normal Flow of Events:   1. Secretary has received email from Solicitor with client and case details 2. Secretary presses "Add New Case" button. 3. System will prompt user to enter client details. 4. Secretary presses “Confirm New Case” button when done. 5. System validates input. 6. A success message is displayed to screen. | | | |
| Subflows:   * Step 3 requires user to enter client's: first name, last name, date of birth, address first line, address second line, postcode, phone number, and initial customer enquiry type. * Step 4 will trigger relevant validation. Validation rules can be found under Data Validation. | | | |
| Alternate/Exceptional Flows:  Appropriate error message is displayed if data is invalid.  If more than one field is invalid, the invalid fields should be outlined in red. | | | |
|  | | | |

#### Use Case: Enter Solicitor Details (1.1.2)

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name: Enter Solicitor Details (2.1.2) | | ID: 1.1.2 | Importance Level: Very High |
| Primary Actor: Employee | **Use Case Type:** Essential, Detail | | |
| Stakeholder and Interests:  Solicitor: Needs to know what cases he/she is responsible for.  Secretary: Wants to support the Solicitor in administration tasks; entering wrong Solicitor details could result in a breakdown of communication. | | | |
| Brief Description: In order to successfully create a new case to be stored in the system, valid Solicitor details have to be entered. If the firm don't specify the required Solicitor details, they will not know what Solicitors are responsible for what cases. | | | |
| Trigger: Add New Case button has been pressed. | | | |
| Relationships:  Association:  Includes: 1.1.4  Extends:  Generalisation: | | | |
| Normal Flow of Events:   1. Secretary has received email from Solicitor with client and case details 2. Secretary presses "Add New Case" button. 3. System will prompt user to enter Solicitor details. 4. Secretary presses “Confirm New Case” button when done. 5. System validates input. 6. A success message is displayed to screen. | | | |
| Subflows:   * Step 3 requires user to select the responsible Solicitor from a dropdown menu. * Step 4 will trigger relevant validation. Validation rules can be found under Data Validation. | | | |
| Alternate/Exceptional Flows:  Appropriate error message is displayed if data is invalid.  If more than one field is invalid, the invalid fields should be outlined in red. | | | |
|  | | | |

#### Use Case: Enter Case Details (1.1.3)

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name: Enter Case Details (2.1.3) | | ID: 1.1.3 | Importance Level: Very High |
| Primary Actor: Employee | **Use Case Type:** Essential, Detail | | |
| Stakeholder and Interests:  Solicitor: Needs accurate case details entered because they can’t remember every detail of every case off the top of their head.  Secretary: Wants to support the Solicitor in administration tasks; entering wrong case details could result in a breakdown of communication. | | | |
| Brief Description: In order to successfully create a new case to be stored in the system, valid case details have to be entered. If the firm don’t enter a case type, title, and description, they will not know how to advise their clients. | | | |
| Trigger: Add New Case button has been pressed. | | | |
| Relationships:  Association:  Includes: 1.1.4, 1.1.3.1  Extends:  Generalisation: | | | |
| Normal Flow of Events:   1. Secretary has received email from Solicitor with client and case details 2. Secretary presses "Add New Case" button. 3. System will prompt user to enter case details. 4. Secretary presses “Confirm New Case” button when done. 5. System validates input. 6. A success message is displayed to screen. | | | |
| Subflows:   * Step 3 requires user to select the responsible Solicitor from a dropdown menu. * Step 4 will trigger relevant validation. Validation rules can be found under Data Validation. | | | |
| Alternate/Exceptional Flows:  Appropriate error message is displayed if data is invalid.  If more than one field is invalid, the invalid fields should be outlined in red. | | | |
|  | | | |

#### Use Case: Assign Reference Number (1.1.6)

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name: Assign Reference Number (2.1.6) | | ID: 1.1.6 | Importance Level: Very High |
| Primary Actor: Employee | **Use Case Type:** Essential, Detail | | |
| Stakeholder and Interests:  Employee: In order to retrieve the correct case to update its details, a case must have a unique identifier. | | | |
| Brief Description: Once a user has entered the required case, client, and Solicitor details and all have been validated, the system must create and assign a unique reference number to that case. | | | |
| Trigger: The user has entered valid details and pressed “Confirm New Case” button. | | | |
| Relationships:  Association:  Includes:  Extends:  Generalisation: | | | |
| Normal Flow of Events:   1. Secretary enters valid details and presses "Confirm New Case" button. 2. The system will generate the unique reference number. 3. This will then be assigned to this new case. 4. A confirmation message will be displayed to the user confirming the case title and its reference number. 5. A case is successfully added to the system's storage. | | | |
| Subflows:   * Step 2 requires the system to concatenate three integers, a dash, and the first three letters of the case type. For example, if reference number 001-CRI already exists, the next Criminal case should be assigned 002-CRI as a case reference number. If 002-CRI already belongs to a Criminal case, the next Criminal case reference number should be 003-CRI, and so on. * Developer note: The maximum number of cases any one Solicitor can have at any one time is 15. If a Solicitor currently has 15 cases ongoing, this will be revealed through use *1.3 View Caseload* or use case *3 Generate Report*. Therefore, when an initial customer enquiry is being reviewed, the Solicitor will check their current workload and decide at that point if they can take on that case. If they are at maximum caseload, it is the responsibility of the Solicitor and not the system to reject that case. Despite this, it is still sensible to at least trigger a brief message to be displayed informing the user that the Solicitor is at maximum capacity. | | | |
| Alternate/Exceptional Flows: | | | |
|  | | | |

#### Use Case: Search Cases (1.2)

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name: Search Cases (2.2) | | ID: 1.2 | Importance Level: Very High |
| Primary Actor: Employee | **Use Case Type:** Essential, Detail | | |
| Stakeholder and Interests:  Solicitor: Has made progress on a case and needs to update the system.  Secretary: Has been instructed to update or edit a case. | | | |
| Brief Description: A Solicitor has made progress on a case and needs to update the case on the system to record this. The user can update the case by searching for the case’s unique reference number. On entering the reference number, the user will click into the case and will be able to record the billable activity. The system will then validate and record the user’s input. If a user searches for a case reference number, the option to delete this case will be available, i.e. a Delete button will be there. | | | |
| Trigger: A case needs to be updated. | | | |
| Relationships:  Association:  Includes: 1.2.1  Extends:  Generalisation: | | | |
| Normal Flow of Events:   1. The need has arisen to update a case. 2. The Solicitor or Secretary enters the case reference number. 3. The relevant case will be displayed to screen. 4. The user will click into this case. 5. The user can then make the necessary amendments. | | | |
| Subflows:  At step 5, the user will be able to amend client details, for example if they have changed their phone number.  To do this, the user presses “Amend Case Details” button.  The system will set all fields except Case Reference Number to editable.  The user can then edit the detail(s) as required. | | | |
| Alternate/Exceptional Flows:  Step 2 will require the system to validate the user’s input. Data validation rules can be found under Data Validation.  After step 3, the user can delete the case if necessary by pressing the Delete button (see use case description 1.2.3). | | | |
|  | | | |

#### Use Case: Record Billable Activities (1.2.2)

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name: Record Billable Activities (2.2.2) | | ID: 1.2.2 | Importance Level: Very High |
| Primary Actor: Employee | **Use Case Type:** Essential, Detail | | |
| Solicitor: Needs to record billable activities as this is the firm’s main source of income.  Secretary: Wants to enter the billable activity details quickly and accurately for good organisation and smooth workflow. | | | |
| Brief Description: This is an important use case which allows the client to keep track of the monies owed to the firm. In order to successfully record a billable activity, the user will enter a valid case reference number and click into the case, from here the system will allow the user to enter the specific activity along with the time spent on that activity. | | | |
| Trigger: A case reference number has been entered and “Record Billable Activity” button pressed. | | | |
| Relationships:  Association:  Includes: 1.2.2.1, 1.2.2.2  Extends:  Generalisation: | | | |
| Normal Flow of Events:   1. The user enters a valid case reference number and clicks into the case. 2. The user selects a specific activity from a dropdown list. 3. The user will enter the time spent on that activity. Elaboration should be sought over how the user would like time spent to be entered. 4. After entering valid data, the user will press a “Confirm” button. 5. The amount for this activity will be displayed. 6. The total amount owed for this case will be displayed. See wireframe for Recording Billable Activities for visual representation. | | | |
| Subflows:   * If the reference number entered at step 1 refers to an Immigration case, the dropdown list of billable activities should also include Home Office Decision Appeal. * Step 2 will display a dropdown list of billable activities which will be one of: First Letter to Client; Perusing Case History; Reviewing Evidence; Meeting with Client; Drafting Legal Documentation; or Hearing at Court. * The time spent on an activity will be clarified with the client, but appropriate validation should be performed and Data Validation will be updated once clarification has been received. | | | |
| Alternate/Exceptional Flows:  Appropriate error message is displayed if data is invalid. | | | |
|  | | | |

#### Use Case: Calculate Monies Owed (1.2.6)

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name: Calculate Monies Owed (2.2.6) | | ID: 1.2.6 | Importance Level: Very High |
| Primary Actor: Employee | **Use Case Type:** Essential, Detail | | |
| Solicitor: Need to know how much to invoice a client.  Secretary: Must ensure invoicing is accurate. | | | |
| Brief Description: This use case describes that system will calculate the figure of monies owed to the firm that is generated by one case. | | | |
| Trigger: The user has entered valid activity details and pressed “Confirm”. | | | |
| Relationships:  Association:  Includes:  Extends:  Generalisation: | | | |
| Normal Flow of Events:   1. The user has pressed "Confirm" button after entering valid activity details 2. The system will calculate the figure owed for the activity entered. 3. The system will calculate and display the total figure owed to the firm by the client. | | | |
| Subflows:   * Step 2 will require the figure to be computed as described below.   The client charges for billable activities based on hourly rates. These hourly rates were obtained in a client interview, but are also included in the Data Validation section for ease of reference. An example is as follows: If Mr Wullyum McRae (Criminal Solicitor) charges £50 an hour and spends 30 minutes meeting a client, that fee charged will be £25. If Mr Wullyum McRae then spends 10 minutes drafting legal documentation, the fee charged will be £12.50. The total monies owed by this case's client is £37.50. | | | |
| Alternate/Exceptional Flows:  It was identified in the preceding use case description that some clarification is needed regarding how time spent should be entered. Once these details are received, this will be updated and accurate rules will be included under Data Validation.  Updated 06/03/2023: Time spent will be entered in units. See Meeting Number 3 Minutes agenda item 6, and Data Validation. | | | |
|  | | | |

#### Use Case: Delete Case (1.2.3)

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name: Delete Case (2.2.3) | | ID: 1.2.3 | Importance Level: Very High |
| Primary Actor: Employee | **Use Case Type:** Essential, Detail | | |
| Stakeholder and Interests:  Solicitor: Has created a case in error or the client has a new Solicitor at a different firm.  Secretary: Has created a case in error or has been asked to delete a case by Solicitor. | | | |
| Brief Description: The firm need to have the ability to remove a case from the system. If a client has changed to a different Solicitor and the firm no longer need to retain their client's data, they will need to be able to delete the case from the system's storage. | | | |
| Trigger: User has searched for a case and pressed "Delete" button. | | | |
| Relationships:  Association:  Includes: 1.2.3.1  Extends:  Generalisation: | | | |
| Normal Flow of Events:   1. The need has arisen to remove a case from the system. 2. The Solicitor or Secretary enters the case reference number. 3. The relevant case will be displayed to screen. 4. The user will click the "Delete" button. 5. The case is removed from the system's storage. | | | |
| Subflows:  Step 4 should trigger an information message being displayed to screen. The information message should read:  "You are about to delete this case. Press Confirm to continue, or Cancel to return." | | | |
| Alternate/Exceptional Flows:  Step 2 will require the system to validate the user’s input. Data validation rules can be found under Data Validation. | | | |
|  | | | |

#### Use Case: View Caseload (1.3)

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name: View Caseload (2.3) | | ID: 1.3 | Importance Level: Very High |
| Primary Actor: Employee | **Use Case Type:** Essential, Detail | | |
| Stakeholder and Interests:  Solicitor: Needs to establish if caseload is at maximum limit to decide whether it is possible to accept a new client.  Secretary: Needs to quickly access an overview of a case for drafting of various legal documentation. | | | |
| Brief Description: An Employee has received an enquiry from a potential client and wishes to view all cases the firm is responsible for at this current time to establish whether a Solicitor has availability to take on a new case. The user will press the “View All Cases” button and the system will retrieve and display all cases.  The user may wish to refine their search to all cases by type. Therefore, the user will press either the “View Criminal Cases”, “View Immigration Cases”, or “View PI Cases” button. The system will then retrieve and display all cases by type. | | | |
| Trigger: User presses "View All Cases" button | | | |
| Relationships:  Association:  Includes: 1.3.1, 1.3.2, 1.3.3  Extends:  Generalisation: | | | |
| Normal Flow of Events:   1. If the firm has received a new customer enquiry asking for legal advice, subflow 1 is followed. 2. If the Secretary is drafting legal documentation in preparation for a Solicitor attending a Hearing at Court, subflow 2 is followed. | | | |
| Subflows:  Subflow 1  If a Solicitor is reviewing a new customer enquiry asking for legal advice:  The Solicitor can't remember how many cases the firm currently has ongoing.  The Solicitor presses "View All Cases" to establish if there is availability for a new client.  Subflow 2  A Secretary is drafting legal documentation in preparation for Solicitor attending a Hearing at Court.  The Secretary can't remember the client's name and case reference number.  The Secretary presses the correct button (either "View Criminal Cases", "View Immigration Cases", or "View PI Cases" button).  The system returns and displays all cases dependant on the user's selection. | | | |
| Alternate/Exceptional Flows: | | | |
|  | | | |

#### Use Case: Log In (2)

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name: Log In (1) | | ID: 2 | Importance Level: High |
| Primary Actor: Employee | **Use Case Type:** Essential, Detail | | |
| Stakeholder and Interests:  Solicitor: Needs to log in to manage their cases.  Secretary: Needs to log in to do their job properly. | | | |
| Brief Description: An Employee has arrived for a long day at work and in order to manage the firm's caseload efficiently, the Employee has to log in to the Case Management System. | | | |
| Trigger: The Employee has arrived for a long day at work. They initiate the Case Management System. | | | |
| Relationships:  Association: Employee  Includes: 2.2  Extends: 2.1  Generalisation: | | | |
| Normal Flow of Events:   1. The user has initiated the Case Management System. 2. The system prompts the user to enter a username and password. 3. The user will enter their credentials. 4. The system will validate the user's input. 5. If the staff member is brand new to the firm, subflow 1 will be followed. | | | |
| Subflows:  Subflow 1  The new staff member's line manager will add them to system.  The system will generate a username (see FR 1.1 in Action Plan Report) by concatenating user’s first name with a dot and the user’s last name.  The username should be in the format: firstname.lastname (see NFR 1 in Action Plan Report).  Step 4 should validate that the username is in the format mentioned above, and that the password contains a minimum of 12 characters, and include at least 1: special character, number, uppercase letter (see NFR 2 in Action Plan Report). | | | |
| Alternate/Exceptional Flows: If log‑in credentials are invalid, error message 1 should be displayed. This can be found under Data Validation. The user can then decide to return to step 2 or to exit the system. | | | |
|  | | | |

#### Use Case: Generate Report (3)

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name: Generate Report (3) | | ID: 3 | Importance Level: Medium |
| Primary Actor: Employee | **Use Case Type:** Detail | | |
| Stakeholder and Interests:  Solicitor: Wants to establish how the firm is performing.  Secretary: Thinks it's a nice feature to have. | | | |
| Brief Description: The firm owner wants to have a look at how business is performing overall. They want to do some business analysis for a staff meeting and would like to be able to report to the team the number of total open cases; the number of cases opened in a specified time period; number of cases closed in a specified time period; and the number of cases allocated to a specific Solicitor. | | | |
| Trigger: The user has pressed the "Generate Report" button. | | | |
| Relationships:  Association: Employee  Includes: 3.1  Extends:  Generalisation: | | | |
| Normal Flow of Events:   1. The user wishes to study some data about the firm. 2. The user presses the "Generate Report" button. 3. The user must select the report type to be generated from a dropdown list. 4. If the number of total open cases is selected, subflow 1 is performed. 5. If the number of cases opened in a specified time period is selected, subflow 2 is performed. 6. If the number of cases closed in a specified time period is selected, subflow 3 is performed. 7. If the number of cases allocated to a specific Solicitor is selected, subflow 4 is performed. | | | |
| Subflows:  Subflow 1   * A case is considered "open" simply if it lies in the system's storage. * The system should display the total number of cases.   Subflow 2   * The system should prompt the user to provide a start date and end date. * The system should then calculate how many new cases have been added to the system in that timeframe.   Subflow 3   * The system should prompt the user to provide a start date and end date. * The system should then calculate how many cases have been removed from the system in that timeframe.   Subflow 4   * The system should prompt the user to enter a Solicitor's name. * The system will then validate that the input contains letters only. * The system will then retrieve and display cases which the specified Solicitor is responsible for. | | | |
| Alternate/Exceptional Flows: If a Solicitor name is entered for subflow 4 and no cases are found, the system should display the following information message:  "Sorry, no cases have been found for " + solName + ". Please ensure a valid name is entered." | | | |
|  | | | |

Note: in meeting number 3 with the client, it was highlighted that this use case can be omitted in the development of the evolving system for now. The client has stated that this use case can be implemented at a later date.

### Data Validation

The preceding use case descriptions make it clear that the evolving system will implement a variety of data validation. The Action Plan report makes reference to some required validation, for example Non‑Functional Requirement 4 states that if log‑in credentials are invalid, the message “Invalid log‑in details entered!” should be displayed. However, we have now carried out prioritisation of the use cases which is slightly different to the prioritised list of Functional Requirements in the Action Plan report: Log In (FR 1) is no longer our top priority (though still important). It is now established that “Manage Cases” is the highest priority use case. Regardless of priority, the use cases require the human user to input a variety of data. We do not want the system to accept any old rubbish, so in attempt to make sure the evolving system will store acceptable data only, I have created the table below which should be used as a guide to validate user input.

**This subsection must be read in conjunction with the fully dressed use case descriptions which make reference to various data validation rules that are detailed below.**

| **Data Validation** | | | |
| --- | --- | --- | --- |
| **Field** | **Validation** | **Notes (if applicable)** | **Error No.** |
| Username | Format must be firstname.lastname |  | 1 |
| Password | See notes | Minimum criteria:   * minimum of 12 characters * at least 1 special character * at least 1 number * at least 1 uppercase character | 1 |
| Case Reference Number | See notes. | When searching, user must enter case reference number in format:   * [three integers]-[first three letters of case type] * Potential for regular expression. * First three characters – digits, 0-9 * Followed by a dash * Followed by either CRI, IMM, or PIN. | 3 |
| Case Type | CRI, IMM, or PI from dropdown. Option must be selected. |  | 2 |
| Case Title | Alphabetical characters only.  See notes | A case title should be in the format:  [client surname] v [opponent] | 4 |
| Case Description | Field cannot be empty |  | 2 |
| Client first name | Field cannot be empty |  | 2 |
| Client first name | Cannot contain numerical values. |  | 5 |
| Client last name | Field cannot be empty |  | 2 |
| Client last name | Cannot contain numerical values. |  | 5 |
| Client DOB | Field cannot be empty |  | 2 |
| Client DOB | See notes | Format required is DD/MM/YYYY  DD is day number of the month and must contain 2 digits.  MM is the month number and must contain 2 digits. The first digit must be either 0 or 1.  YYYY is the year number and must be 4 digits long.  If YYYY modulo 4 is 0 and MM is 02, the upper bounds of DD is 29.  **We need specific format for DOB, so would make sense to** **use regular expression to match DOB pattern.**  String class contains method called matches(String regex) that returns true if a String matches the given regular expression – allows us to determine exactly what the DOB input should match against and we can specify multiple values.  Pattern class allows us to define a regular expression; compiled regex pattern can speed up the system if pattern is used frequently.  Matcher class can store possible match between a Pattern and a String. After initialising, we can use .matches() which returns true if the regex given in the Pattern declaration matches the String.  **Potential** pseudocode for validating DOB format: | 6 |
| Client DOB | See notes | Above takes care of validating the format, but system should also check that values are valid too. Values outside of range of days and months are invalid. Potential to use DateFormat class to validate DOB value. Provide simpleDate pattern to define the order of date elements. The user’s input for DOB field can be parsed. Try catch will define the Boolean return value. Potential pseudocode for validating DOB value: | 7 |
| Client address (line 1) | Field cannot be empty. See notes. | Address may contain alphanumeric data; data too varied for common regular expressions as some addresses start with house number, some start with flat number (alphanumeric, like 1a), some are simple names (like The Beach House). | 2 |
| Client address (line 2) | Field cannot be empty. See notes. | Address should contain additional information such as floor number, street name, town name, county. | 2 |
| Client Postcode | See notes. | All postcodes will be UK based.  Basic rules are:  Minimum 5, maximum 7 alphanumeric characters.  Only separator allowed is a single space.  Potential for regular expression as postcodes follow a common pattern. Some research shows that “the rules covering which characters can appear at particular positions are rather complicated and fraught with exceptions” (Goyvaerts and Levithan, 2012). Goyvaerts and Levithan offer the following regex to validate a UK postcode:  ^(?:(?:[A-PR-UWYZ][0-9]{1,2}|[A-PR-UWYZ][A-HK-Y][0-9]{1,2}↵  |[A-PR-UWYZ][0-9][A-HJKSTUW]|[A-PR-UWYZ][A-HK-Y][0-9]↵  [ABEHMNPRV-Y])●[0-9][ABD-HJLNP-UW-Z]{2}|GIR 0AA)$ | 8 |
| Client Phone Num. | Digits only. | Must start with a 0 and must be 10 digits long. | 9 |
| Enquiry Type | CRI, IMM, or PI from dropdown. Option must be selected. |  | 2 |
| Solicitor | Field cannot be empty. | A Solicitor must be selected from the dropdown list when adding new case. | 2 |
| Billable Activity | Option must be selected from dropdown list. Cannot be empty. |  | 2 |
| Time spent | Field cannot be empty |  | 2 |
| Time spent | See notes. | Minimum “units” value is 0.25. Maximum “units” value is 10.  Acceptable values between 0.25 and 10 are values incremented by .25. For example, 0.5, 0.75, 2, 2.25, 2.50 are all acceptable values. | 10 |
| New Employee Job Title | Field cannot be empty |  | 2 |
| Multiple Invalid Fields | See notes. | If multiple fields are invalid or illegal, the field boxes should be outlined in red. | 11 |

#### Error Messages

|  |  |  |
| --- | --- | --- |
| **Error Messages** | | |
| **No.** | **Message** | **Description** |
|  | Invalid log-in details entered! | Username or password does not meet criteria. |
|  | Field cannot be blank. | A required field has been left empty. |
|  | Please enter a valid case reference number. | Case reference number not n valid format. |
|  | Case title must be in format [client surname] v [opponent] using alphabetical characters only. | Case title contains illegal characters or omits the “v”. |
|  | Name field cannot contain numbers. | A person’s name cannot contain digits. |
|  | DOB invalid. DOB must be entered in format DD/MM/YYYY. | DOB format incorrect or illegal. |
|  | DOB must contain valid numeric values. | DOB contains illegal values. |
|  | Postcode invalid. | Postcode is illegal. |
|  | Phone number invalid. Must contain digits only and cannot contain whitespace or special characters. | Phone number must not contain letters. |
|  | Enter valid value for time spent. | Value invalid. |
|  | Invalid fields. Please review. | If multiple fields are invalid. |

## The Domain Model

To ultimately build a successful software solution for the client, we need to thoroughly decompose the overall problem domain. To further analyse the problem domain in an object‑oriented manner, we generally decompose the problem by entities. One of the key tasks in analysing the problem domain is the identification of conceptual classes. In order to progress to the conceptual model, it is helpful to perform a second natural language analysis on the use case descriptions. As the UP is iterative in nature, the Action Plan report submitted previously has initiated an analysis of the domain in providing a natural language analysis and a top‑level conceptual model.

### Natural Language Analysis

A natural language analysis was performed on the initial project brief in the Action Plan Report which resulted in the identification of several potential entities and actions. From this and various other Inception activities (such as initial planning models and background research), we produced a top‑level conceptual model (Figure 8). Larman states that the “*domain model is a visualization of noteworthy domain concepts and vocabulary. Where are those terms found? In the use cases.*” (Larman, 2005).

With the development of The Use Case Model, we can now perform a second linguistic analysis on the fully dressed use case descriptions contained in The Use Case Model. This is a key strategy I will use to iterate over identifying conceptual classes.

Figure Top-level Conceptual Model

**CLIENT**

**CLIENT**

pertains to

pertains to

**CASE**

**CASE**

**CASE DETAILS**

**CASE DETAILS**

contains

contains

**CASELOAD**

**CASELOAD**

**USER**

**USER**

**REPORTS**

**REPORTS**

**EMPLOYEE**

**EMPLOYEE**

consists of

consists of

managed by

managed by

is a

is a

generates

generates

The yellow‑coloured highlighting represents the nouns/entities, and the purple‑coloured highlighting represents verbs/behaviours. I have summarised the common terms identified below. The analysis has been performed on three of the fully dressed use case descriptions: "Manage Cases", "Log In", and "Generate Report". I have limited this activity to these specific use case descriptions because they encompass all other use cases, and as use case diagrams and descriptions reflect functional requirements very closely, this will capture all the necessary requirements to build a successful system. It is also worth remembering that a linguistic analysis has already been performed and is included in the Action Plan report submitted previously, so it is not worth the project team's time to repeatedly perform textual analyses; instead it would be sensible to start thinking of using our time to dip in to the UP's implementation workflow and begin concurrent programming after this and the remaining analysis and design activities.

#### Potential Entities

| **Use Case (ID)** | **Term** | **No. of Occurrences** | **Comments** |
| --- | --- | --- | --- |
| Manage Cases (1) | Cases/Case | 10 | A case is handled by a Solicitor. Potential for enumeration depending on implementation. |
| Employee | 3 | Refers to all client’s employees – Solicitors and Legal Secretaries. Action Plan Report suggested this may be a potential superclass. However both a Solicitor and secretary should have access to same functionality, so there may be no need for a superclass here. |
| Workload | 2 | Relates to the business case for the new system. The evolving solution aims to help in controlling the firm's workload. |
| Manner | 1 | The manner in which the firm want to organise their workload – in a structured and efficient manner. Redundant term – meaningless to system functionality. |
| Solicitor | 8 | An employee of the firm; responsible for legal cases. |
| Firm's Caseload | 1 | The cases the firm deals with currently. |
| Firm | 2 | Refers to our client, McRae & Dick Solicitors. |
| Customer enquiries | 1 | Part of business process – a case will not be created before an enquiry has been reviewed. System does not have to deal with customer enquiries, but will have to allow user to specify enquiry type (CRI, IMM, or PI) when adding a new case. |
| Legal advice | 1 | What the firm's client have contacted them for. This is the firm's "product" that they sell – they provide legal advice and get paid for it. |
| Matters | 1 | Another term for a "case"; irrelevant to the evolving system. |
| New client | 1 | If a firm is creating a new case, they have received an enquiry from a new client. |
| New Case | 2 | If the enquiry is reviewed and accepted by a Solicitor, a new case will be created. |
| System | 2 | The overall scope – a case management system is being developed. |
| Specific case | 1 | Every case will have a unique identifier specific to that one case. |
| Details/Client details | 6 | Various client, case, and Solicitor data that system should store as identified in Action Plan Report (see FR 2 and NFR 6, 7, and 8). |
| Legal documentation / documents | 2 | Drafting this is a billable activity (see NFR 15.5 in Action Plan Report), |
| Client's name | 1 | Part of client data to be stored. |
| New client enquiry | 1 | A new client is seeking legal advice and has contacted the firm. |
| Subflow 1 | 2 | The basic process to be followed if a Solicitor has decided to accept a new case. |
| Secretary | 1 | Will be one of the main end users of the evolving system. |
| "Add New Case" button | 1 | This button should be included and obvious in the main navigation panel. |
| Billable activities/activities | 3 | The thing that makes the Solicitor money. |
| Subflow 2 | 2 | The basic process to be followed if a billable activity has been performed. |
| Client | 2 | The person a case pertains to. |
| New phone number | 1 | Refers to a simple example of why case details may have to be updated. |
| Subflow 3 | 2 | The basic process to be followed if access to a specific case is needed. |
| Search bar | 2 | Very important; should be in obvious location and should allow a user to type in a case reference number. |
| User | 3 | The human that will use the evolving system once complete. |
| Case reference number | 1 | The unique identifier of any one specific case. |
| Error message | 1 | Displayed if invalid data is entered. Further details on data validation can be found under subsection Data Validation. |
| Data | 1 | Refers to various user input. All data should be validated as specified under subsection Data Validation. |
| Field/invalid fields | 2 | The box where a user will enter data. Invalid input will result in a field being outlined red. |
| Log In (2) | Employee | 4 | Repetitive term. See comment above in Manage Cases. |
| Cases | 1 | Repetitive term. See comment above in Manage Cases. |
| Job | 1 | Simply refers to the user's job in the legal firm. Not important to the functionality of the system. |
| Day | 2 | The employee's day at work will consist of using the new system. |
| Work | 2 | Another term for "job". |
| Firm's caseload | 1 | Repetitive term. See comment above in Manage Cases. |
| Case Management System | 3 | Identifies the overall scope of the project. |
| User | 4 | Repetitive term. See comment above in Manage Cases. |
| Username | 4 | One part of log-in credentials. A mandatory field for a user to access the system. Format must be: firstname.lastname. |
| Password | 2 | The other part of log-in credentials. Also a mandatory field for a user to access the system. Must contain at least 1: special character, number, uppercase letter. |
| Credentials | 1 | The username and password together. Credentials must be valid to gain access to the system. |
| System | 4 | The system being developed. A redundant term as has no meaning to the functionality of the system. |
| User's input | 1 | The data which a user is required to enter. In this case, the log-in credentials. |
| Staff member | 1 | An employee of the legal firm. If a staff member is new to the firm, system must generate them a username (firstname.lastname). |
| Subflow 1 | 2 | The process to be followed if the user is a new member of staff. |
| Format | 2 | Simply refers to what constitutes a valid username and password. See comments for username and password, along with subsection Data Validation. |
| Special character | 1 | A password must contain at least 1 special character. |
| Number | 1 | A password must contain at least 1 number. |
| Uppercase letter | 1 | A password must contain at least 1 uppercase letter. |
| Log-in credentials | 1 | The data unique to the user that allows access to the system if valid. |
| Error message | 1 | The text to be displayed if log-in credentials are not valid. Specific error message can be found in subsection Data Validation. |
| Generate Report (3) | Report | 1 | What the user should be able to generate. |
| Employee | 2 | Repetitive term. See comments above in Manage Cases. |
| Firm | 2 | Repetitive term. See comments above in Manage Cases. |
| Nice feature | 1 | The ability to generate reports would be a "nice feature" for the firm to have. Suggests a lesser priority, but it would probably make for a better user experience. |
| Firm owner | 1 | Mr Wullyum McRae. A redundant term, all users will have the same functionality available to them. |
| Business | 1 | The legal firm. |
| Staff meeting | 1 | The generation of reports will be very useful in staff meetings – data will be retrieved quickly and programming the system to analyse the data is more time efficient that figuring it out manually. |
| Team | 1 | All of the staff that work for the client. |
| Number of total open cases | 2 | Simply refers to the number of cases that are stored in the system. If a case is not open, there is no requirement to store the data. As this is a brand‑new system, any "closed" cases will not be stored. It may be a good idea to future proof the system by including a case status field as part of case details. This would allow the firm to "reopen" a case if necessary. **Discuss with client if they would like case status to be stored in system.** |
| Number of cases opened | 2 | A report to be generated – will require user to set the desired time period. **Seek clarification on how the client would like to enter the start and end date.** |
| Specified time period | 4 | Relates to the two types of reports that require the user to set the specified time period. See above for comments. |
| Number of cases closed | 2 | Similar to "Number of cases opened", suggests the system will have to store case status data as well. Included in agenda for meeting 3 with client. |
| Number of cases allocated | 1 | Another report type – suggests system should be able to retrieve cases by Solicitor. User to enter Solicitor's name. **Potential for this to be a key in a HashMap. Value may be a Case object.** |
| Specific Solicitor | 2 | Relates to above term, "number of cases allocated". Suggests the system should be able to sort the caseload by Solicitor. HashMap data structure suggested above. Solicitor name may act as key. |
| User | 7 | Repetitive term. See comment above in Manage Cases. |
| "Generate Report" button | 2 | System must be designed with this button present in the navigation menu at all times. |
| Data | 1 | Simply refers to data about the firm which should be generated by the system. |
| Report type | 1 | Can be either: number of total open cases; the number of cases opened in a specified time period; number of cases closed in a specified time period; or the number of cases allocated to a specific Solicitor. |
| Dropdown list | 1 | User should be able to select the type of report from a dropdown list. Promotes ease of use. |
| Subflow 1 | 2 | The process to be followed if user selects number of total open cases. |
| Subflow 2 | 2 | The process to be followed if user selects number of cases opened in a specified time period. |
| Subflow 3 | 2 | The process to be followed if user selects number of cases closed in a specified time period. |
| Subflow 4 | 2 | The process to be followed if user selects number of cases allocated to a specific Solicitor. |
| "open" | 1 | Suggests a case has a status – open or closed. Currently, a case is considered "open" if it lies in the system's storage. However, as pointed out above, the firm may want to retain cases in the system that have been closed. The client will be asked if they wish to be able to set a case's status. |
| System's storage | 1 | The data in the back end of the system. |
| Start date | 2 | A start date is required for select report types. |
| End date | 2 | An end date is required for select report types. |
| Cases | 3 | Repetitive term. See comment above in Manage Cases. |
| Timeframe | 1 | The start and end date set by the user for select report types. |
| Solicitor's name | 2 | If the user wants to view the number of cases allocated to a specific Solicitor, the Solicitor's name must be entered. Suggested above this data may be a potential key in a HashMap data structure. |
| Letters | 1 | Valid Solicitor's name will contain letters only. |
| Information message | 1 | The text to be displayed if no cases are found for the Solicitor name entered. |

#### Potential Behaviours

| **Use Case (ID)** | **Term** | **No. of Occurrences** | **Comments** |
| --- | --- | --- | --- |
| Manage Cases (1) | Manage | 1 | Refers to all basic CRUD functions of a legal case in the system. Includes creating, retrieving, updating, and deleting a case. |
| Organise(ing) | 3 | Suggests data should be organised efficiently; entering a case's unique reference number should retrieve the correct case, for example. |
| Support | 1 | Refers to the job of a Secretary who performs tasks to ease the Solicitor's workload. |
| Receive(d) | 2 | Simply refers to the firm receiving customer enquiries. This is the start of the business process of creating a new case. |
| Seeking | 1 | The firm deals with clients who are seeking out legal advice. Irrelevant to the functionality of the system. |
| Accept | 1 | Does not refer to the system; refers to the Solicitor accepting a new client. The Solicitor makes the decision to accept a new case. |
| Add | 1 | The user should be able to add a new case to the system; a crucial aspect of the evolving system. One of the basic CRUD functions. |
| Progress(ed) | 3 | A Solicitor will make progress on a case. Not relevant to the system's functionality, but it is a trigger for having to update a case as progress usually suggests a billable activity has been performed. |
| Update | 1 | Another basic CRUD function. The user must be able to update a case, for example in recording a billable activity and updating the total figure of monies owed to the firm. |
| Draft | 1 | Part of a business process – Solicitor or Secretary will draft legal documentation in preparation for a Hearing at Court, for example. |
| Look-up | 1 | The user must be able to "look up" a specific case to make an amendment, for example. Term can be used interchangeably with "retrieve" of CRUD functions. Non-technical word for retrieve function. |
| Fill | 1 | Solicitor/Secretaries have to fill in various legal documentation. Accuracy is crucial as these are important documents. To fill in legal documents accurately, the case management system must store accurate case/client/Solicitor details – if a case is updated by the user, for example, the system absolutely must update the case accordingly and store the correct data in the correct place. |
| Followed | 2 | Just refers to the subflows that are followed dependant on the normal flow of events step. |
| Changed | 1 | Refers to changes outside of the system, but system must be able to cope with these changes. For example, if a client changes their phone number, the user must be able to change this detail on the system easily. |
| Obtain(ed) | 2 | It is common in this business to have to obtain specific details of a case. For example, the Secretary will not have every single client name and address memorised, so obtaining (or, retrieving) such details through the system is important. |
| Decided | 1 | Refers to a Solicitor deciding to accept a new case. Is part of the business process of taking on a new case. |
| Email | 1 | Refers to the Solicitor emailing the relevant details to Secretary for creation of a new case. Part of the business process of creating a new case. |
| Press | 1 | The user will press the "Add New Case" button. Refers to user interaction. They will press a button to access desired functionality. |
| Record | 1 | Refers to the recording of billable activities. Very important that the system can track these activities as it will speed up the overall invoicing process for the Secretary as clarified in meeting number 2 (see last paragraph of agenda item 7 of minutes). |
| Locate | 1 | Refers to the user locating the search bar. The system should be designed so that this search bar is immediately obvious to the user. |
| Enter | 1 | The user will enter a variety of data; in this use case description "enter" refers to entering a case reference number in the search bar. |
| Displayed | 1 | The text of an error message must be displayed to screen when necessary. |
| Outlined | 1 | If input is invalid, the relevant text field should be outlined in red. |
| Log In (2) | Log In | 4 | Refers to the user logging in to the system; one of the main use cases identified from early in Inception. A high priority use case. |
| Manage | 2 | Repetitive term. See comment above in Manage Cases. |
| Arrived | 2 | Just refers to the Employee arriving for work, which is probably when they will log in to the system. Not relevant to the functionality of the system, just context for the trigger and description. |
| Initiate(d) | 2 | This is the first step before logging in – they will initiate the system to start their session. |
| Prompts | 1 | Refers to a prompting the user to enter their log‑in credentials. Suggests screen should be designed to be intuitive – a prompt is essentially a cue to the user to do something. For example, a help tip in the text field that reads "Enter username". |
| Enter | 2 | The user must enter their credentials. Suggests system will accept user input, which has been clear from the outset. |
| Validate | 2 | The system will have to validate the user's log‑in credentials. Validation rules can be found in subsection Data Validation. |
| Followed | 1 | Repetitive term. See comment above in Manage Cases. |
| Generate | 1 | Refers to the generation of a username for a new member of staff. **But how will the system know when to generate a new username?** It might be worth suggesting to the client to also include a "Our Staff" button in the navigation panel, where a current Employee can add a new member of staff and ask the system to generate a new username. **Include this point in agenda for meeting 3.** |
| Contains | 1 | Simply refers to what a password should be made up of: minimum 12 characters (as specified in NFR 2 in Action Plan Report). |
| Include | 1 | Simply refers to what a password should include: at least 1 special character, number, and uppercase letter (as specified in NFR 2 in Action Plan Report). |
| Displayed | 1 | Repetitive term. See comment above in Manage Cases. |
| Found | 1 | Simply refers to where validation rules can be found (subsection Data Validation). |
| Decide | 1 | Refers to the user's choices if invalid log‑in credentials are entered: return to log‑in screen, or exit the system. |
| Return | 1 | The user can return to the log‑in screen if invalid credentials entered. |
| Exit | 1 | Exit the system; self‑explanatory. |
| Generate Report (3) | Generate(d) | 2 | Refers to the generation of reports; a nice feature of the system, but not absolutely crucial to the overall functionality. |
| Wants | 2 | The Solicitor wants to view the data in the reports; note use of "wants" and not "needs", suggests it would enhance the user's experience of the system, but not critical to the system satisfying its objective of helping the firm manage their workload more efficiently. |
| Establish | 1 | Refers to establishing how the firm is performing – the data in the reports will help the Solicitors see if the firm has an active caseload or if business is struggling. |
| Performing | 1 | As above – how the business is performing in terms of caseload. |
| Thinks | 1 | The Secretary's opinion; irrelevant to functionality of system. |
| Have | 1 | The Secretary thinks this use case will enhance their experience of the system. |
| Look | 1 | Another term for viewing the data. Suggests the reports should be clear and simple so they are easy to look at and easy to interpret. |
| Business analysis | 1 | The owner wants to do some business analysis to report back to the team at a staff meeting. |
| Press(es)/(ed) | 2 | The pressing of buttons; self‑explanatory. |
| Wishes | 1 | The user's desire to view data. |
| Study | 1 | The reports allow the user to study data about the firm. |
| Select(ed) | 5 | The system should be design so the user can select a report; option should be selected from a dropdown for ease of use. |
| Performed | 4 | Just refers to the subflow that is performed dependant on normal flow of events step. |
| Considered | 1 | Refers to what distinguishes a case as "open". A case is considered "open" simply if it lies in system's storage. Already highlighted a query here (see Potential Entities subsection) about case status and is included in agenda for meeting number 3 with client. |
| Lies | 1 | Refers to data that is located in the back‑end of the system. |
| Display | 2 | The relevant report should be displayed to screen. |
| Prompt | 3 | Repetitive term. See comment above in Log In. |
| Provide | 2 | Refers to the user specifying the start and end date of which report should reflect. |
| Calculate | 2 | The system should be able to calculate how many cases have been added or removed from system in a specified time period. |
| Added | 1 | Refers to cases that have been added to the system. |
| Removed | 1 | Refers to cases that have been removed from the system. |
| Enter(ed) | 2 | Repetitive term. Variety of data will be entered by the user. |
| Validate | 1 | Repetitive term. All user input should undergo relevant validation. Rules can be found under subsection Data Validation. |
| Contains | 1 | Repetitive term. Refers to what user input should contain. |
| Retrieve | 1 | Basic CRUD function – system should retrieve cases based on specified Solicitor name. |
| Found | 1 | Repetitive term. Simply refers to data the system finds based on a given condition. |

This linguistic analysis was performed **prior to** meeting number 3 with the client. This meeting highlighted that the use case “Generate Report” (ID 3) can be omitted in the development of the evolving system for now. The client has stated that this use case can be implemented at a later date. I have retained the analysis performed on this use case to save time and labour when the time comes to implement this requirement.

In meeting number 3, the client also emphasised that the use case “Record Billable Activities” (ID 1.2.2) is an important requirement (see Meeting Number 3 Minutes, agenda item 6). Given the client’s need to record monies owed accurately, I have decided to perform a linguistic analysis on this use case too. The results are shown within the Summary of meeting number 3 with the client.

### Client Feedback

After carrying out what Larman refers to as one of the most valuable exercises in understanding the behaviour of the evolving system – the use case descriptions – it is sensible to update the client with the further analysis that has been performed and to seek answers to questions that have arisen as a result. As analysis has been initiated in the previous Action Plan report, we have a grounding to begin developing a basic view model of the evolving system. As such, the following meeting agenda and meeting minutes reflect that these activities have been performed concurrently. Specifically, the client was presented with the fully dressed use case diagram along with a variety of wireframes. I have omitted these visual aids from the following agenda to avoid cluttering this document (the use case diagram can be found under section Fully Dressed Use Case Diagrams and the wireframes can be found under section Wireframes).

The agenda may seem to suggest that the project team are performing a variety of activities that belong to different workflows that the UP supports. A balanced discussion was presented in Section 1 of the Action Plan report regarding the methodologies available and the decision was made to adopt the UP. The Action Plan report placed emphasis on the agile nature of the UP; an approach that is not rigid accommodates for activities relating to different workflows to take place simultaneously. This is why the agenda includes a use case diagram along with a variety of wireframes. The project team are still analysing the problem, but there is enough groundwork done to start designing the view model at the same time. This lets the client see visually how the use cases will transform into their end product and gives them an opportunity to feed back their likes and dislikes about the evolving system. At this point, we really see the benefits of the UP coming through. So far, we have:

* developed accurate requirements / use cases of the evolving system based on our interpretation of the brief (which the client has approved);
* provided the logic of system behaviours based on the requirements and use cases; and
* presented the start of the view model design to the client.

Had the alternative methodology (the Waterfall methodology) been chosen, the client would likely still be waiting on a list of requirements. The regular client interaction that is typical of the UP keeps the project on track as there is ample opportunity for the client to contribute their feedback and opinions.

#### Meeting Number 3 Agenda

The following agenda was issued to the client prior to the third interview.

**Meeting 3 to be held at 09:25 on 01/03/2023**

**Falkirk Campus, Forth Valley College**

|  |  |
| --- | --- |
| **Present:** |  |
|  |  |
|  |  |
| **Apologies:** |  |
|  |  |
| **Chair:** |  |
| **Minute Taker:** |  |

1. **Minutes of Previous Meeting**

All good.

1. **Matters Arising**

No matters have arisen.

1. **Visual Aid (2mins)**

The following diagram represent further analysis of the project.[[1]](#footnote-1)

* Fully dressed use case diagram (PDF also provided)
* Log In wireframe
* Home page wireframe
* Add New Case wireframe
* Record Billable Activities

1. **Navigation (1min)**

We'd like to include an additional page called "Our Staff". This will allow you to add a new Solicitor if the firm decides to employ someone new. It will also allow you to generate a new username for the new member of staff.

1. **Recording Billable Activities (1min)**

Clarification on how user should input the time spent on a task. Eg:

* Two separate fields for minutes and hours? Eg. 0h 59m.
* One field that requires user to enter something like 0.15 for 15 minutes / 0.5 for 30 minutes / 1 for 60 minutes / 1.5 for 90 minutes.

Please specify.

1. **Generating Reports (1min)**

Confirm the following reports are satisfactory:

* number of total open cases;
* the number of cases opened in a specified time period;
* number of cases closed in a specified time period; and
* the number of cases allocated to a specific Solicitor

How would you like to enter the start and end date of specified time period?

Would you like the status of a case to be stored? This would allow you to retain closed cases in the future. Please specify.

1. **Any Other Business (2mins)**
2. **Date of Next Meeting (<1 min)**

To be confirmed.

#### Meeting Number 3 Minutes

The following meeting minutes were prepared after the third client interview.

**Meeting 3 held at 09:25 on 01/03/2023**

**Room 0.021, Falkirk Campus, Forth Valley College**

|  |  |
| --- | --- |
| **Present:** | Daria Vekic (DV), Project Manager |
|  | Susan Gardner (SG), Client |
|  |  |
| **Apologies:** |  |
|  |  |
| **Chair:** | Daria Vekic (DV), Chairperson |
| **Minute Taker:** | Daria Vekic (DV), Minute Secretary |

1. **Minutes of Previous Meeting**

There were no issues highlighted in the minutes for meeting 2.

1. **Matters Arising**

No matters have arisen.

1. **Visual Aids (2 mins)**

DV presented the fully dressed use case diagram (which can be found in Meeting Number 3 Agenda) to the client and explained that the diagram aims to capture all of the most important requirements of the evolving system. The client had a close look at the diagram and there was discussion about the Generate Report use case. DV explained this use case is included as an agenda item (item 7) as the project team would like confirmation that the specified reports are satisfactory. SG stated that this use case is not a priority at this stage and can be omitted for the development of the current evolving system. SG noted that this use case can be implemented in future development. **Action point: DV to update use case description for Generate Report to reflect this.**

1. **Wireframes (5 mins)**

DV then presented various wireframes (which can be found in Meeting Number 3 Agenda) to the client to illustrate the intended structure of relevant pages in the evolving system. SG is happy with the layout the wireframes show and is pleased that when adding a new case, the client can enter all data on one window rather than navigating to separate windows. DV noted that this is likely to progress so that the fields for client name and address will be split over separate fields; also explained that this allows for more control over the data being entered and will ensure validity. Client approved of this decision.

It was also noted that as we already know the details of Solicitors that are employed, the system will already store this, enabling the project team to implement a simple dropdown list for the user to select the responsible Solicitor. DV highlighted this will save the user from repeatedly entering Solicitor details.

DV explained the wireframes will lead into the design of the user interface. This lead into a short discussion about the styling of the user interface and DV said that the initial project brief already stated the company colours are black, white, and green. SG confirmed that as long as the system looks professional, intuitive, and follows this colour scheme then the project team can have some freedom here.

1. **Navigation (1 min)**

DV asked SG for some elaboration on the Log In process, specifically how the system will know when to generate a new username as a "Sign Up" feature is not recommended considering the sensitive nature of the data being dealt with. DV suggested that as this system will be used solely within the firm, an "Our Staff" button be included in the navigation panel. If a new employee joins the firm, a Secretary can then add the new employee's details to the system and then press a "Generate Credentials" button. SG is happy with this suggestion. **Action point: DV to create use case description and updated wireframe to reflect this new use case.**

1. **Recording Billable Activities (1 min)**

There was discussion around the use case Record Billable Activities (2.2.2 in use case diagram) and how the system should accept the "time spent" data. DV proposed two example formats of how time can be entered: two separate fields, one for hours and one for minutes (for example field 1 – 0h; field 2 – 59m); or a single field that requires time in unit measurements (for example, 0.5 would represent a half hour). SG stated the firm charge by units, so the system should implement the latter option where, for example, a half hour is represented as 0.5. The client provided the following breakdown of "time spent" units and would like the following costings to be accepted in the system:

|  |  |
| --- | --- |
| **Unit** | **Time** |
| 0.25 | 15 minutes |
| 0.5 | 30 minutes |
| 0.75 | 45 minutes |
| 1 | 60 minutes |

The table above summarises charges up to and including one hour, but the client would like the system to accept charges up to and including 10 units. Action point: **DV to update: use case description for Recording Billable Activities (ID 1.2.2), Calculate Monies Owed (ID 1.2.6), and Data Validation section accordingly.**

1. **Generating Reports (1 min)**

DV noted the second NLA raised the question of whether the system should store a case status (i.e. whether it is open or closed), which would allow the firm to view historical cases or even reopen a case if needed. However, it was established earlier in the meeting that this use case can be shelved for future development, therefore can be addressed at that time. **Action point: update module description to note that Generate Reports does not have to be implemented for now. Do not omit use case description or NLA as this can be referred to should the client request for this functionality to be implemented at a later date.**

1. **Any Other Business (2 mins)**

No other business.

1. **Date of Next Meeting (<1 min)**

To be confirmed.

#### Summary

The client meeting on 01/03/2023 has revealed some new information. There are no major revisions required to the wireframes presented, but we know now that the client does not want the use case “Generate Report” implemented at this time. The use case description has been retained in this document, but it is noted below the description that this need not be pursued for the time being.

We do need, though, an **additional use case description** for a new use case. To trigger the generation of a username for a new member of staff, a current member of staff must be able to add a new employee to the system. **Therefore, the navigation panel should now include a button for “Our Staff”**. From this, the user will be able to **add a new employee** which will trigger the generation of a new username. This also means the use case description for “Log In” will require a few updates. I have iterated over this use case description and included the updated description below.

##### Updated Use Case Description: Log In (2)

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name: Log In | | ID: 2 | Importance Level: High |
| Primary Actor: Employee | **Use Case Type:** Essential, Detail | | |
| Stakeholder and Interests:  Solicitor: Needs to log in to manage their cases.  Secretary: Needs to log in to do their job properly. | | | |
| Brief Description: An Employee has arrived for a long day at work and in order to manage the firm's caseload efficiently, the Employee has to log in to the Case Management System. | | | |
| Trigger: The Employee has arrived for a long day at work. They initiate the Case Management System. | | | |
| Relationships:  Association: Employee  Includes:  Extends:  Generalisation: | | | |
| Normal Flow of Events:   1. The user has initiated the Case Management System. 2. The system prompts the user to enter a username and password. 3. If the user is a new employee, subflow 1 is followed. 4. If the user is a current employee, subflow 2 is followed. | | | |
| Subflows:  Subflow 1  The new employee will press the “New Employee” link on the Log In screen.  The new employee will be prompted to enter the username that has been given to them.  The new employee will then enter their desired password.  The system will validate this input and inform the user if it is acceptable.  If it matches the password criteria, the user will be able to return to step 2 and log in with their new credentials.  If it does not match the password criteria, the user will re-enter a password to meet the criteria.  Subflow 2  The user – a current employee – will enter their username and password.  The system will validate that the log-in credentials are valid.  The user will be presented with the Home screen.  Steps 3 and 4 should validate that the username is in a valid format, and that the password contains a minimum of 12 characters, and include at least 1: special character, number, uppercase letter (see NFR 2 in Action Plan Report). | | | |
| Alternate/Exceptional Flows:   * If the new employee has entered an unacceptable password, the system will display the relevant error message which can be found under Data Validation. * The system will then prompt the user to enter another password to match the criteria. * If log‑in credentials are invalid, the relevant error message should be displayed. This can be found under Data Validation. The user can then decide to return to step 2 or to exit the system. | | | |
|  | | | |

##### Recording Billable Activities – Natural Language Analysis

The other point to note from meeting number 3 is the importance of the “Record Billable Activities” (ID 1.2.2) use case - see Meeting Number 3 Minutes, agenda item 6. Given the client’s need to record monies owed accurately, I have decided to perform a linguistic analysis on this use case too. The results are shown below.

| **Potential Entities** | | | |
| --- | --- | --- | --- |
| **Use Case (ID)** | **Term** | **No. of Occurrences** | **Comments** |
| Record Billable Activities (1.2.2) | Billable Activities | 5 | The services a Solicitor can charge for. |
| Employee | 1 | An employee of the firm. The primary user of the system. |
| Firm | 2 | Our client. Redundant and repetitive term – not meaningful to system. |
| Source of income | 1 | The chargeable activities are the firm’s main source of income. |
| Billable activity details | 1 | Specific data will be required to accurately record a billable activity. |
| Organisation | 1 | Refers to the organising of the firm’s workflow. The system will add value to the firm by allowing for better organisation of monies owed to them. |
| Workflow | 1 | The workflow should be smooth to allow for successful running of the business. Refers to the successful completion of carrying out tasks. |
| Use case | 1 | Simply refers to the use case being described. Redundant term. |
| Client | 3 | Refers to client as in the legal firm as my client; or a client of the firm. Repetitive term. Already identified client details to be stored in system – see description for Use Case: Enter Client Details (1.1.1). |
| Monies owed | 1 | Refers to the figure of money that is owed to the Solicitor. Calculated value based on time spent figure, entered by user. |
| Valid case reference number | 2 | Must be entered to retrieve the correct case. Validation explained in Data Validation section. |
| Case | 1 | Repetitive term. A potential class/object. |
| System | 1 | Repetitive term. Redundant as simply refers to system as a whole. |
| User | 6 | Refers to the employee who is interacting with the system. |
| Specific activity | 2 | Important. Refers to specific activities that a Solicitor can charge for. Must be selected from dropdown list. |
| Time spent | 4 | Important. This data will be used to calculate monies owed. See costings table in Meeting Number 3 Minutes, agenda item 6). Time spent should be entered in format of units; not hours and minutes. |
| Activity | 3 | The activity the Solicitor has spent time on and must charge for. |
| Case reference number | 3 | The unique identifier of a case. Must be valid in order to successfully retrieve a case. |
| Dropdown list | 3 | We know the activities a Solicitor can perform. These will be displayed in a dropdown list to save user from typing data in repeatedly. |
| Valid data | 1 | All required input must be valid. |
| “Confirm” button | 1 | The button the user must press in order to successfully record the activity and time spent. |
| Total amount owed | 1 | Refers to the figure of money that the Solicitor’s client owes them. Not quite an object, but not a redundant term either. |
| Wireframe | 1 | Simply refers to the wireframe that relates to this use case. Can be found under Wireframes section, subsection Recording Billable Activities. |
| Visual representation | 1 | Simply refers to the wireframe that visually represents how Recording Billable Activities will look. |
| Immigration case | 1 | A type of case. Has one additional activity that is unique to this case type. |
| Home Office Decision Appeal | 1 | The additional activity that is unique to an Immigration case. |
| First Letter to Client | 1 | An activity that should be accepted for any case, regardless of type. |
| Perusing Case History | 1 | An activity that should be accepted for any case, regardless of type. |
| Reviewing Evidence | 1 | An activity that should be accepted for any case, regardless of type. |
| Meeting with Client | 1 | An activity that should be accepted for any case, regardless of type. |
| Drafting Legal Documentation | 1 | An activity that should be accepted for any case, regardless of type. |
| Hearing at Court | 1 | An activity that should be accepted for any case, regardless of type. |
| Validation | 1 | The process the system must perform to establish if input is acceptable or not. |
| Clarification | 1 | Simply refers to the clarification needed on how the user would like to enter time spent. Client confirmed it should be entered in units. See term “Time spent” above. |
| Error message | 1 | Simply the text to be displayed to screen if input is not acceptable. |
| Data | 1 | Refers to user input. |

| **Potential Behaviours** | | | |
| --- | --- | --- | --- |
| **Use Case (ID)** | **Term** | **No. of Occurrences** | **Comments** |
| Record Billable Activities (1.2.2) | Record | 2 | Refers to the storing of the billable activity and time spent. |
| Enter(ed)(ing) | 10 | The user will enter a variety of data. |
| Allows | 1 | Refers to the value the use case will add to the firm. Recording billable activities allows the firm to systematically keep a record of the monies owed to them. |
| Track | 1 | The firm need to track the monies owed to them accurately to enable precise invoicing. |
| Click(ed) | 3 | Refers to the user navigating into a case. |
| Selects | 1 | The user will select an activity from a dropdown list. Required to successfully record an activity. |
| Press | 1 | Refers to the user pressing the “Confirm” button, which will be the trigger to add data to the system. |
| Display(ed) | 3 | The system should display various data to screen. |
| Refers | 1 | Subflow indicates if case reference number entered is an Immigration case, it should include Home Office Decision Appeal. |
| Include | 1 | Refers to the additional activity that an Immigration case should include. |
| Clarified | 1 | Refers to the clarification needed for “time spent” field. |
| Performed | 1 | Refers to the validation processes that should be performed. |
| Updated | 1 | Refers to the relevant data validation that will be updated to reflect the clarification received from the client on how “time spent” should be entered. |
| Received | 1 | Refers to the clarification that we would like regarding time spent. See term “Time spent” in potential entities table. |

From the analysis above, it is clear that the same nouns and verbs are common throughout the use case descriptions. For this reason, I have included a use case description for our new use case – “Add Staff Member” – but have not performed a linguistic analysis on it. For purposes of clarity, I have also included an updated prioritised list of use cases below and will refer to this numbering going forward. The new use case is listed under Log In as in order to add a new employee to the system, a current employee should be able to log in successfully to achieve this use case.

| **Prioritised Use Cases** | |
| --- | --- |
| **Priority** | **Use Case** |
| 1 | Manage Cases (currently use case 2)  This includes 2.1 – 2.3. |
| 2 | Log In (currently use case 1)  This includes the current 1.1 and 1.2 |
| **3** | **Add Staff Member (new use case)** |
| 4 | Generate Reports (currently use case 3) |
| 5 | Exit System (currently use case 4) |

##### Use Case: Add Staff Member (3)

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Name: Add Staff Member | | ID: 3 | Importance Level: High |
| Primary Actor: Employee | **Use Case Type:** Essential, Detail | | |
| Stakeholder and Interests:  Solicitor: Has hired a new employee and needs them to use the case management system.  Secretary: Has been asked to add new employee to the system in order to generate the username. | | | |
| Brief Description: The firm may hire a new employee and must obtain a staff username for the new employee. In order to trigger use case 2.1 (“Generate Username”), the system must receive the first and last name of the employee. | | | |
| Trigger: A new employee has been hired and they need log in credentials to access the system. | | | |
| Relationships:  Association: Employee  Includes:  Extends:  Generalisation: | | | |
| Normal Flow of Events:   1. A candidate has accepted offer of employment. 2. The Solicitor has asked the Secretary to add the new employee’s details to the system. 3. Secretary presses “Our Staff” button. 4. Secretary presses “Add Employee” button and enters required data. 5. Secretary presses “Confirm” button and system validates all input. 6. If input is valid, subflow 1 is performed. 7. Username is given to new employee who | | | |
| Subflows:  Subflow 1  System will generate the new employee’s username by concatenating the first name with a dot and the last name.  The new username is displayed to screen and given to new employee.  Step 2 requires same details as per NFR 8.1 (in Action Plan report):  first name, last name, date of birth, address first line, address second line, postcode, phone number, job title, and job hierarchy (responsible for and to). | | | |
| Alternate/Exceptional Flows:  If input is invalid, the relevant error message should be displayed to screen as described under Data Validation section. | | | |
|  | | | |

### Conceptual Classes

With a second linguistic analysis now performed, it is possible to review the conceptual model that was included in the Action Plan report (see Figure 8 in this document). In meeting number 2, the client reviewed this conceptual model and did not raise any concerns. The conceptual classes included in the model at Figure 8 are also common repetitive nouns that appear in our second linguistic analysis. This suggests that the conceptual model produced in the Action Plan report has correctly identified the most relevant entities the business in the context of case management.

The conceptual classes established in analysing the domain model are not intended to equate to a software class in the evolving system. In the domain model, a conceptual class can be viewed “in terms of its symbol, intension, and extension” (Larman, 2005). Its purpose, as described by Raul Sidnei Wazlawick, is to “discover how the information has to be structured and transformed” in order to represent the “comprehension of information by users, not its physical representation” (Sidnei, 2014). As such, we must remember we are still analysing the problem domain here. The evolving system is the solution to the problem domain; the conceptual model is how we interpret the end user sees the information. The objective at this point is to “study the problem. But the computational system is a solution, and therefore, it belongs to the design” (Sidnei, 2014). Larman also encourages the use of a category list when identifying conceptual classes, which includes, but is not limited to, "physical or tangible objects", "specifications, designs, or descriptions of things", and "roles of people". Also included in this list is the category "processes" which Larman states are "often *not* represented as a concept, but may be", and "records of finance, work, contracts, legal matters".

This raises two questions about the initial, top‑level conceptual model. Firstly, whether the conceptual model should be refined to include the Log In process; and secondly, does the conceptual class "Reports" absolutely have to remain in the model. If we consider what is conceptually ideal from the primary stakeholder's point of view (the legal firm), then the Log In process is not something that is specific or unique to this business context. Most software systems in any business will require their staff to log in. The conceptual classes, it should be remembered, are not intended to translate into software classes or components. So, our client is likely to see the Log In process not as conceptually ideal, but a necessary part of the system. Of general processes and transactions, Larman writes that "transactions with external services are useful to show in a domain model because activities and processes tend to revolve around them. They are important concepts" and follows on to state that "domain models are not necessarily correct or wrong" (Larman,2005). Considering the use cases and requirements we have so far, there is not a great number of *processes* identified. The client specified in meeting 1 (see meeting number 1 minutes in AP report, agenda item 5) that they only wish for the evolving system to track the figure of monies owed to the firm – so the system will not be designed to be able to process any kind of transactions. As we are really only dealing with one significant process in the evolving system (the Log In process), I am inclined to add this in to the conceptual model. It would be wiser to model it than to forget about it.

This now leaves us with the question of including a "Reports" conceptual class or not. The generation of reports has remained a primary use case since the refinement of the very initial, top‑level use case diagram in the Action Plan report. However, I question where the information within a report that is generated comes from. The generation of reports is essentially retrieving data from elsewhere in the system, performing a calculation on that data, and presenting the results in an easy to read manner. In terms of the client's business rules, the reports do not offer any significant or important advantage to the firm. It is identified in our linguistic analysis of Use Case: Generate Report (3) that the ability to generate reports is a nice extra for the system to have. There is also confirmation from the client that this use case can be omitted for now and can be implemented in a later iteration (see Meeting Number 3 Minutes, agenda item 7). Given that the client has explicitly stated they do not want this implemented at the moment and that the information which a report will display is extracted from elsewhere in the system (for example, from the total number of cases), this really means that there is duplicate information in the current conceptual model. For these reasons, I will remove "Reports" from the already‑identified conceptual model (Figure 8).

Additionally, meeting 3 with the client highlighted that to trigger the system to generate a new username for a new employee, the system should include an "Our Staff" button to be included in the navigation panel (see Meeting Number 3 Minutes, agenda item 5). This will lead the user to a window where new employee details can be entered and a username generated. As we have now established that storing staff details is an important requirement, I have included this in Figure 9 below, which now shows an updated conceptual model with basic relationship lines that reflect the discussion above.

enters

pertains to

**EMPLOYEE DETAILS**

**EMPLOYEE**

**CLIENT**

**CLIENT**

pertains to

pertains to

**CASE**

**CASE**

**CASE DETAILS**

**CASE DETAILS**

contains

contains

**CASELOAD**

**CASELOAD**

**USER**

**USER**

**EMPLOYEE**

**EMPLOYEE**

consists of

consists of

managed by

managed by

is a

is a

**LOG IN**

**EMPLOYEE**

must

pertains to

Figure Refined Conceptual Model

used in

pertains to

handled by

contains

provides

pertains to

**CLIENT DETAILS**

**CLIENT**

reviewed by

pertains to

required to update

pertains to

**SOLICITOR**

**CLIENT**

may

pertains to

supported by

pertains to

may

pertains to

**SECRETARY**

**CLIENT**

#### Relationships

I mentioned that the conceptual model at Figure 9 shows basic relationship lines that reflect the discussion prior. It is common for these models to display multiplicity notation. This defines “the number of entity type instances that can be at one end of an association” (mcleblanc, n.d.). However, I have chosen not to include multiplicity notation here because this is the conceptual model, or in Sidnei’s words it is a “comprehension of information by users, not its physical representation” (Sidnei, 2014).

If we had opted to develop a solution using a back-end relational database system, as was discussed in the Action Plan report, I would have created an Entity Relationship Diagram (ERD) which similarly identifies important entities and relationships among those entities. However, in the context of relational database management systems (RDMS), an entity is an object that is transformed into a table in the database. It is more important in that context to illustrate multiplicity (known as cardinality in database design) because that ERD will communicate a set of data rules that apply to that database's physical model (for example, you wouldn't want a system to allow two customers to reserve the same seat on a flight). The conceptual model shown at Figure 9, however, is not intended to transform into any software class or component. I wanted a conceptual model that is simple whilst capturing all of the business and information needs of the client, to ensure it is easy for the client to interpret.

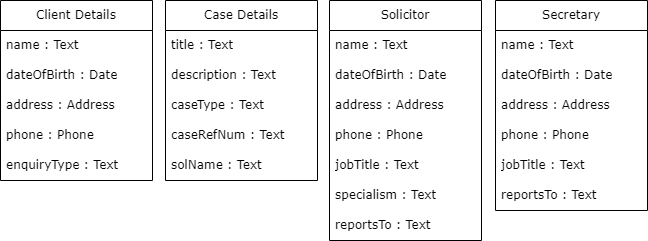
I may decide to set class names that take inspiration from our conceptual model, as this project will plan a solution for the client using a Java project with data structures. The class diagrams included later in this report will provide more detail of the multiplicity between software classes. I have, however, elaborated on the basic relationships that the conceptual model illustrates:

* The caseload has to be managed in the system by a user.
  + A user will be an employee, who must log in to access the system.
  + To log in to the system, the employee must have log in credentials, part of which (the username) are generated using employee details.
  + To generate a username for use in the log in process, an employee with access to the system must enter the details of a new employee (the only users without a username).
* A caseload is simply a collection of all individual cases the firm is responsible for.
  + A single case is created on behalf of one client, and will contain a type, title, and description along with a unique identifier (see FR 2.1 and FR 2.4 in Action Plan report)
  + The client has no relationship to any part of the evolving system, except for fulfilling the business process of providing their details to the Solicitor. The Solicitor will always review the client details provided by the client before accepting a new case.
  + A case will always be handled by a Solicitor, whose details will be stored in the system and associated with every case belonging to that Solicitor.[[2]](#footnote-2)
  + The Solicitor may pass all details obtained (the case details and client details) on to a Secretary who supports the Solicitor with day to day tasks if a new case is required to be created. The Secretary will then log in (if they are not already logged in) to perform any necessary tasks.
  + Alternatively, the Solicitor may log in to the system themselves if needed to perform a task.

#### Attributes

It is also common for a domain model to show attributes of the conceptual classes. The use case descriptions supply us with enough analysis to enable us to show where information needs to be remembered. I have only included simple attributes, keeping in mind the conceptual model is first and foremost a communication tool – this is also why I have broken up the diagram into individual conceptual classes so as to avoid overcrowding. I have not included attributes for every single conceptual class as it is common practice, and indeed preferable according to Larman, to only include simple attributes or data types. You will see some data types are represented below as a non‑primitive class (for example, Date and Address). Larman advises that if an attribute is “composed of separate sections” or is associated with validation processes (Larman, 2005), then consideration should be given to representing that attribute as a non-primitive class. It would be possible then to show these attributes in their own boxes with association lines, but I would like to keep the diagram intuitive so the client can easily read and interpret it and have therefore illustrated these non‑primitive classes **in the attribute compartment** of their class box (Figure 10) without association lines.

Figure Conceptual Classes with Attributes



### Final Conceptual Model

The final conceptual model that ties the conceptual classes with attributes and associations together is shown below at Figure 11 below. For clarity, the conceptual classes Employee and Employee Details do not show attributes because this information is stored in Secretary and Solicitor, which are both a type of Employee. I have included Employee and Employee Details as conceptual classes merely to show that a user can only be an employee of the firm and that we need to know information about that employee in order to log in (a username is generated using new employee details):

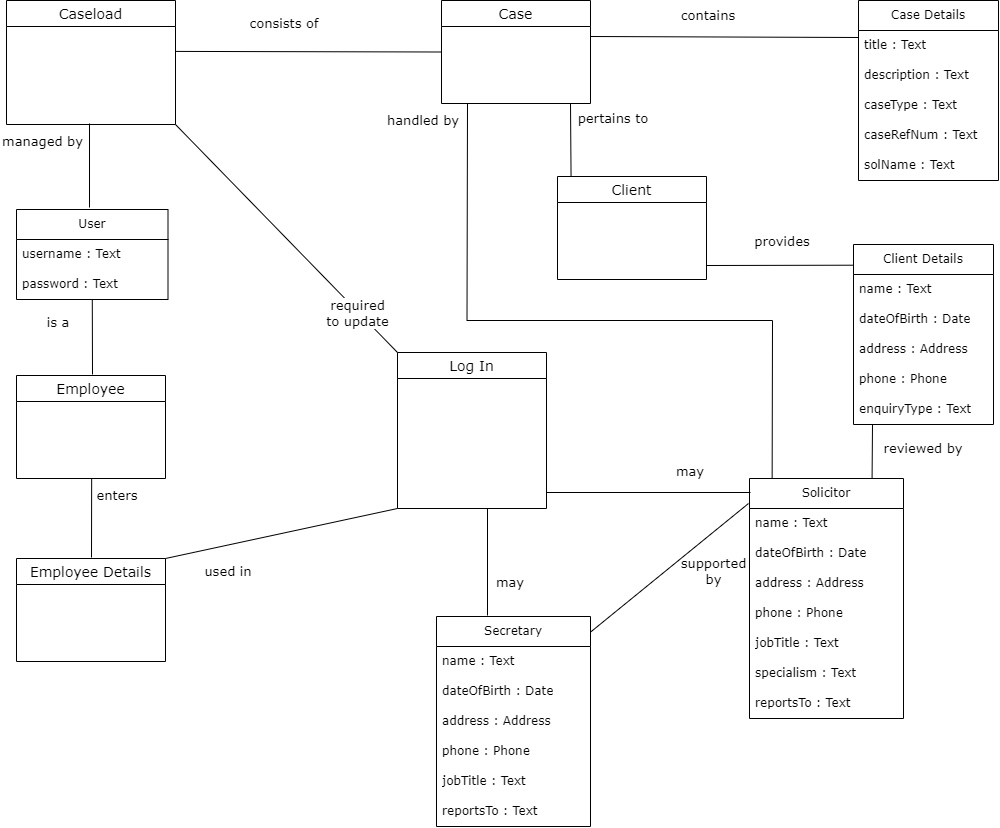
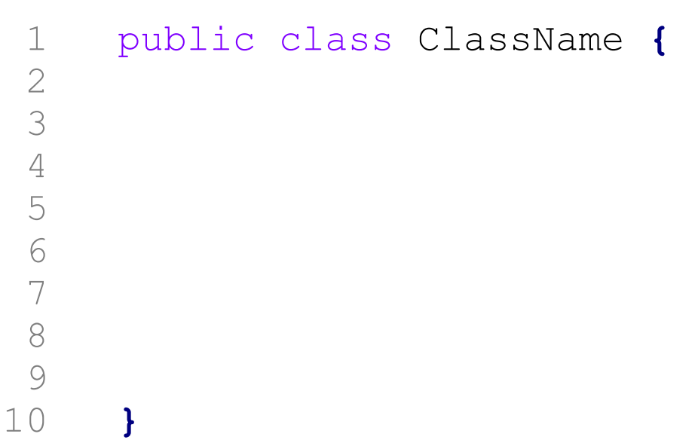


Figure The Final Conceptual Model

### Responsibilities and Collaborations

The conceptual model alone is not enough to allow for programming to start. However, there is enough groundwork now to enable us to look at the problem in an object‑oriented manner. This is why some requirements and analysis workflow activities (specifically the second Natural Language Analysis and the description for the new Use Case: Add Staff Member (3)) have been included under this section The Domain Model. The UP supports various activities to happen concurrently, but it also supports the natural progression of requirements into analysis into design into implementation. We are simply not constrained to finishing any one workflow in its entirety before moving into the next.

Any software built on an object‑oriented programming model will be designed around data and objects; unlike a model based on a logic programming paradigm where facts and rules are written as a series clauses. A program based on logic programming would then search an and‑or‑tree to determine the possibilities of which objects satisfy the overall objective. But in this project, software classes will be at the core of the evolving system. An object‑oriented program will consist of many objects, and to structure the blueprint of an object, a class has to be created. At a basic level, any class in Java will conform to this format:



**Properties**

**Behaviours**

Figure A Java class is the blueprint of an object

In UML, a software class is shown through class diagrams. Through this, we can illustrate the types of objects the system can construct, the information which that object can store, and the operations which that object can perform.

Many of the key benefits of object‑oriented programming can be communicated through UML class diagrams. Some of these principles are abstraction, encapsulation, and modularity. It was clearly identified in the AP report (see Development Route Option 2) that Java is an adaptable programming language as it is platform independent and it lends itself well to object‑oriented programming, meaning it offers us the benefit of being able to reuse code which saves the project team time, and less time spent on the project ultimately saves the client money, or at least contributes to keeping the project costs within budget. The design of the software classes is where the benefits of this chosen development route will begin to come to fruition. If this design is thought through carefully, the program will be as robust as we can possibly make it, meaning there is less risk of error and delay later in the project. To really emphasise the importance of designing a software product, I have quoted the following from *Data Structures and Algorithms in Java* which describes the consequences of poorly designed software:

*…software that is not robust could be deadly. This point was driven home in the late 1980s in accidents involving Therac-25, a radiation-therapy machine, which severely overdosed six patients between 1985 and 1987, some of whom died from complications resulting from their radiation overdose. All six accidents were traced to software errors*. (Goodrich, Tamassia and Goldwasser, 2014)

A previous version of the Therac machine was not object‑oriented, but the decision was made to reuse code from this older version which made the Therac-25 unsuitable for the hardware that it was used with. This case management system project isn't relevant at all to the kind of application described above, but it really highlights how important it is to think through how the pieces of your software will fit together to result in a successful system.

## Designing an Object‑Oriented System

So how do we begin to design an object‑oriented system? So far a lot of time has been spent in analysing the problem domain. Now we have to set out a solution that will fulfil the client's objective of making the management of data easy and quick, as we identified in the System Proposal in the AP report submitted previously.

### CRC Cards

An informal but effective technique to start thinking of how to implement the solution as an object‑oriented software system is to draw out Class, Responsibility and Collaboration (CRC) cards. This process encourages the development team to brainstorm what software classes should be implemented and to discuss how classes should or should not interact with each other. It is a good method to enable decision‑making with regard to what we should eliminate or add as a software class to our design. This process will consider all of the groundwork that has been carried out so far, particularly the findings of the linguistic analyses (both in this document and in the AP report), all use cases and use case descriptions, and the conceptual model that has just been discussed.

This is generally an activity that is performed in a small group. However, this is an individual project, so I cannot really provide any notable points that were discussed with anyone. To begin the process of developing CRC cards, I have taken what I feel are entities that are of absolute importance to the business and to the system. This is supposed to be a relatively rapid process, so I have initially went with what seemed instinctive and natural. We arrived at the final conceptual model through carefully analysing all of the use cases and use case descriptions, and there is no doubt that legal cases are what the firm's work centres around. It is also very clear that the evolving system is going to have to store information about a case, so I started with "Case" for the first CRC card (see Figure 13), bearing in mind that responsibilities may be reorganised and classes may be removed or added later in the process – this is informal; the software will not be built based on CRC cards.

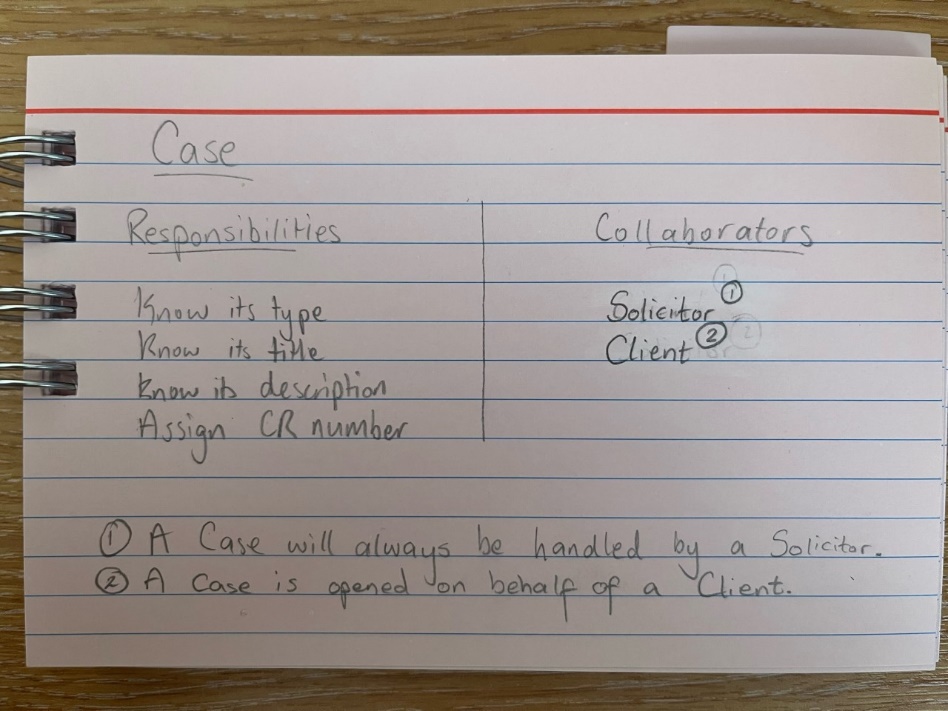


Figure Case CRC Card

The conceptual model identified that a Case will have a relationship with a Client and Solicitor. So I then created CRC cards for these classes.

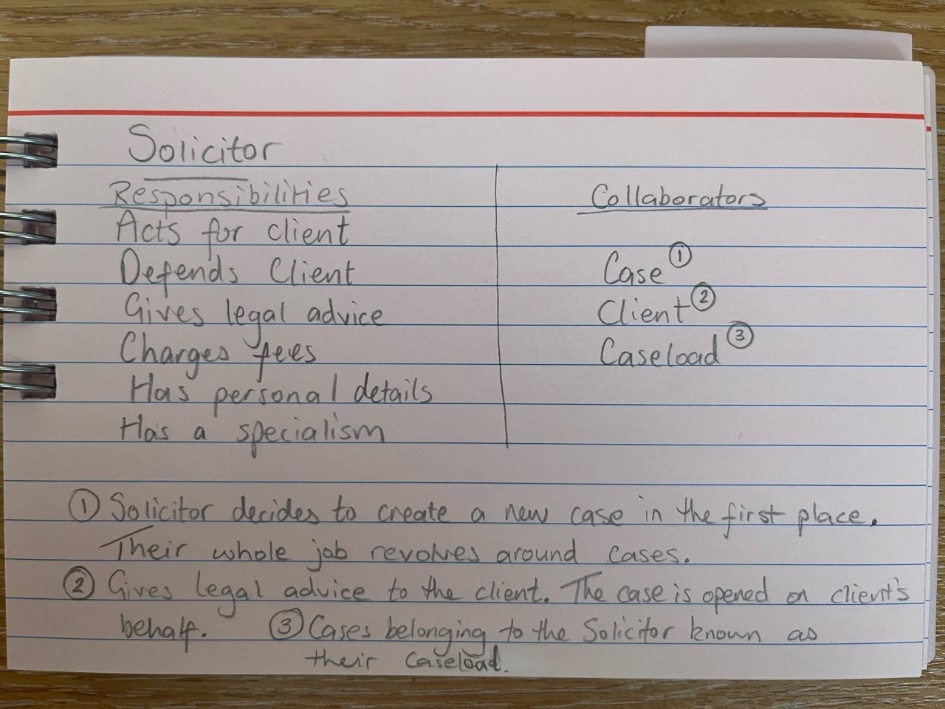


Figure Solicitor - CRC Card

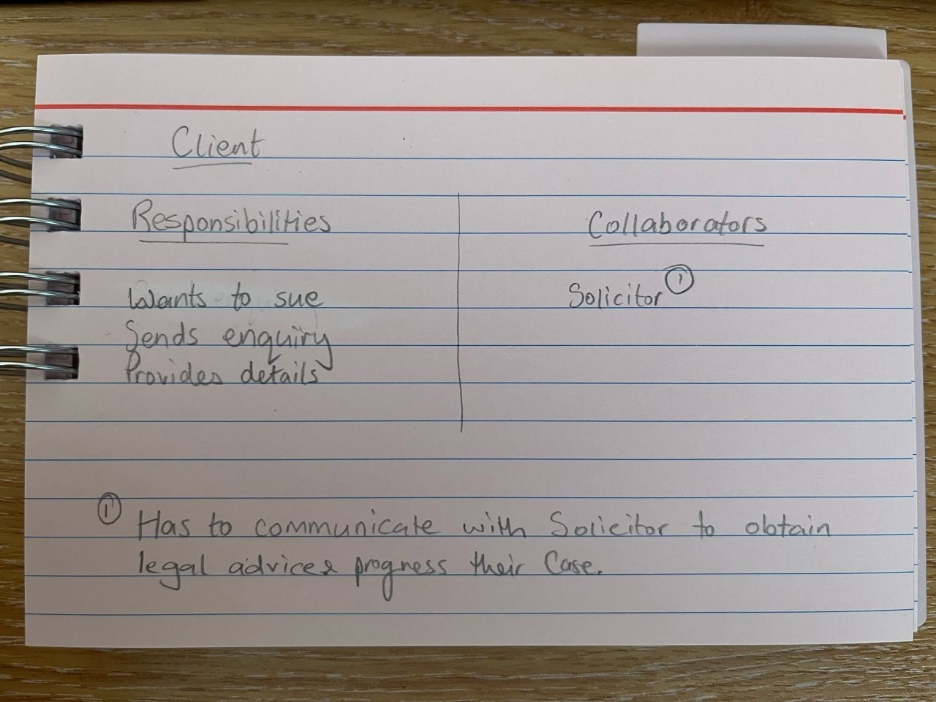


Figure Client - CRC Card

The question now is whether to include the above classes as software classes. It has been clear since the inception of the project that the firm wants their system to be able to record a variety of information relating to their cases. We developed a comprehensive list of functional requirements in the Action Plan report that explicitly states the system should accept valid case, client, and Solicitor details. Without all of this information, the overall managing of cases will be extremely difficult and cumbersome because the user will not be able to establish what case belongs to what Solicitor, and what client the case relates to. I therefore think it is appropriate to retain Case, Solicitor, and Client as classes that will become software classes: we are probably going to need to make an object out of these entities to ensure accurate data is stored.

But how will the system actually work with only three software classes? The conceptual model was primarily a communication tool, which combined with the use case diagrams communicated to the client everything the evolving system will allow them to do. The conceptual model does not identify any specific entity that tells me the user will be able to track the monies owed to the firm, but the use case diagram and descriptions do. If we review the minutes of meeting number 2, agenda item 7 in the AP report, the client has specifically stated that they only want the system to track the activities and the time spent. Then, if we review the minutes of meeting number 3, we see the only issue highlighted from the fully dressed use case diagram was the “Generate Report” use case. I take from this that recording billable activities and calculating a running total of the monies owed for each individual case is all the system should be capable of at this stage. I have therefore created a CRC card for a potential entity “Activity” (Figure 16).

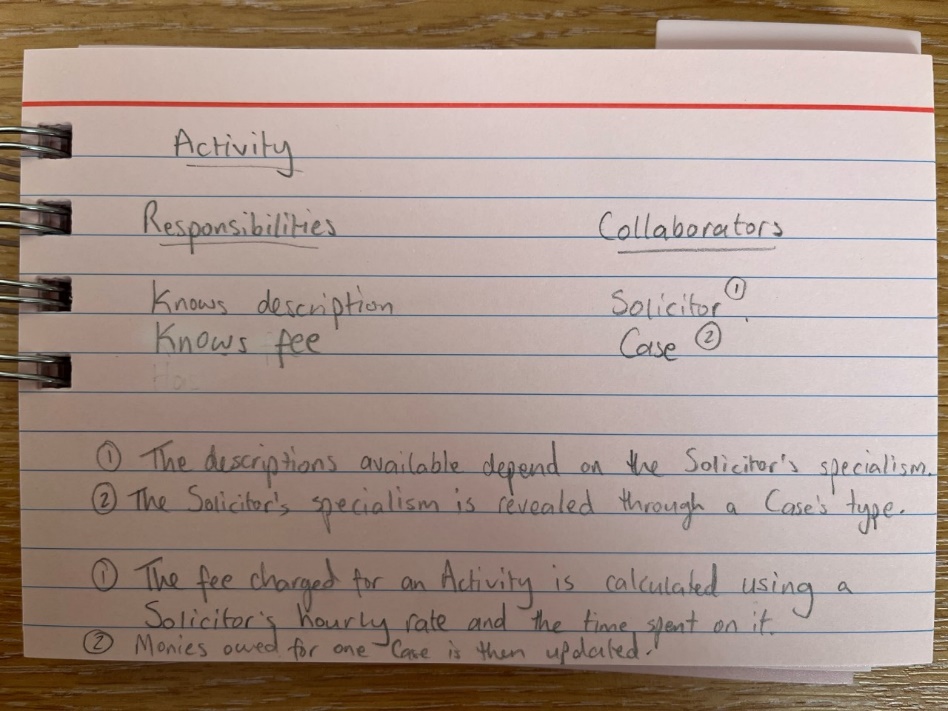


Figure Activity - CRC Card

I have described the relevant relationships when developing the conceptual model, and have also included descriptions of the relationships between the classes on the CRC cards above. With this grounding, I want to devise an initial class diagram that will represent a basic overview of software classes. This will likely be a better way of drawing attention to any discrepancies or issues to be addressed in terms of the object‑oriented software.

### Data Binding Model

If we are going to build an object‑oriented system, we need to decide how and where we will store any data that the system has to be able to handle. The Action Plan report submitted previously stated that this project would plan a solution using a Java project with data structures, as opposed to a database. We do not have to make final decisions about what data structures we are going to use for every single piece of data just now, but I have provided a short summary below of what could be suitable data structures for key bits of data that we know we will have to store.

#### Data Structures

|  |  |
| --- | --- |
| **What we need to store** | **What we can store it in** |
| Username and password for each Employee | We need to be able to match up the employee to **their** username and password, not someone else’s username and password. If we assign a staffId number to all employees, we could potentially implement a **Map** and take advantage of the simple key and value pair – provide staffId as a key, and their input as a value. Repeat for password field. |
| Cases | We know the whole point of the system is for the client to manage their caseload. The system is therefore going to have to store all of the cases somewhere. We don’t know how many cases the system is going to hold at any one time because the firm may remove cases and add cases all the time. So we need something flexible like an **ArrayList** to hold all of the cases. This will be able to grow or shrink dynamically and will be able to accommodate an unknown number of cases. |
| Case Details | We need to store a bunch of details for each case. There will probably be a need to have a Case class so we can construct Case objects. We know the client wants to be able to search for cases, and they’ll do this by inputting a case reference number. We could potentially implement a **Map** here using the user’s input as the key, and then retrieve the Case with that reference number. |
| Billable Activities | We know the client wants to record billable activities – this is basically how they make their money. A billable activity is essentially the product they sell. We’re going to have to store the activity’s description somewhere. We already know every possible value of an activity’s description (see NFR 15 and 16 in AP report), so there is potential here to create an **enum class**. |
| Solicitor Details | The system is going to have to store details about their Solicitors in some way. There will probably be a need to define an Employee class, as meeting number 3 revealed a new use case – “Add Staff Member”, so an Employee class would allow for storing Solicitor details and Secretary details. An Employee class would create the need to store their job title, so there is probably the potential to create an **enum class** here too. |
| Client Details | Similar to Solicitor details, every Case is going to have a Client linked to it. Defining a Client class would allow us to store the required details. |

## The Static Model

The CRC cards technique acts as a sort of bridge between viewing the problem domain through use cases and conceptual classes and moving into the designing of an object‑oriented software system. I mentioned in The Use Case Model that many believe the Elaboration phase of the UP equates to the general Design phase of a standard SDLC. But really there was still a lot of analysis to be done. Now, there is enough work done to design the structure and architecture of the evolving system.

### Initial Class Diagram (1)

The CRC cards helped us to identify that it is appropriate to retain Case, Client, and Solicitor classes in the evolving system. I then considered how an object‑oriented system would account for tracking monies owed to the firm. Using the CRC cards as a basis, I have developed the following top‑level class diagram with some basic attributes and operations.

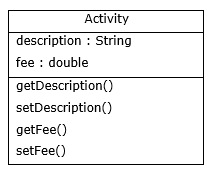
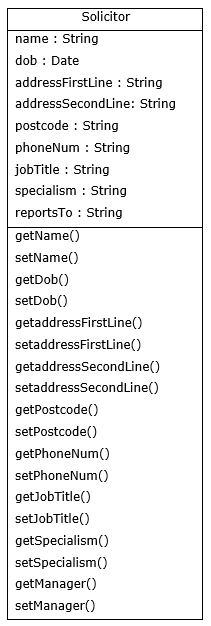
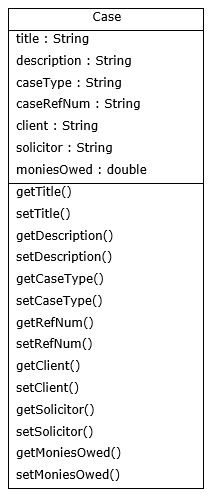
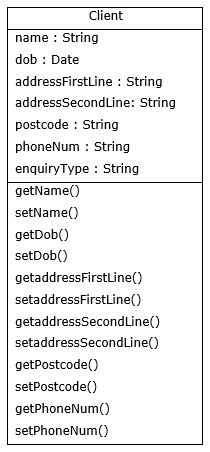


Figure Initial Class Diagram (1)



There’s a few issues with the above class diagrams, though. The most obvious issues are (in no particular order): the class relationships are not illustrated; the Client and Solicitor classes are a little busy indicating they hold too much responsibility; and each class contains a lot of String data, which will require a lot of validation. Miles and Hamilton’s Learning UML 2.0 offers two very concise and succinct chapters on software classes. They state that classes “do not live in a vacuum”, and explain that some relationships are stronger than others:

*Two classes that are strongly dependent on one another are said to be tightly coupled; changes to one class will most likely affect the other class. Tight coupling is usually, but not always, a bad thing; therefore, the stronger the relationship, the more careful you need to be*. (Miles and Hamilton, 2006)

If we consider closely the potential relationships of the classes at Figure 17, I would argue that there are some classes missing.

#### Dependancy

Some software classes will always depend on another software class. In other words, at runtime, some classes have to work hand in hand with each other to achieve a goal. So it is all well and good having a Case class that stores all the required properties and offers all the necessary behaviours to construct one legal case, but how will this information be displayed to the user? And how will the user be able to create many different cases, or search for a specific case? This strongly suggests to me we need to create a class for the user interface, and we need to decide how and where the collection of legal cases are going to be stored to enable the interaction between the user interface and the data stored in the system. The former seems obvious – a UserInterface class that deals with the responsibilities and operations of user interaction; responsible for constructing the graphical user interface and registering all necessary event handlers. The user is going to add to and update the collection of legal cases, so the inclusion of a Caseload class of which the UserInterface is dependant on is one potential solution. It was identified earlier when discussing potential data structures (in section Data Binding Model) that an ArrayList would be able to cope with storing an unknown number of Case objects, suggesting a property of the Caseload class is a List property containing any number of Case objects. This is a suitable data structure for this context, but it’s important to bear in mind that the final decision as to what data structure will be used will ultimately reside with the developer when they come to **implement** the solution – we are simply **planning** the solution here and have only slightly dipped in to the implementation workflow.

#### Package Structure

There are also still decisions to be made regarding how the user interface is going to relate to the Caseload class. It was mentioned above that a potential solution is to create a UserInterface class that is dependant on the Caseload class (which contains data in the form of many Case objects). It would aid security in the evolving system to separate these into packages. This would help in maintaining an organised structure in the program as it would effectively separate the graphical user interface from the business domain. Miles and Hamilton suggest that tight coupling is often a bad thing, so splitting the user interface into its own package would create distance between these classes. Well-designed software aims to ensure that a class does not bear too much responsibility, so by including a **separate package** to deal with the business domain, with a **Caseload class** that simply worries only about managing the caseload, the software will be far more maintainable – any changes that are required are less likely to have a severe impact on the program’s functionality overall by adopting this package structure. Put simply, if we make changes to one class in a loosely coupled system, it will generally not have as much of an impact on other classes:

*Good OO design is when each class and module in your software does one basic thing, and that one thing really well. As soon as one class starts doing two or three different things, you’re probably moving away from cohesion, and good OO design. … High cohesion and loose coupling adds up to software that can easily be extended, or even broken up and reused, because all the objects in the software aren’t interdependent. Think about it this way: the higher the cohesion in your application, the better defined each object’s job is. And the better defined an object (and its job) is, the easier it is to pull that object out of one context, and have the object do the same job in another context. The object is happy to just keep on doing its very specific job, no matter where it’s being used.* (Mclaughlin, Pollice and West, 2007)

If a loosely coupled design means the end result will be easily extendable and reusable, I want to attempt to plan a solution that constructs objects that are not interdependent on each other as far as possible.

#### Enumeration

One of the issues with the initial class diagram was the amount of String data being held. This isn’t necessarily a bad thing, but it means if we compare these values at any point – and we *will* have to compare user input to what we consider as acceptable input frequently for validation purposes – then our code runs the risk of becoming very unreadable very quickly. It’s always good practice to keep code readable, in case other developers come to work on the software in the future. There are properties of certain objects where we already know every possible value, specifically a **case type**, a **Solicitor’s job title**, and an **Activity’s description**. These properties are currently all String values, so to avoid any problems with misspellings or letter case, we could create an enumerated type. Where the job title property is concerned, an enumerated type will increase the program’s flexibility because the system is going to be used by employees of the firm, so if they hire any new employee the system already knows what an acceptable value may be for any employee. This means we will not have to worry about any issues regarding this data as we can ensure that the system will only accept valid values for these properties. The user will not be able to make a mistake where these details are concerned because the system will simply not allow anything other than values defined in the enum classes. An enum class makes the code far more readable, and if the firm were to, for example, employee a new Solicitor who deals with a different type of legal case, the only change we would have to make is to add in another property to the enum class. The values for a case type are also the values of a Solicitor’s specialism, so we can reuse this data when setting a Solicitor’s specialism. Overall, enumeration classes help to ensure data is safe and accurate, and stored in the right place.

An enum class for a case type *may* look something like this:

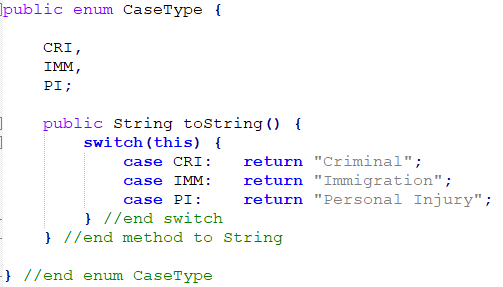


Figure Case Type - Enumeration

Similarly, an enum class for a job title and activity *may* look like this:

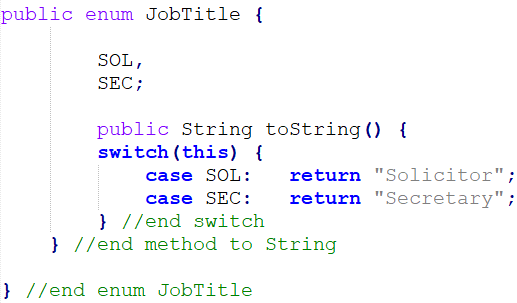


Figure Job Title - Enumeration

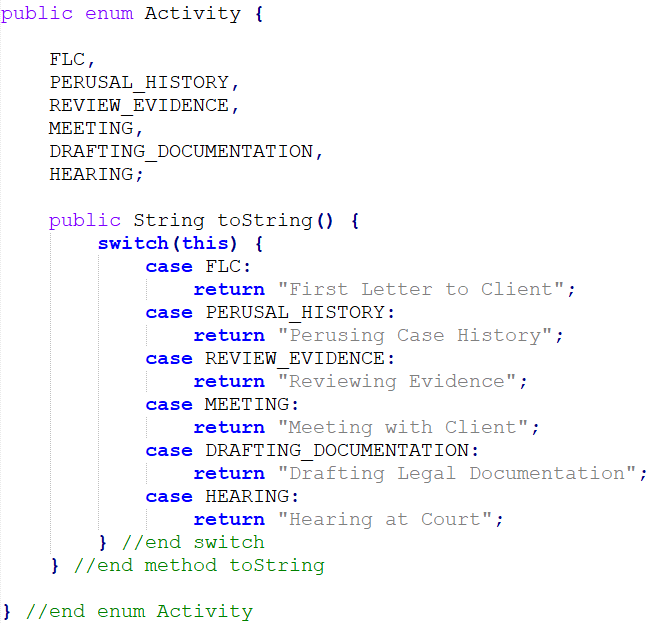


Figure Activity - Enumeration

#### Person – Generalisation and Inheritance

There is also a lot of common fields if we compare the Client and Solicitor class. These fields all relate to common personal details that are often required for a multitude of reasons (for example if they hire a new employee, they obtain the employee’s personal details; if they take on a new client, they obtain the client’s personal details, and so on). We need to decide what kind of relationship this is in terms of software classes. There are going to be a number of contexts where the system will use a person’s personal details. If we look at the entities we have mentioned both here, in our use case descriptions and also in the conceptual model, the nouns “client”, “Solicitor”, “employee”, and “Secretary” come up again and again. I think this is a suitable context to implement generalisation, also known as inheritance. There’s not much variation in terms of their user roles in the new system – a client will not be using the system at all, and employee/Solicitor/Secretary will all have the same access to the same functionality. I am therefore inclined to implement a superclass Person (they are all types of persons), with subclasses of Client and Employee. I would like to implement two subclasses of Person – currently there are only two types of employee (Secretary and Solicitor), and the only variance between a Secretary and Solicitor is the addition of a specialism property for a Solicitor, so it makes sense to have an Employee class that extends the superclass Person. I then want to show that an Employee can be a Solicitor or a Secretary; I want to the ability to distinguish between a Solicitor object and a Secretary object because the system is to track the monies owed based on activities performed by a Solicitor; not a Secretary. We therefore need to store their hourly rate somewhere as this hourly rate value has to be used in computing the activity fee (there is no other way to calculate this figure).

In considering the software classes to be implemented, I realise an important property has been overlooked: adding in an id field to the Employee and Client class will allow us to identify or refer to the correct object. If we identified every Client object through their name or address, we cannot safely rely on the system retrieving the client we are actually looking for. Even though the client has stated that they only want to be able to search cases at this point, it seems sensible to assign an identification number to employees and staff. This would allow the system to be extended easily in the future if the client wanted to be able to search for a client – **clarification should be sought from the client regarding format of identification numbers for Employees and Clients.** A top level class diagram showing this inheritance is shown below:

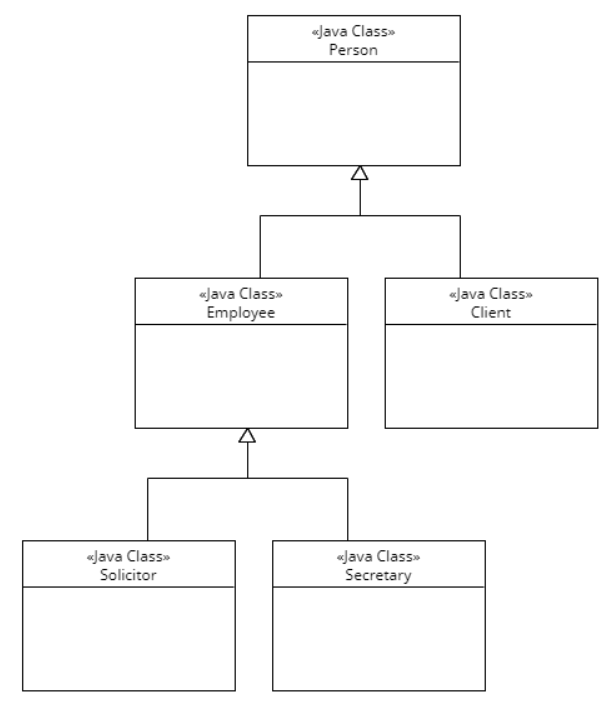


Figure Top Level Class Diagram - Generalisation (Person)

Before I provide an updated class diagram, the final problem to address with the class diagram at Figure 17 is the “fee” property of the Activity class. We are going to have to calculate a minimum of two things for each individual Case somewhere in the system: (1) the fee charged for the time spent on a single activity; and (2) the total monies owed for all the activities a Solicitor has performed. As it stands, there is a moniesOwed property in the Case class diagram, and there is a fee property in the Activity class diagram. To arrive at the fee figure for a single billable activity, we need to accept a value for the time spent on that activity. The system cannot determine this – it is the Solicitor that performs the billable activity, and they may spend anywhere from 15 minutes up to 10 hours on an activity (apparently). The only control we can have here is over the time spent measurement – we know from Meeting Number 3 Minutes, agenda item 6 that the firm charges by units. A unit is equal to 60 minutes. Every Solicitor working for the firm charges this way. I do not think it is worth creating an enum class to define these values as they are numerical. It would be simpler to define a fixed size array in the BillableActivity class that holds valid time spent units. What is missing from the initial class diagram is a property that holds a value for a Solicitor object’s hourly rate (which varies between Solicitors), and a property that holds a value for the time spent on an billable activity.

### Initial Class Diagram (2)

Taking all of the above into consideration, we end up with this:

No relationship line!  
Needs revision.

Packages

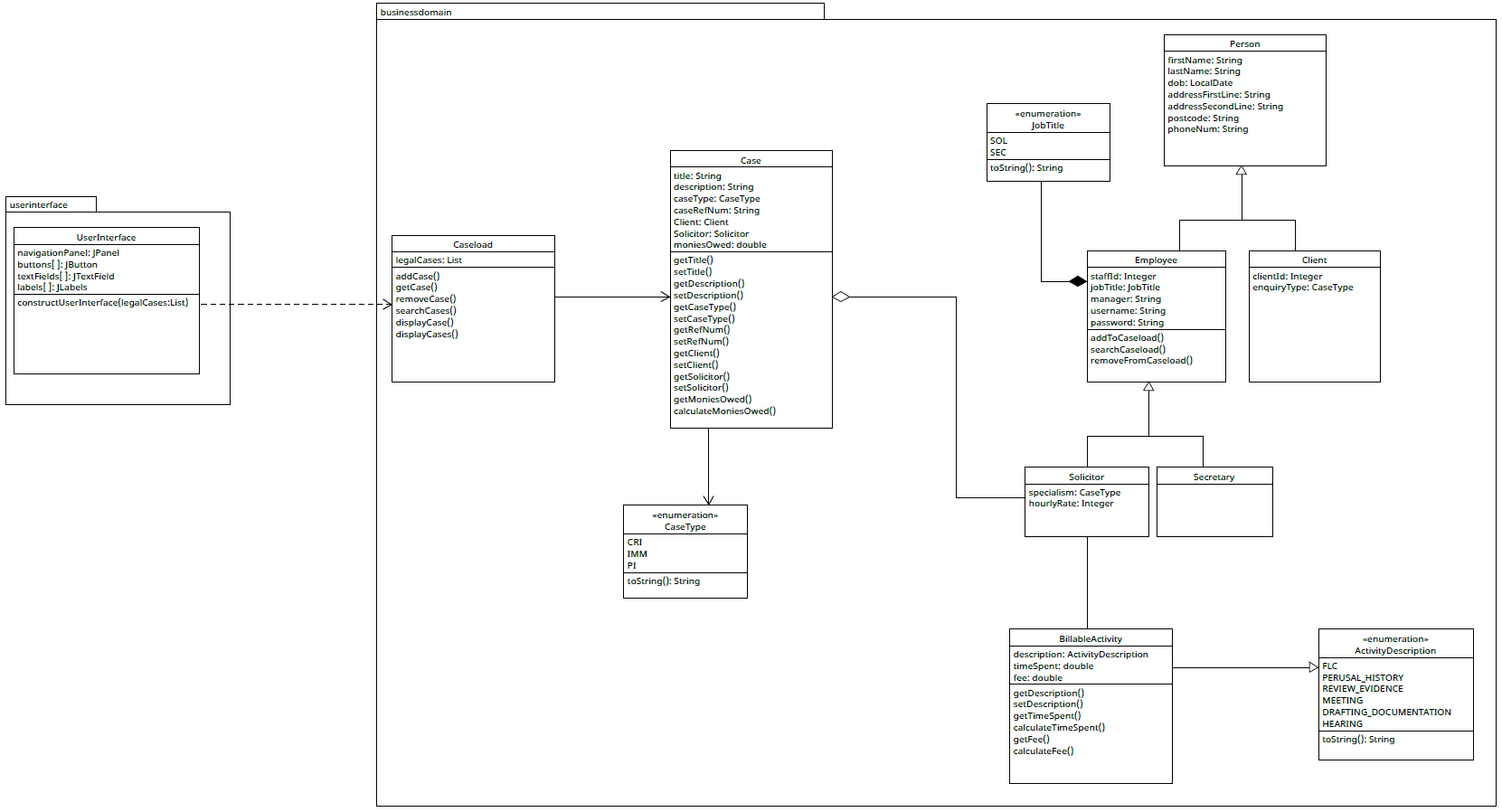


Figure Initial Class Diagram (2)



Composition**\***

Dependency

Association

**Include navigability arrow here?**

Aggregation

**\*Developer note**: debatable whether this should be a composition relationship. May be changed.

Association

Inheritance

Association

The above class diagram is a better reflection of the software classes that may be implemented in the evolving system. To summarise this class diagram at Figure 22, we can see that:

* the user interface is **dependent** on the Caseload (no functionality would be available to the user if their input isn't stored somewhere);
* the Caseload is **associated** with Case – Caseload will contain references to case objects in the form of an attribute. This has been shown as a List data structure, allowing us to take advantage of pre‑defined methods such as add() and remove().[[3]](#footnote-3) The navigability arrow tells us that we can ask a Caseload what Cases it contains, rather than asking a Case what Caseload it belongs to;
* a Solicitor owns a Case object (**aggregation** relationship) – this is to reflect that every Case will be handled by a Solicitor, as we established in our analysis of the business. It might be argued here that as it makes sense to say a Solicitor "has-a" case, then it might be worth considering a composition relationship here. However, if a Solicitor left the firm to work elsewhere, this does not mean that a Case is over and complete. In other words, if a Solicitor object is destroyed, we do not want to destroy the Case object along with it. They should be able to exist independent of each other;
* in order to construct an Employee object, there must be a JobTitle. This is shown as a **composition** relationship, which works similarly to an aggregation relationship in that it simply results in the addition of a property at the time of implementation. I debated whether to reflect this as a composition relationship. I really wanted to communicate that you can’t construct an Employee object without a job title. However, it would be plausible to show this relationship as an association with a navigability arrow that indicates only the Employee class should contain an association‑introduced property to hold the job title data (and not vice versa). I think spending too long deliberating over what type of relationship this should be has overcomplicated the diagram and I am inclined to make this amendment. However, an enum is really just a data type I have defined, and it sounds sensible to say an Employee "has a" a JobTitle (composition relationships generally represent an "has a" relationship in OOP). As an Employee object shouldn't really exist without a job title, I will retain the composition relationship for this reason.

There are a few more details I would like to show in the class diagram.

### Case – Inheritance or Interface?

In Figure 22 I have not included subclasses of Case to reflect the type of cases that can be constructed. But we know that an Immigration case should show an additional billable activity that is unique to that type of Case. Rather than implementing inheritance for different types of cases (which would make the application *less* loosely coupled (which is bad)), it would be safer to define that if a Solicitor’s specialism attribute is not equal to Immigration, then the ActivityDescription field should not accept "Home Office Decision Appeal" as valid input.

It may be possible to define a CriminalCase, ImmigrationCase, and PICase class by including a Case **interface**, which isn't as strong a relationship as inheritance. A class that implements an interface is forced to implement all abstract methods. Using an interface would force the separation of what a class can do, to how it actually does it, which could allow the advantage of polymorphism as the methods described in the interface will be implemented by all classes that implement the interface (implementing an interface is essentially saying “if you implement me, you must do x, y, z, but you can do it in your own way”). In other words, using an interface class is a binding contract between the analyst and developer to force certain behaviours. An advantage of including interfaces in our system is that there is no limit to the number of interfaces a class can implement, but a class can only ever extend one single class. Even if a Case interface brought no advantages to us right now, it would make it much easier to make changes to the program in the future. Say, for example, the firm started charging clients in different ways according to their type of case, it would be far easier to safely add specialised methods to classes that implement the Case interface as it would be to change a superclass without impacting the overall program. For now, I will **retain the Case class as a Java class**, but it is possible the class structure here may change during development to include a Case interface.

This is a part of software design that really highlights how valuable working with a good team is. This is an individual project, so this decision‑making falls on to myself only. This would be a really good discussion to have with another member of a development team, and would probably be far less time consuming (my indecisiveness is really showing here!). However since I am developing this system by adopting an agile methodology, the development process is much more fluid, so the class structure in the actual solution may well differ from the class diagrams provided here. Of course, updated and accurate diagrams will be provided during the development stage if this does happen.

### An Abstract Person

I mentioned earlier it is not always the best choice to implement software that is tightly coupled. Inheritance does just this – the subclasses are able to access the internal components of the superclass, so these subclasses are in fact tightly coupled to the superclass. The reason I have shown inheritance for Person, Employee, and Client above is that the subclasses are actually a more specialised type of a Person, and there are clear properties and behaviours that belong to an Employee but not to a Client – it is not just for convenience in this context. In real world terms, an Employee *is a* Person; a Client *is a* Person, and so on. These types of relationships complement the use of abstract classes, where subclasses can access the implemented methods at the super level, but have to implement the abstract methods at the sub level.

The alternative would be to create a Person interface (as I discussed above in the context of Case objects) which is a Java construct that defines the roles an object must assume. An interface class contains method signatures that have no implementation code, it is often referred to “*as a very simple contract that declares, “These are the operations that must be implemented by classes that intend to meet this contract”*” (Miles and Hamilton, 2006). The drawback of using interfaces is that where base functionality is required to be defined, the interface can’t contain that implementation.

There is no golden rule in OOP to guide our decision of whether to use an interface or an abstract class. As we need behaviours to access and retrieve details relating to a person, I think it is a better option to **use an abstract Person class**. Looking more closely at the Person class, it would not make sense to construct an object out of Person because this has to be **either** an Employee or a Client. To complete a Person object, certain pieces have to be added to it. In other words, we don't want the ability to instantiate a Person object. To achieve this, the Person class should be defined as abstract.

There is another amendment which, on reflection, I think should be made to Initial Class Diagram (2) to make the evolving system a little more flexible. To design the system so that it is less tightly coupled, it is possible for the program to work **without** a Solicitor or Secretary subclass. The Secretary subclass contains no specialised properties or behaviours, and the Solicitor subclass simply contains two properties. These two properties can be moved to the Employee class, where an additional constructor method can be included to reflect that a Secretary object does not need a specialism or hourlyRate property.

### Cases, Clients, and Composition

It was pointed out at Initial Class Diagram (2) that a Case class contained a property of type Client, but there was no relationship line to communicate this. This was an oversight on my part – I clearly had in mind that the evolving system will have to be able to link together a Case object and a Client object, but had not decided on the relationship type.

The relationship between a Client object and a Case brings into question the legal firm's overall objective of the evolving system. That is, in a nutshell, to make the management of their caseload easier. I take from that that the Case is what concerns the firm the most. What good would it be if we could retrieve Client object details, but not the Case that relates to that Client? This suggests that a Case has a Client; a single Case object should contain a single Client object (Figure 23). This is a system to track clients and their corresponding legal case. So, if a Case object is destroyed, the corresponding Client object should be destroyed. As things stand currently, if a Client doesn't have a Case, they shouldn't be in the system. Perhaps in the future the firm would like to be able to track historical cases, but if we review the discussion with our client at Meeting Number 3 Minutes, agenda item 7, our client specified that tracking historical cases is not a requirement at this time.

Put simply, a Client object should not exist if the Case object does not exist. The Client doesn't really own the legal case; it is the Solicitor that oversees and therefore owns the legal case on the client's behalf. With this in mind, a **composition relationship** will show that a single Case object should contain a single Client object.

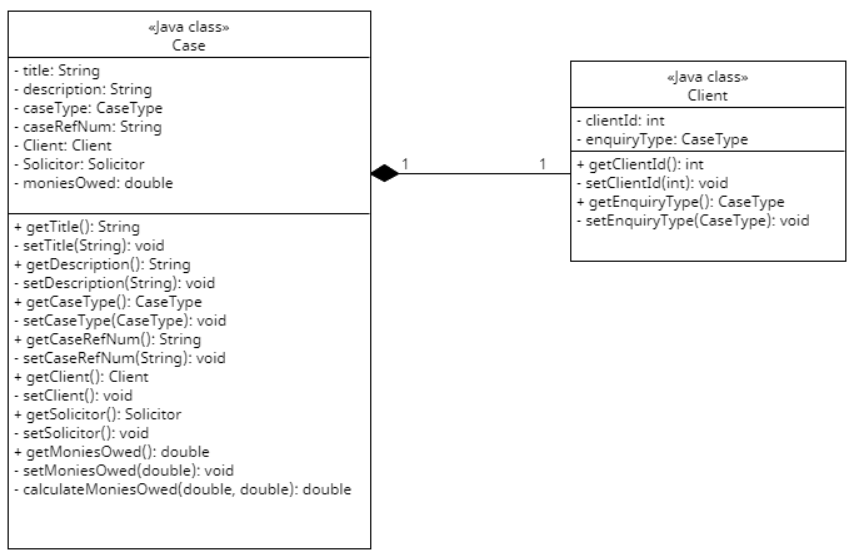


Figure Composition - Case and Client

### Billable Activities and Associations

The relationship between the BillableActivity class and Solicitor class, and the BillableActivity class and Activity Description enum class raises a couple of questions. The BillableActivity class holds responsibility for storing the time spent on an activity and calculating a single activity's fee using that timeSpent figure. In order to calculate the fee, the BillableActivity class also needs to know the Solicitor's hourly rate, which is an attribute of the Solicitor class. It was shown as an association relationship in an attempt to communicate that: a Solicitor class works with a BillableActivity object in order to arrive at a total monies owed figure for a Case, and that the relationship is stronger than a simple dependency. So we're left with the question of trying to design this relationship that is stronger than dependency but not strong enough to include a property that is a reference to another object. If a user adds a new case to the system, there might not be the need to record billable activities straight away. Therefore, it would not make sense to have a reference to a BillableActivity object in the Solicitor class. Perhaps, though, it would make more sense for a BillableActivity object to contain a reference to a Employee object. This would allow the BillableActivity class to access the hourly rate of the Solicitor. A simple navigability arrow pointing to the Employee class will achieve this. Similarly, when an activity is to be recorded in the system, it will have know what Case the activity related to. This will ensure the totalMoniesOwed figure will be accurate.

In addition, the relationship between BillableActivity and ActivityDescription should be revised to show a composition relationship rather than an association. This is a better reflection of the relationship type between these two classes – a BillableActivity **has a** description.

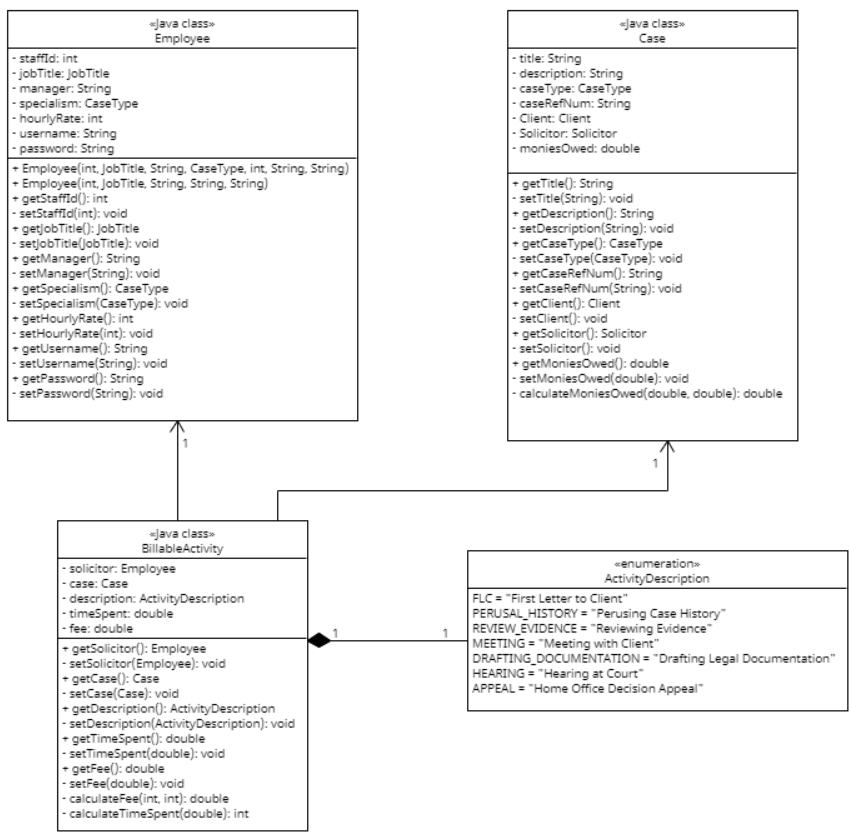


Figure BillableActivity - Associations and Composition

All of the discussion above reflects on how the evolving system will implement a solution relating to the business domain. The system will implement this in its own package, which was mentioned earlier under subsection Package Structure. The Initial Class Diagram (2) shows how the system will contain a package responsible for the user interface and a package responsible for the business domain. The program will be driven from a Driver class, which I intend to separate into its own package. This will also allow for testing as the application is developed.

### Refined Class Diagram

With all of the above taken into consideration, the class diagram on the following page attempts to illustrate the structure of the evolving system. This has also been provided in PDF format. From this diagram it is clear there will be three packages – one for the user interface, one for the business domain, and one to drive and test the application. It is important to remember that during development, details may be changed by the development team in the way the system is implemented. For example, the UserInterface class currently contains just a few properties, but there will definitely be more properties than this to construct the graphical user interface. Regardless, the system will be tested thoroughly to ensure it meets its requirements and updated diagrams with accompanying documentation will be provided.

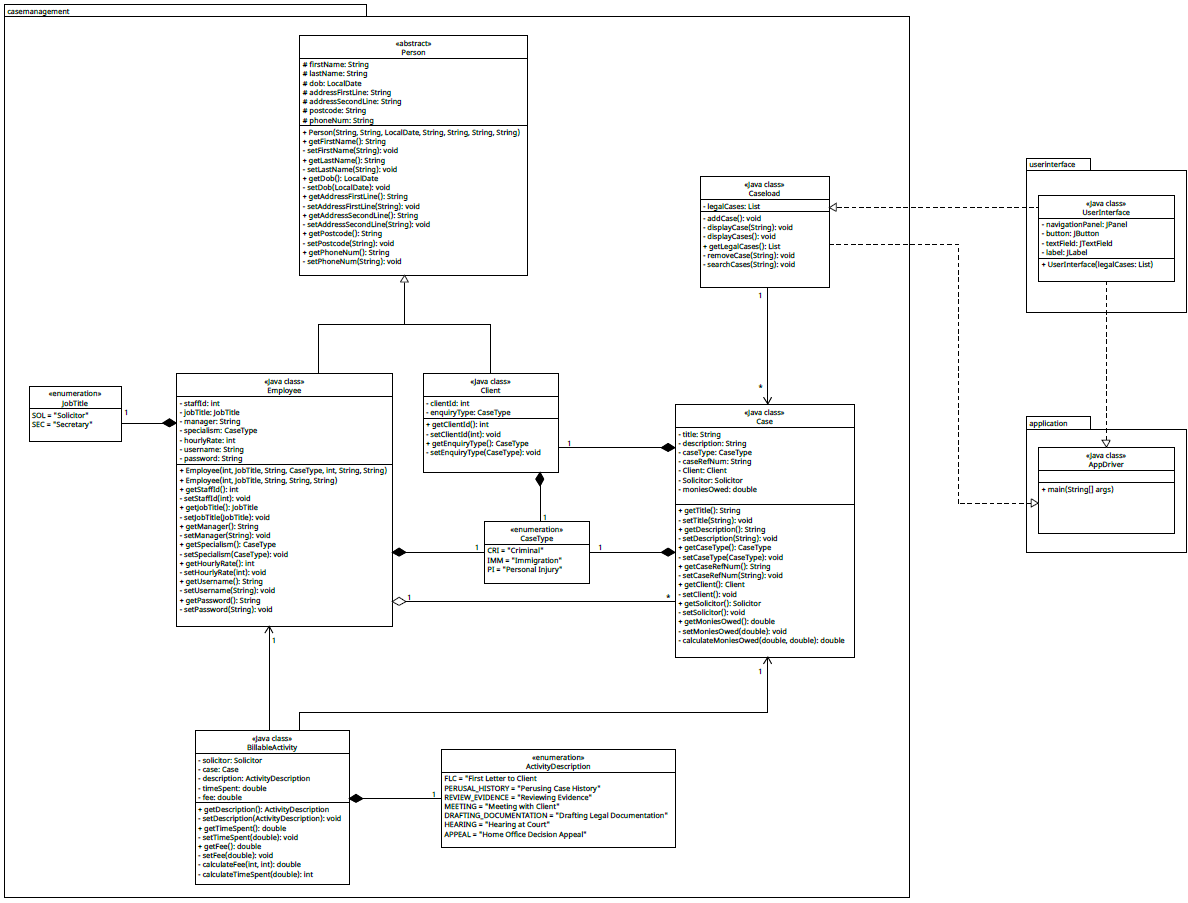


Figure Refined Class Diagram

## The Dynamic Model

So far, a lot of consideration has been given to how the evolving system should behave at compile time. There’s also a lot of information available to us about what our end users should be able to achieve when using the evolving system (the use cases). But none of this communicates in great detail what a user’s interaction will trigger in terms of system operations. Of course the use case descriptions make an attempt to describe what the system should do or store or validate, and the section Data Validation sets out what is considered valid data along with the relevant error messages that should be displayed. So, we have touched on how we want the system to behave when the user interacts with it in certain ways, including a time-independent view of the logical structure of software classes in the problem domain (the class diagrams). But to a development team, a sequence diagram is likely to mean much more and communicate more concisely how the system should behave at runtime. It’s important we do everything we can to build a successful system for the client, and an overload of textual information is unlikely to help all of the project team. Having visual representations of information is sensible and effective to ensure consistent communication. Larman describes sequences diagrams as:

*… a picture that shows, for a particular scenario of a use case, the events that external actors generate, their order, and inter-system events. All systems are treated as a black box; the emphasis of the diagram is events that cross the system boundary from actors to systems*. (Larman, 2005)

With this in mind, a sequence diagram can be used to keep track of communication, to control the ordering of communication between things. It conveys the direct links of communication, allowing us to map the process of object interactions. A sequence diagram shows the order of interactions and any messages exchanged between objects, and can also convey synchronous and asynchronous messaging. Synchronous communication is where a message is sent from the sender which is then processed by the receiver; the sender waits for a message to be returned by the receiver. A good example of synchronous communication is pressing keys on the keyboard: every letter comes up in the order that you type them; the synchronous communication here usually happens very quickly, and the user will probably not even notice that with each key that is pressed, a message is sent and processed by the receiver because it happens at speed. The drawback of synchronous communication is it can suffer from interference: if there’s a power surge, for example, it risks those messages being lost – if power is lost but a user keeps typing, those keystroke messages aren’t being sent or stored anywhere. Asynchronous communication refers to communication is typically concerned with operations. It is possible for asynchronous messages to have a return message, but generally there is just a message being sent and not returned. IBM offer a very good example of asynchronous communication:

*A bank customer could apply for credit but can receive banking information over the phone or request money from an ATM, while waiting to hear about the credit application*. (www.ibm.com, n.d.)

For a project of this complexity level, my sequence diagrams are concerned more with synchronous communication. Earlier in this report, The Use Case Model provides several use case descriptions. To lead us into designing the software system, we performed a second linguistic analysis on a number of these descriptions. I explicitly stated that this analysis focused on three use case descriptions (one of which will not be taken into development, as agreed with the client in meeting number 3), solely because at that point these three descriptions encompassed all other use cases and so closely relates to all necessary requirements that the system should meet. After client meeting number 3, the use case “Generate Report” was shelved, and another linguistic analysis was performed on “Record Billable Activities” as the client placed a lot of importance on this during meeting 3. It would not be a productive use of time to create a large number of sequence diagrams in this context; instead, I have focused again on three important use cases in creating the following sequence diagrams: “Log In” (2), “Create New Case” (1.1), and “Search Cases” (1.2).

### Sequence Diagram – Log In

#### Basic Log In

A series of steps to successfully achieve this use case was described in the earlier use case description. I have started by creating a sequence diagram that shows the **very basic** sequence of steps required for an employee to log in to the evolving system:

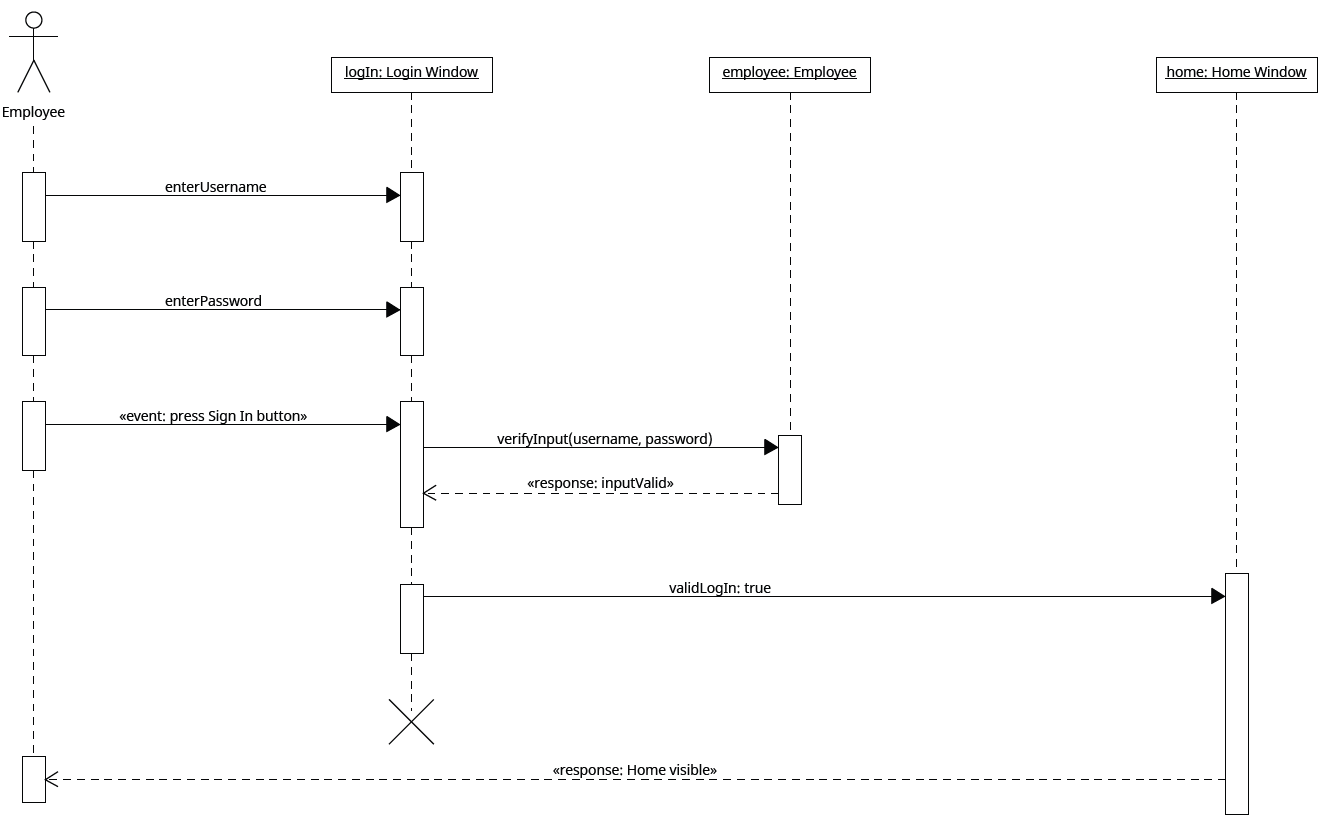


Figure Sequence Diagram - Log In

The diagram at Figure 26 attempts to communicate the following steps.

1. The Employee enters their username.
2. The Employee enters their password.
3. The Employee presses “Sign In” button.
4. The username and password are checked against the username and password stored in the relevant Employee object.
5. The username and password are a match.
6. The Employee is presented with the Home window.

#### Log In Validation

Now that the very basic sequence of steps has been laid out, we can now elaborate on this to communicate the validation requirements. I have included a brief description of the purpose and summary of the following sequence diagram for clarity.

**Purpose**: A sequence diagram to show links of communication for log-in process.

**Summary**: An Employee can log in with username and password, which must be validated.

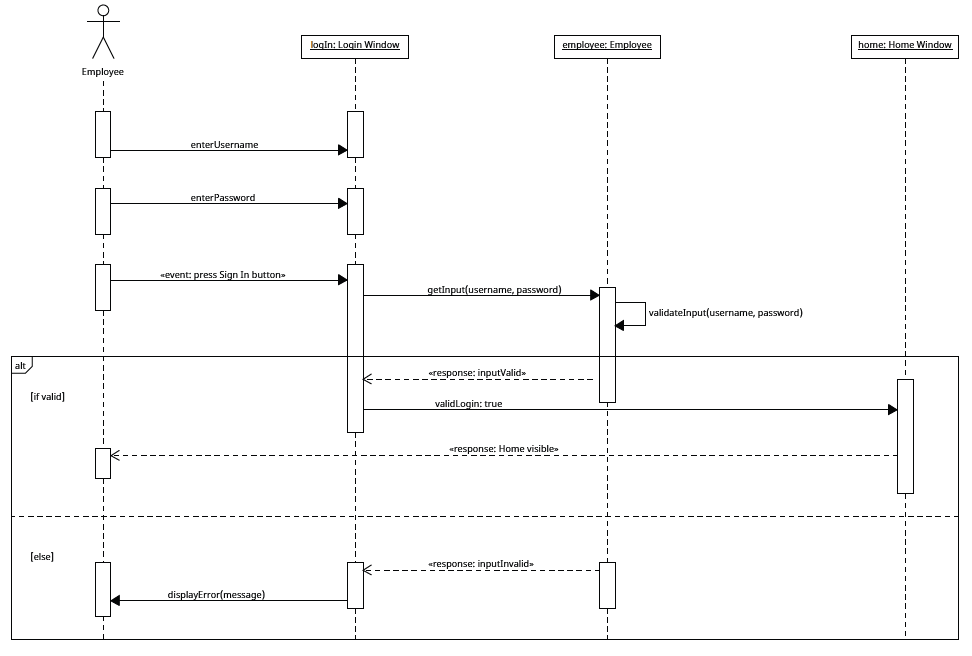


Figure Sequence Diagram - Log In with Validation

From the above diagram, we can now see that:

1. The Employee user enters their username to the Log In window.
2. The Employee user enters their password to the Log In window.
3. The Employee user presses “Sign In” button.
4. The data entered is sent to the Employee object (where details are stored).
5. The data is checked that it meets a condition.
6. If it meets the condition, the Home window is displayed to the Employee user.
7. If the condition is not met, the data is invalid.
8. An error message is displayed to the Employee user.
9. The process is repeated until the condition in the loop is met.

#### New Employee Log In

The sequence diagram at Figure 27 more closely reflects the description of events in the use case description for “Log In” by accounting for the validation of the user’s input. However, it doesn’t illustrate the communication between objects when the user is a new Employee that has to set their password.

The diagram below aims to show the ordering of communication when the Actor is a new Employee logging in to the system for the first time. I have separated this into its own diagram because a sequence diagram can become very unreadable very quickly if we include a lot of options and choices, as this usually results in the nesting of frames which isn’t really desirable or helpful.

**Purpose**: A sequence diagram to show links of communication for a new Employee setting their password.

**Summary**: A new Employee has to set their own password to create their staff account.

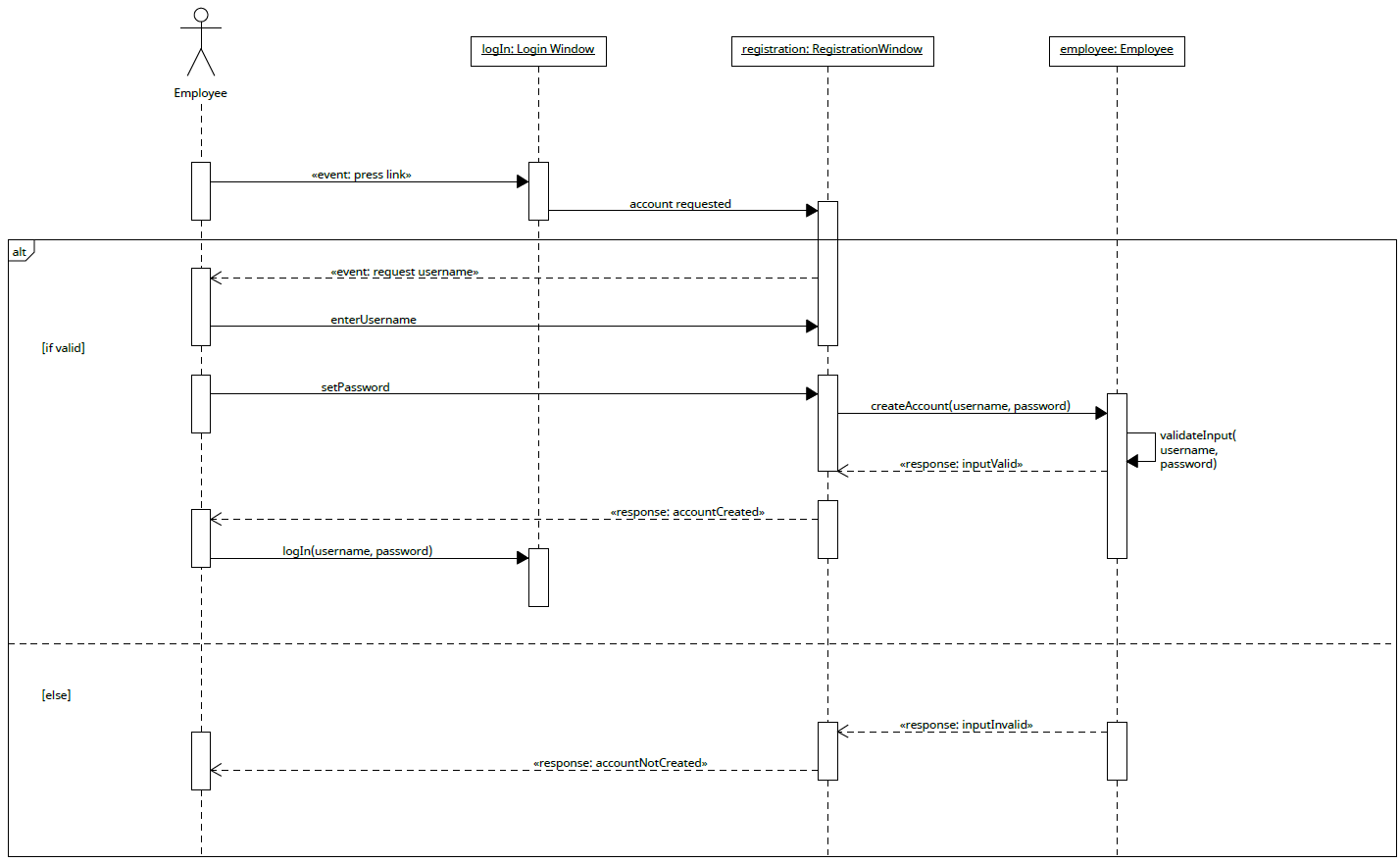


Figure Sequence Diagram - New Employee Log In

From the above diagram, we can now see that to complete the process of creating a staff account:

1. The Employee user presses the link in the Log In window to set their password.
2. The Employee user enters username when prompted.
3. The Employee user enters their desired password.
4. The data entered is used to construct a new Employee object.
5. If the data is invalid:
   1. the object is not constructed;
   2. the process is repeated until object receives acceptable data.

### Sequence Diagram – Create New Case

When the user presses the “Add New Case” button, they will be prompted to enter a variety of details that are all necessary to make up a Case object. You will see from the user interface for Add New Case that the user enters the data in one window, rather than having to navigate between different windows to enter different data. This is deliberate to make it easier for the user (it means less clicks of the mouse), and the client approved of this user interface design in client meeting number 4. As such, it makes the sequence diagram below a little busier. You will note I have not included nested frames to indicate where conditions should be met. This is because it becomes too unreadable with many lines crossing over. The diagram does include the pig’s ear notation to indicate that if details are not validated, it should continue to request data until it receives valid input, but the response messages are not shown so as to not overcrowd the diagram. The sequence diagram below communicates the basic order of communication when a new Case object is being added to the system.

**Purpose**: A sequence diagram to show links of communication for adding a new case to the system.

**Summary**: An Employee can add a new case by entering necessary details.

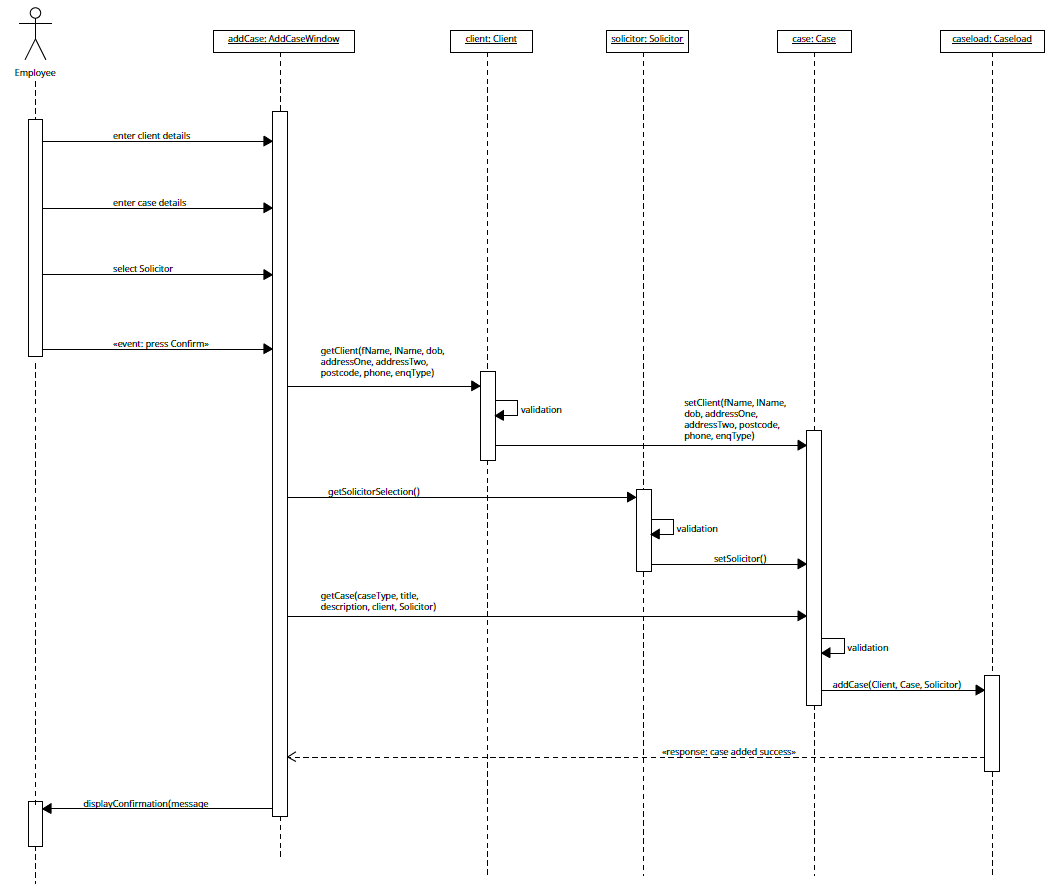


Figure Sequence Diagram - Create New Case

From the diagram above, we can see that:

1. The Employee user enters Client details in the AddCase window.
2. The Employee user enters Case details in the AddCase window.
3. The Employee user selects a Solicitor in the AddCase window.
4. The Employee user presses the “Confirm” button.
5. The Client data is sent to the Client object instance.
6. The Client data is validated.
7. The Client data is sent to the Case object instance.
8. The Solicitor data is sent to the Solicitor object instance.
9. The Solicitor data is validated.
10. The Solicitor data is sent to the Case object instance.
11. The Case data is sent to the Case object instance.
12. The Case data is validated.
13. The Case object is successfully constructed and added to the Caseload.
14. A confirmation message is displayed to the user.

As explained, the diagram at Figure 29 does not show return messages if validation conditions are not met, but if steps 5, 6, and/or 7 reveal input is unacceptable, the system should inform the user and request invalid input to be re‑entered. There is already detailed information regarding data validation included in this document which I believe, combined with the sequence diagrams, is enough artefacts for the development team to construct a successful system; this is preferable to creating unreadable and busy diagrams.

### Sequence Diagram – Search Cases

When the user wants to retrieve a specific case, they are required to enter a case reference number and press “Search”. The sequence diagram below represents the basic ordering of communication when this event happens.

**Purpose**: A sequence diagram to show links of communication for searching for a specific case.

**Summary**: An Employee can search for a case by entering the case reference number.

The Search bar will be visible on various interfaces, hence the generic naming of this object instance.

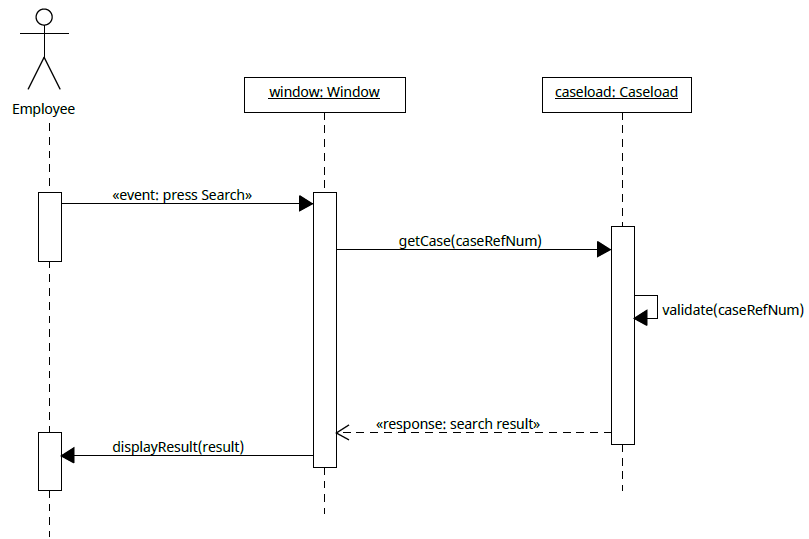


Figure Sequence Diagram - Search Cases

From the diagram above, we can see that:

1. The Employee user presses the “Search” button.
2. The case reference number is sent to the collection of cases stored in the system.
3. The case reference number is validated.
4. If the data is valid, the case is retrieved and sent to the graphical user interface window object.
5. The case details are then displayed to the user.

Although this sequence diagram illustrates that user input is validated, it does not show how unacceptable data is handled. It also doesn’t take into account that the use case description states that after searching for a case, the user can delete that case. Unlike the sequence diagram for creating a new case, this sequence diagram is not too crowded. The diagram below includes the sequence of messages exchanged when error handling is considered and the choice to delete a case.

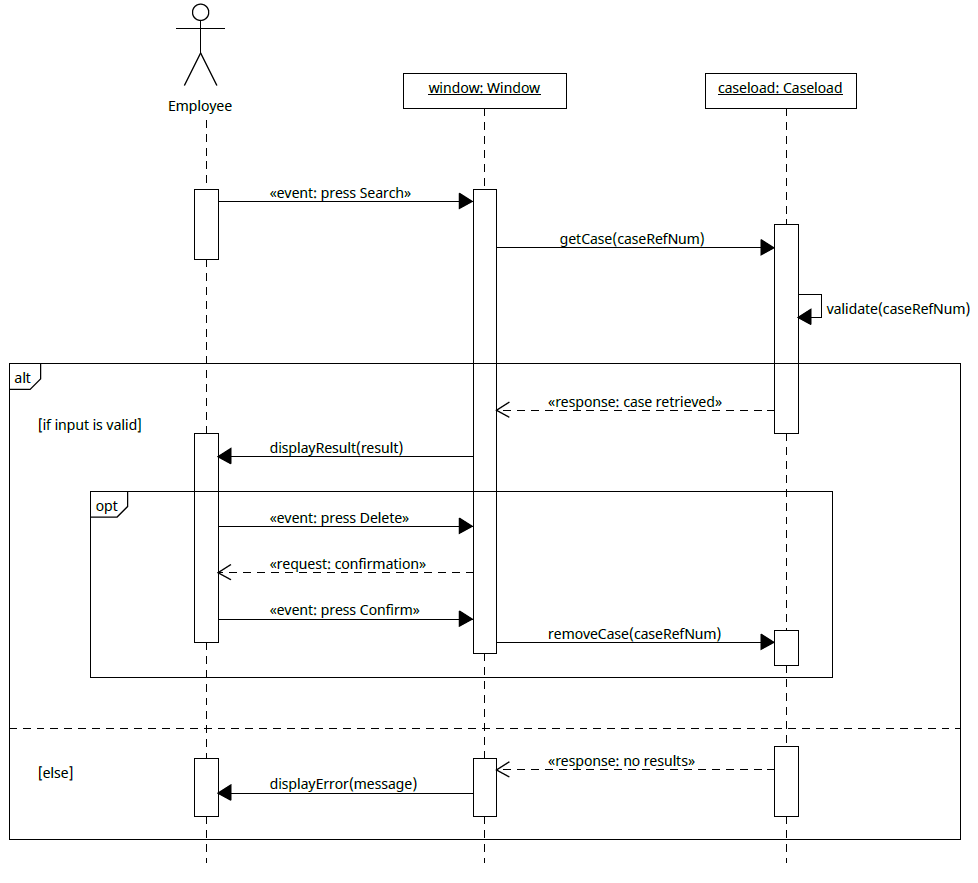


Figure Sequence Diagram - Search Cases with Delete Option

With these details added, we can now see from the diagram at Figure 31 that:

1. The Employee user presses the “Search” button.
2. The case reference number is sent to the collection of cases stored in the system.
3. The case reference number is validated.
4. If the data is valid:
   1. the case is retrieved and sent to the graphical user interface window object;
   2. the window then displays the case details to the user;
   3. the user has the option to press “Delete”:
      1. the user is prompted to confirm their action;
      2. if the user confirms, the case is removed from Caseload.
5. Else:
   1. an error message is displayed informing the user input is invalid.

It is important to note that at this stage in planning the client’s solution, all artefacts are subject to change during implementation of the planned solution. That is to say that if the development team discover a flaw in the diagrams presented here, details may be changed and artefacts will be updated.

### Activity Diagrams

With the ordering of communication set out for some of our key use cases, it is a good point now to devise activity diagrams to complement these sequence diagrams.

An activity diagram is a very accessible visual representation of how the system will achieve its use cases owing to its use of flowchart‑like notation. It’s beneficial both for those with technical knowledge and without as it sets out clearly a chain of events that, overall, represents a business process. In Learning UML 2.0, an activity diagram is described as “useful for describing processes to a broad audience” (Miles and Hamilton, 2006).

The diagram’s emphasis on business processes means it also complements the use case model nicely, too. In fact, the activity diagrams could well have been created after the use case model was set out. However, they also complement the sequence diagrams just as well, and as they are similar to flowcharts (their roots are in flowcharts and data flow diagrams, according to Miles and Hamilton), it seems appropriate to create them after considering the ordering of communication in the sequence diagrams.

#### Activity Diagram – Log In

I have started by providing a simple activity diagram for a successful log in process (Figure 32).

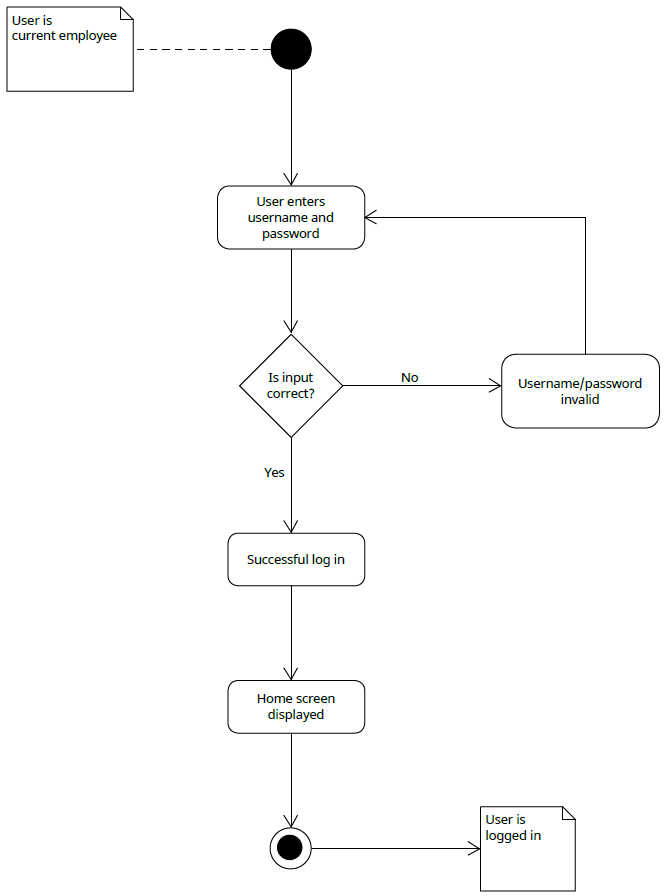


Figure Activity Diagram - Log In

The diagram above shows at the start of the log-in process, the user is a current employee, so this diagram is not concerned with generating any usernames or the setting of a new employee’s password. Its value is in its simplicity – if the sequence diagram becomes unreadable or busy (and it easily does), the activity diagram breaks this down for us by asking a series of questions. From Figure 32 we can easily see that the log-in process is described as:

1. The user enters their username and password
2. If it’s correct, they’re logged in. The home screen is presented.
3. If it’s not correct, go back to step 1.

It clearly suggests the log-in process should be within a loop. As the activity diagram implies that the user is asked for their username and password at least once, it would be appropriate to use a do-while loop.

What the activity diagram at Figure 32 doesn’t show is the process for a new Employee logging in. Below illustrates the steps involved before a new Employee is able to log in to the system:

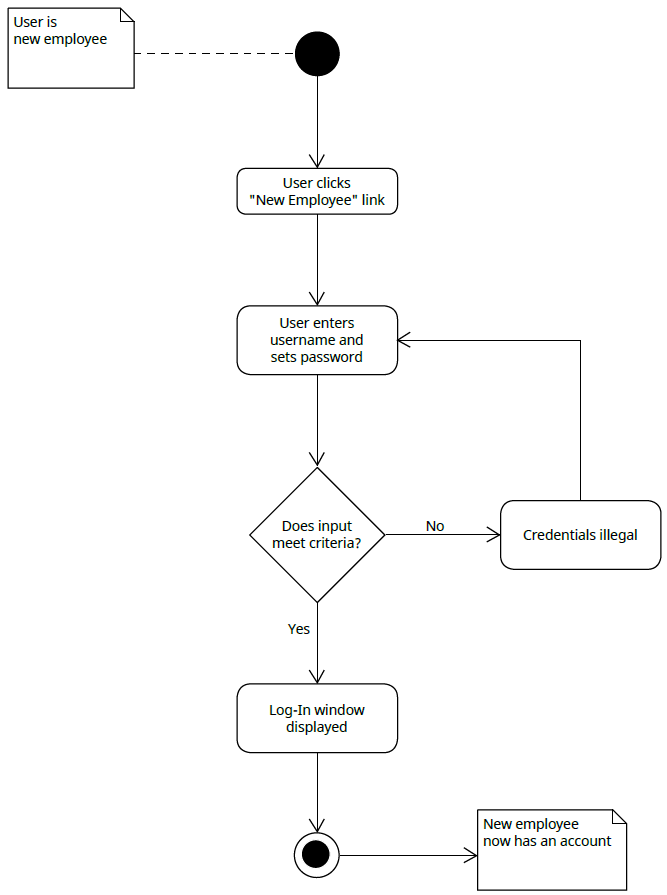


Figure Activity Diagram - New Employee Log In

It is obviously very similar to Figure 32, with similar questions being asked at decision points. At the finish point of this activity diagram, the new Employee now has log‑in credentials and can follow the process described at Figure 32.

#### Activity Diagram – Create New Case

Another key use case is “Create New Case”. The diagram below represents the process of adding a new case to the system.

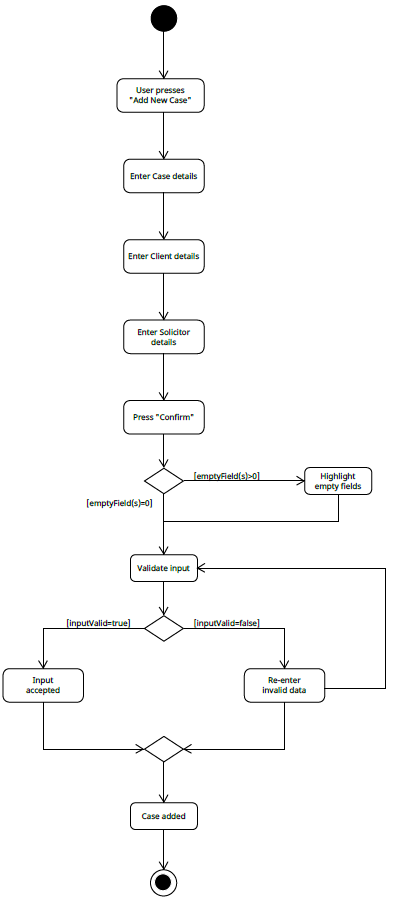


Figure Activity Diagram - Add New Case

The diagram at Figure 34 complements the use case description for adding a new case to the system as it also conveys the user will have to press the “Add New Case” button, enter all relevant details before a case can be created. It also describes that the user cannot leave any fields empty, as stated in the Data Validation section. If all fields are filled, the program can proceed to validate all of the data. If the user’s input is all acceptable, the case is added to the system; if not, the user is prompted to re‑enter acceptable data. This will repeat until input is valid, at which point the case will be added to the system.

#### Activity Diagram – Search Cases

Another key use case identified in The Use Case Model was the ability to search for a case. As detailed use case descriptions have been provided for the options that are available to the user after a search has returned results, I have included an activity diagram to reflect one of those options to ensure it remains readable and comprehensible. The use case description for “Search Cases” describes that a user may remove a case from the system by pressing the “Delete” button. The reason I have chosen to represent this as an activity diagram is that I want to emphasise to the development team that this must trigger a message to be displayed that requires the user to confirm their action. There is a discussion provided later in this report (in Section 2 – View Model) on Human Computer Interaction (HCI), and, in summary, I want the user to be able to use the system confidently. If they can build up trust in the system, then the system can be described as accessible. The ability to delete a case hasn’t really had much attention so far, so I have deliberately chosen to represent this functionality in the activity diagram below, mainly to highlight what should happen if the user presses the “Delete” button after searching for a case.

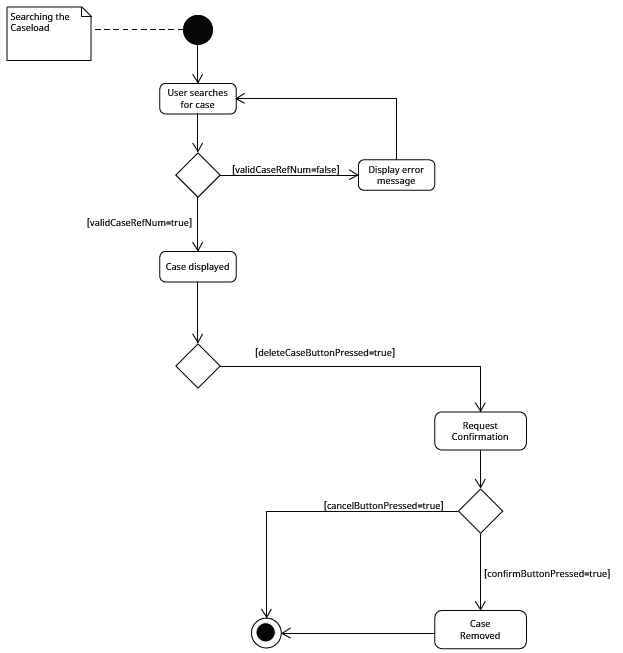


Figure Activity Diagram - Search Cases

# Section 2 – View Model

## User Interface Design

Before personal computers became commonplace, generally a computer user would be a computer scientist who had specific knowledge of how to interact with a computer. From the late 1970s onwards, the personal computing industry began to grow and with the introduction of a graphical user interface (GUI) in 1983 by Apple, sales increased globally. It is estimated that that the market size of personal computers will increase from $200.59 billion in 2022, to $214.06 billion in 2023 (www.thebusinessresearchcompany.com, n.d.). Needless to say, with computers being accessed by more and more commoners as time went on, how to interact with the machine became an issue as it was no longer confined to use by skilled computer scientists, giving rise to the concept of human computer interaction (HCI). HCI is “*tremendously broad in scope*” and it intermingles a wide variety of disciplines, specifically “*psychology (particularly cognitive psychology and experimental psychology), sociology, anthropology, cognitive science, computer science, and linguistics*” (Scott Mackenzie, 2013).

A study of HCI is not the purpose of this document, though. It is simply the key aspects of HCI that should inform our design of the evolving system. In designing the user interface, it is important to understand that there is a strong correlation between the Information Architecture (IA) of any software that a human user can interact with and a system that is intuitive and accessible. It is in the development of high-quality IA that eliminates users' frustration and confusion. IA is commonly discussed in a website development context, but if a user cannot find the content or functionality they are looking for in any digital application – whether that be an industry‑specific application such as this legal case management system, or whether it is an ecommerce website – then the user will inevitably spend more time trying to figure out how to navigate the system than they do on productive tasks that the system is built to help them with. User Experience (UX) architect Nick Babich highlights the consequences that poor IA can have:

*When finding information becomes too complicated or too slow, there’s a risk that people will simply abandon it. And when people abandon an app or a website, it’s more difficult to bring them back. This is where information architecture design plays a key role.* (Babich, 2020)

To a non‑technical user, the IA is invisible. This user is unlikely to be able to make a valid judgement on the quality of the IA of the application. However, a user's experience of using your software can reveal significant strengths or weaknesses of the IA of the system. If the user is swearing at your system and ready to throw their computer out the window, then it is likely you need to iterate over analysis & design workflows of the UP.

Our client runs a busy legal firm, and we have already established in our Action Plan report that Solicitors spend a lot of time on administration tasks, so the aim of the following subsections is to define a high quality IA by addressing: page layouts in the form of wireframes, a prototype of the system, an analysis of client interaction, a data binding design which includes a mapping document, a style guide that offers evaluative conclusions on usability and accessibility features, colour schemes, and font choices, and finally analysis of further client interaction.

### Wireframes

I identify above that HCI concerns many disciplines; Babich states that IA "*has roots in various fields, cognitive psychology is one of the most important, because this discipline defines the way we structure information*", and lists three fundamental aspects of cognitive psychology that are most influential in IA: Gestalt principles, mental models, and cognitive load (Babich, 2020). I will focus just now on cognitive load, and will explain the relevant of Gestalt principles and mental models as our design progresses.

More broadly, cognitive load refers to the dependence on short term memory to process information that is unfamiliar. For example, if a school pupil is struggling to cope with a classroom environment, their cognitive load is increased because they are dealing with this unnecessary demand along with learning whatever is being taught. Another example is the disruption brought by the Covid‑19 pandemic: many people were forced into making a very sudden change of routine and working remotely meant people may have had to learn how to configure and navigate video‑conferencing software. The working memory is very limited, and as such any task that cannot be done on "auto‑pilot" activates the working memory: a conscious effort is required to adapt to this new routine.

These wireframes have therefore been designed with a view to developing a system that is simple to understand and navigate, which I hope allows our client to adapt easily to their new system.

#### Log In

The user will be required to log in to access the system. Figure 36 below illustrates the basic look of the Log In page.

Figure Log In Wireframe

Company logo

Input username

Input password

Sign In button

#### Home

On successfully logging in, the user will initially be presented with a Home page.

Figure Home Page Wireframe

Search bar

No. of Criminal cases

No. of PI cases

No. of Immigration cases

No. of total cases

Navigation menu

“Add New Case” button

“View All Cases” button

“View Criminal Cases” button

“View Immigration Cases” button

“View PI Cases” button

“Generate Reports” button

#### Searching

From the Home page, the user will be able to enter a case reference number in the search bar. The wireframe blow shows how search results will be laid out *after* a valid reference number has been entered.

Navigation menu remains the same

Search bar remains obvious

Figure Searching - Wireframe

Results of a successful search

Actions

Amend Case Details Button

Add Billable Activity Button

Delete Case Button

Case reference number

Client address

Client phone number

Client name

Case description

Solicitor name

Case type

Case title

#### Recording Billable Activities

After searching for a specific case, the system then allows the user to record any billable activities relating to that case. The following wireframe shows the page layout if the “Add Billable Activity” button is pressed,

Search bar remains obvious

Navigation menu remains the same

Select Activity Label

Enter Time Spent Label

Total Amount Owed Label

Total Amount Owed Figure

Confirm Button

Input time spent here

Select Activity Dropdown

Figure Recording Billable Activities - Wireframe

#### Add New Case

The system proposal in the Action Plan report identifies that the navigation menu will have various buttons. The wireframe below demonstrates the layout of the screen when the user presses the Add New Case button.

Navigation menu remains the same

Add New Case Button

View All Cases Button

View Criminal Cases Button

View Immigration Cases Button

View PI Cases Button

Generate Reports Button

Case Details Label

Search bar remains obvious

Client Details Label

Case Type, Title, Description entered here

Client name, DOB, address, postcode, phone num, enquiry type entered here

Confirm New Case Button

Solicitor Details Label

Responsible Solicitor selected from dropdown list

Figure Add New Case - Wireframe

#### View Caseload

It is clear from Figure 37 that there is a “View All Cases” button in the main navigation panel. The wireframe below reflects the page layout after this button is pressed.

Search bar remains obvious

Figure View Caseload Wireframe

Table headings

Case Reference Number

Client Phone Number

“Home” button

Case Type

Client Name

Solicitor Name

#### Our Staff

Meeting number 3 gave rise to a new use case – the ability to add a new employee to the system. This will allow the user to generate a new username for that new employee. To achieve this use case, the user will press the “Our Staff” button in the navigation panel, which will present them with a screen displaying a list of employees. The wireframe below shows the basic layout of this page.

Search bar remains obvious

Figure Our Staff - Wireframe

“Job Title” table heading

“Employee Name” table heading

“Add New Employee” button

#### Add New Employee

From the “Our Staff” screen, the user will be able to add a new employee to the system. After pressing the “Add New Employee” button, the user will be taken to another screen where the new employee’s details can be entered. The wireframe below represents the basic structure of this screen.

Figure Add New Employee - Wireframe

Search bar remains obvious

“New Employee” Details Label

Employee name, DOB, address, postcode, phone num, job title, job hierarchy entered here

“Confirm” Button

### User Analysis

We can see from the wireframes above that the structure of the user interface should remain simple throughout the whole system. This is not to insult the intelligence of the end‑user. We know the end‑users are educated people – they are qualified Solicitors and Secretaries after all – but we still cannot take for granted that they have background knowledge in computing science, and there will always be a risk of human error no matter how simple the technology is. In considering the style guide for the evolving system, I have assumed that the age range of the firm’s employees may range from young adult (late teens, early 20s) to middle‑aged (maximum age will probably be the UK retirement age which has changed in recent years, but we can take the average retirement age to be around about 64). There are many steps we can take when developing the evolving system to adhere to good design heuristics to ensure it is fit for use by a diverse range of users and is accessible regardless of age.

We know the users are working in a busy firm. It is therefore likely that at very busy times, the users of our evolving system might not be paying full attention to the system in front of them, perhaps trying to multi‑task whilst searching for a case. This can naturally lead to human error. The description for Use Case: Delete Case (1.2.3) specifies that an information message should be displayed to the user if they press the “Delete” button. This is because the user might be having a blonde moment (we all have them) and accidentally presses the “Delete” button. Instead of inducing a panic attack in the user, who for a moment believes they’ve lost all the details of that case, we can simply program the system to display an information message informing the user that they have pressed “Delete” and request that they press “Continue” if this was intentional, or press “Cancel” if not. The activity diagram at Figure 35 illustrates this process. In doing so, we’re passing some control over to the user who will build up a trust in the system because it has saved them from making a mistake. This doesn’t require much effort on the developer’s side, and it makes the system far more user friendly by giving them a second chance to confirm their intentions.

A large part of good quality HCI design is ensuring consistency throughout the system. If the user sees graphics or icons they are familiar with, we are reducing the cognitive load on them because they don’t have to overthink what an unfamiliar icon implies or represents. You will see in the interface for Searching cases that the “Delete” button is coloured red and has the text “Delete” written on it. This might seem like a minor detail and not worth mentioning, but it’s really important the user can easily infer what the interface controls will trigger.

In addition, the following interfaces make use of helpful text that is displayed by default in any field that is to accept user input. This is an excellent technique to make the system intuitive – if the Log In screen had two empty boxes, then we’re assuming that the user knows to enter the username in the top box and the password in the lower box. It is never advisable to assume anything – we want to make sure the correct data goes through the correct validation routines, so we absolutely need the user to enter the username in the username box, and the password in the password box. You will also note that the Log In interface is kept very minimalistic. I have designed this interface like this because I don’t want to overload the user with textual information. All the user needs to know is where to enter their username and where to enter their password. As the system is being used in a workplace environment, it is likely it will be used by employees on a daily basis, so they do not need constant reminders that the username must be in a specific format, or that their password must meet a minimum criteria. I also implement this principle in the design of the Add New Case interface. The field that requires a date of birth to be entered very clearly shows that the system expects this data in a specific format (DD/MM/YYYY). There are a wide variety of formats that digital applications accept dates of birth, so if the interface doesn’t display this informative text it can cause frustration in the user and can result in time being wasted; the user might have a lot of work to do and probably doesn’t have time to waste on repeatedly entering a date of birth in different formats to see what’s correct. It’s so much easier for the user if they are informed of how they should input data as it minimises the cognitive load: they don’t have to think too much (we wouldn’t want that!) about how to interact with the system.

To design an accessible interface, all components should be consistent regardless of what the user is viewing. This means that the user only has to interpret it once, and then isn’t surprised with anything new, keeping that cognitive load as low as we can. I have designed the interfaces with this in mind and have included a vertical navigation menu on the left hand side of the screen that is frozen to that position all the time. This allows the user to be able to access any of the system’s functionality at any time, regardless of where they are in the system. Given that our client will be using the new system on a desktop computer, there is currently no need to adapt the design of the system to mobile devices. The use of a simple layout of components though will make it easier to make the system compatible with mobile devices – we identified this as an advantage of adopting a Java project as our development route: a platform‑independent programming language coupled with a simple, accessible interface design means adapting the system in the future will be less cumbersome.

### Style Guide

I noted earlier when setting out use case descriptions that Larman describes one of the most valuable exercises a team can do is to write text. But simply writing text that describes the logic of the system alone isn’t really what makes a digital application accessible. Designing a system that is accessible requires a combination of activities. A system with sound logic contributes to making it accessible (so it behaves as we expect it to behave), but the visual presentation of a system is equally as important. We should remember, as mentioned earlier, that computer users historically were qualified scientists. But now, people with all sorts of abilities use information systems. Had we opted for development route option 1 (a web solution), we would be able use the WCAG guidelines to ensure our system meets their Success Criterion. However, many of the same principles apply. In fact, the background research performed on similar systems in the AP report explicitly makes reference to the aesthetics of iManage’s graphical user interface and how this made the system difficult to use. So, to ensure accessibility in the evolving system we are developing, I have designed the interfaces with a high contrast so that components are easier to locate. The initial project brief states that the company colours are black, white, and green, so I intend to implement a colour scheme that is primarily monochromatic. Although the WCAG guidelines are aimed at web development, it is still useful to use WebAIM’s Contrast Checker to check if the main colour scheme complies with the guidelines – the principles behind colour choices are the same regardless of the digital application.

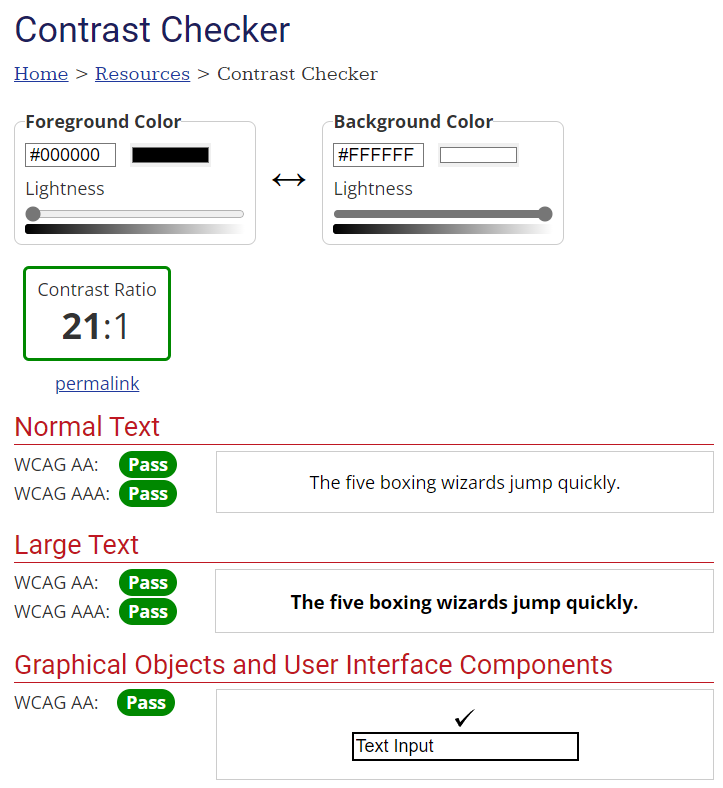


Figure GUI Colour Scheme - Contrast Checker

Also helpful in ensuring an application is accessible to users with colour blindness is ACart’s contrast checker, which tells if a given colour scheme is suitable for users who are colour blind.

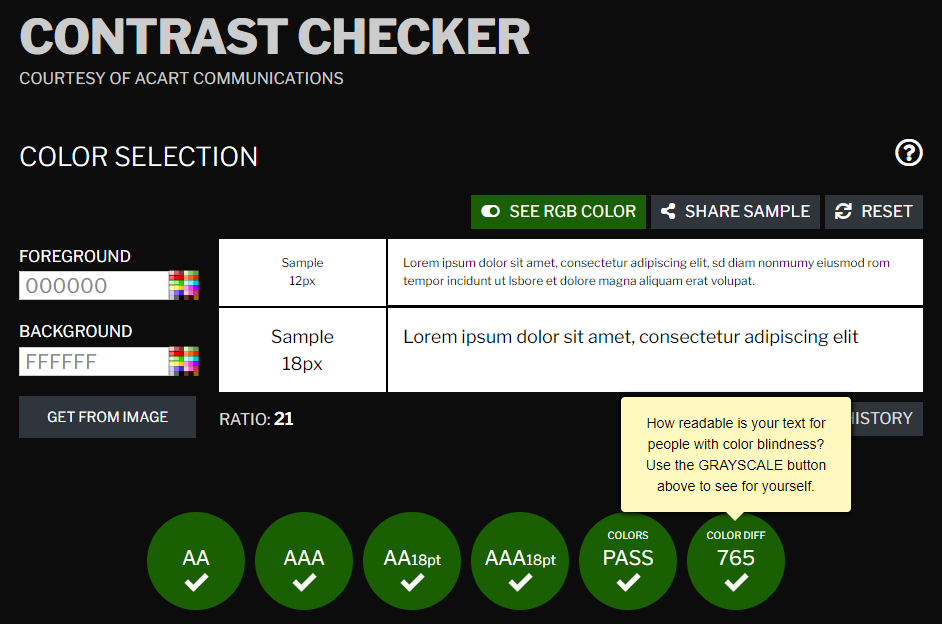


Figure GUI Colour Scheme - Readability

I have also designed the following interfaces using sans‑serif font. In general, there is no single font type that is optimal for every end‑user. Again, the simpler it is to read and interpret, the less difficult it is to navigate.

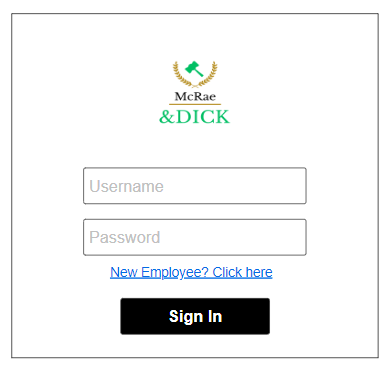
### User Interfaces

The graphical user interfaces below have been designed with the points discussed above in mind. They implement a simple and consistent minimalistic design that aims to reduce cognitive load on the user.

#### Log In

We know from client meeting number 3 that the client is happy with the basic layout of the Log In wireframe. As no changes are needed to the structure of this, the following interface demonstrates what the Log In screen will look like when a user initiates the system. It does include the addition of the link to enable new employees to create a password.

Figure Log In - User Interface



#### Home

If log in is successful, the user will be presented with the following screen:

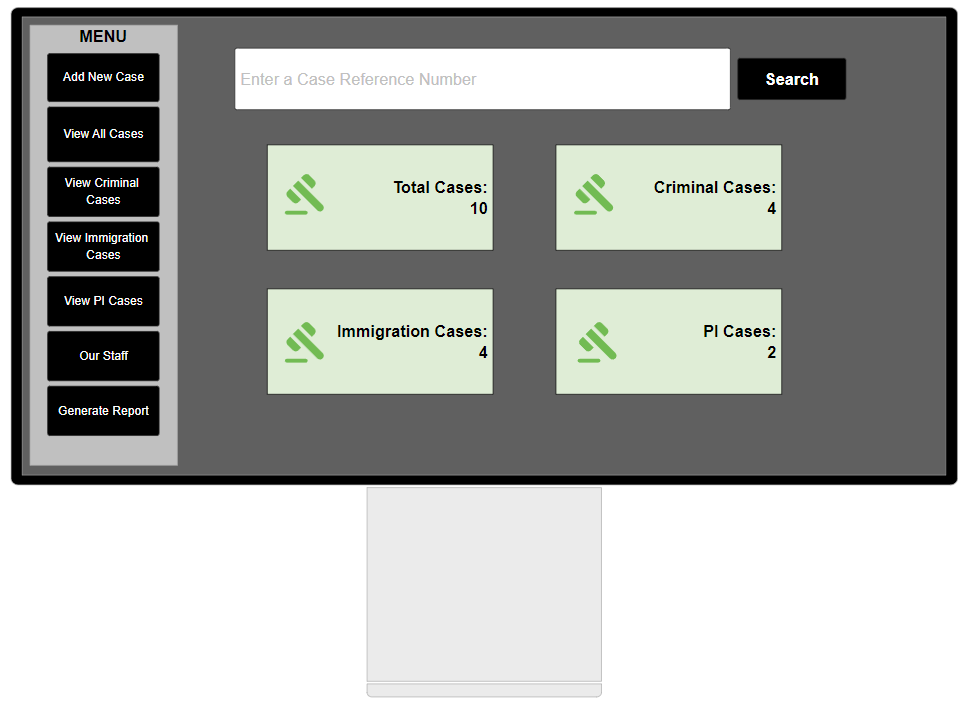


Figure Home - User Interface

#### Add New Case

From the home screen, the user may press the “Add New Case” button, which will display the following screen:

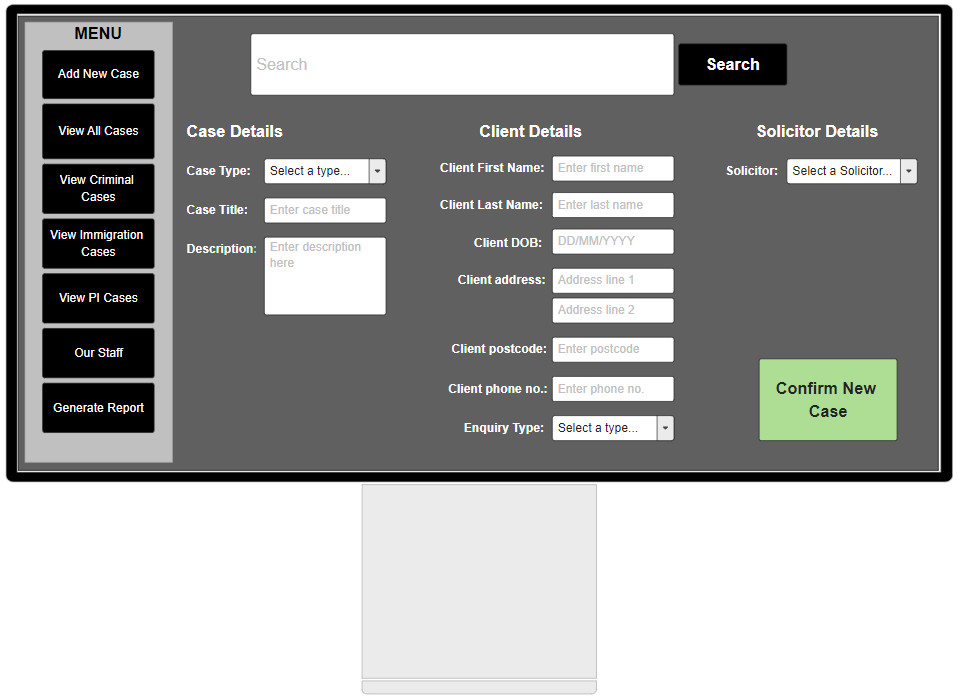


Figure Add New Case - User Interface

#### Searching

If the user performs a search on a valid case reference number, the results of that search will be presented as follows:

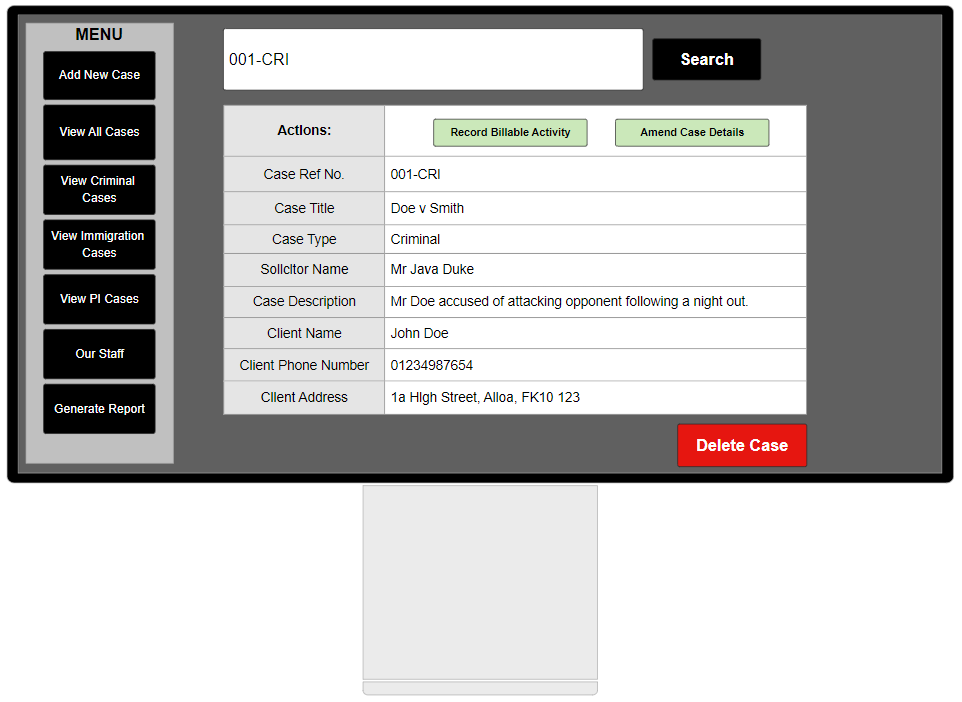


Figure Search - User Interface

#### Recording Billable Activities

After searching for a case, the user may press the “Record Billable Activity” button. This will present the user with a screen which allows them to enter a billable activity’s details and the time spent on that activity. Once valid details are entered and the user presses “Confirm”, the activity fee and total amount owed for this case is displayed:

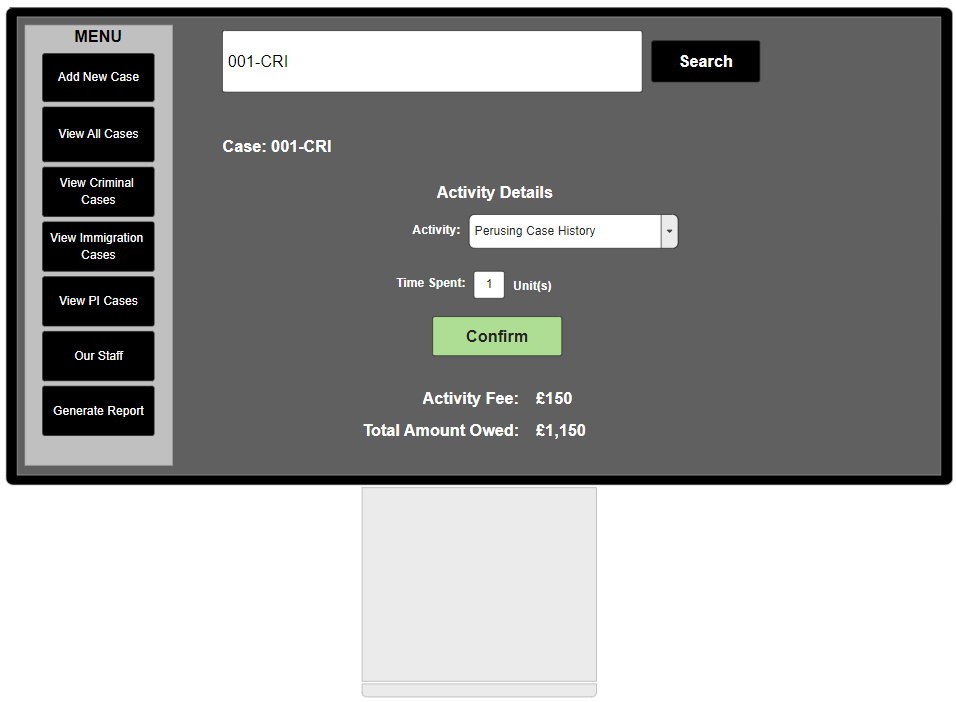


Figure Record Billable Activity - User Interface

#### View Caseload

From the navigation panel, the user may press the “View All Cases” button. This will present the following interface to the user:

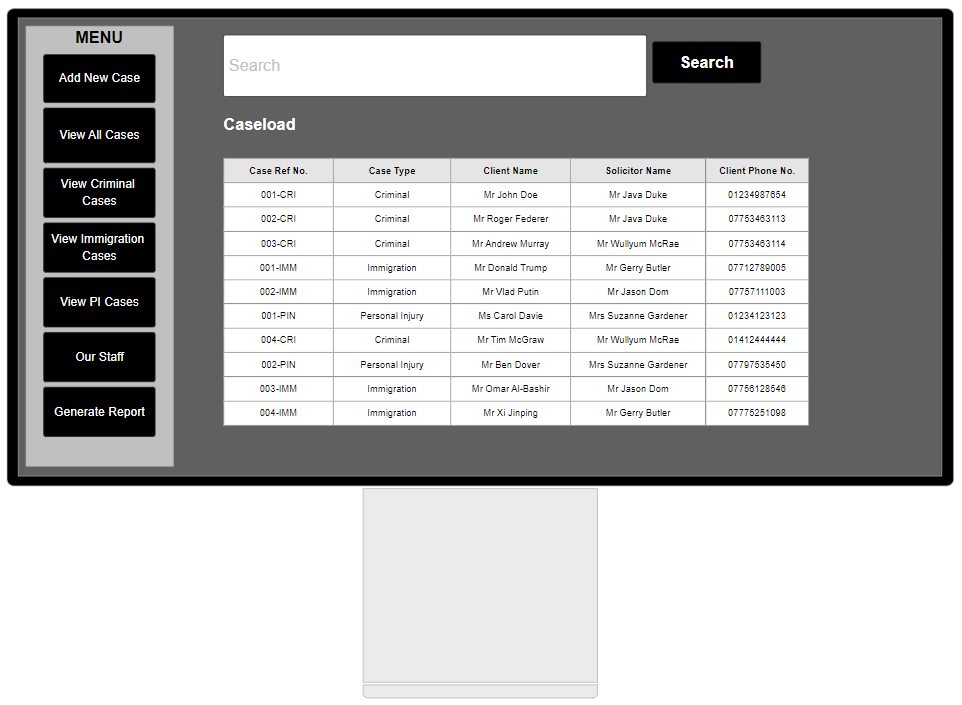


Figure View Caseload - User Interface

#### Our Staff

If the user presses the “Our Staff” button in the navigation panel, a screen will be presented that displays the firm’s current employees. From here, the user may press “Add New Employee”.

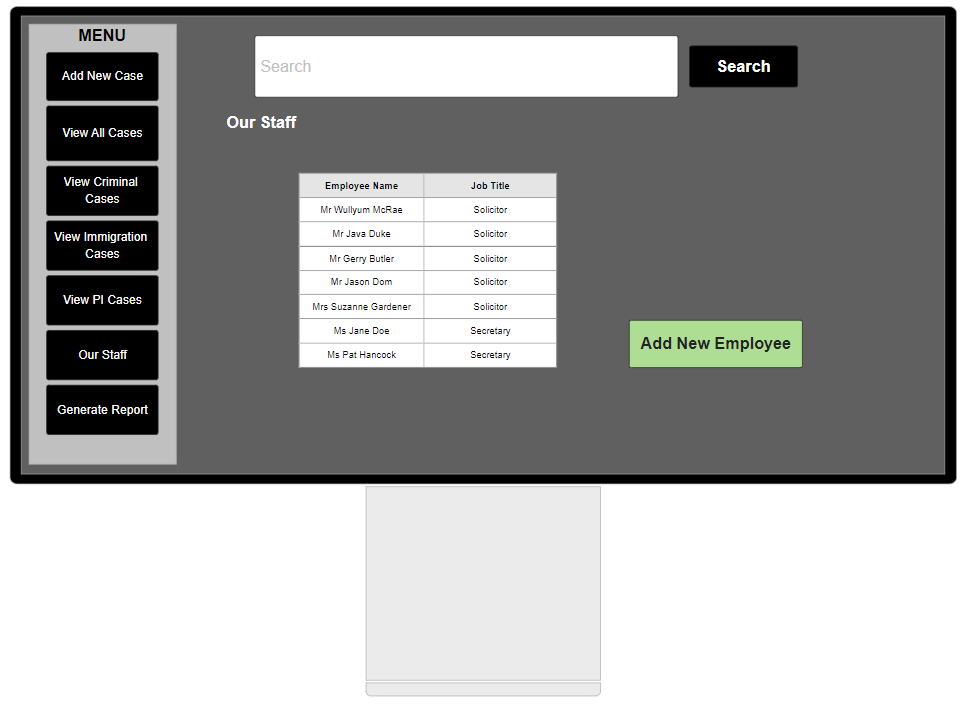


Figure Our Staff - User Interface

#### Add New Employee

If the user presses the “Add New Employee” button shown in the “Our Staff” interface, they will be able to add a new employee to the system by providing the relevant details.



Figure Add New Employee - User Interface

If the user presses “Confirm”, the system will display a confirmation message along with the new username, which in this case would be “daria.vekic”.

### Client Feedback

The fourth meeting with the client primarily involved presenting the user interfaces to the client for their approval. Some clarification was sought as to whether their staff and clients already had id numbers as a result of the discussion provided after Initial Class Diagram (1) provoked discussion of implementing inheritance in the context of a Person class.

This meeting was really a key opportunity for the client to say whether the design is going in the right direction or not. It also allowed for confirmation of some business processes, such as how a new employee will set their new password.

#### Meeting Number 4 Agenda

The following agenda was issued to the client prior to the fourth interview.

**Meeting 4 to be held at 10:20 on 08/03/2023**

**Falkirk Campus, Forth Valley College**

|  |  |
| --- | --- |
| **Present:** |  |
|  |  |
|  |  |
| **Apologies:** |  |
|  |  |
| **Chair:** |  |
| **Minute Taker:** |  |

1. **Minutes of Previous Meeting**
2. **Matters Arising**

We’d like the system to be able to assign identification numbers to your employees and to your clients. If your employees and clients already have this, please specify the current format.

1. **User Interface (5 mins)**

Please have a close look at the following user interfaces and specify if there are any details (no matter how small!) that you would like changed, or if you have any preferences or concerns about them.[[4]](#footnote-4)

* Log In – added a link for new employees to follow. Will allow them to set their own password. **Please specify if you would rather have the system generate a password for the user.**
* Successful log-in results in Home screen
* Adding a new case to the system
* Searching for a case
* Recording Billable Activities
* Viewing all cases
* Our Staff
* Adding a New Employee to the system

1. **Any Other Business (2mins)**
2. **Date of Next Meeting (<1 min)**

To be confirmed.

#### Meeting Number 4 Minutes

The following meeting minutes were prepared after the fourth client interview.

**Meeting 4 held at 10:20 on 08/03/2023**

**Room 0.021, Falkirk Campus, Forth Valley College**

|  |  |
| --- | --- |
| **Present:** | Daria Vekic (DV), Project Manager |
|  | Susan Gardner (SG), Client |
|  |  |
| **Apologies:** |  |
|  |  |
| **Chair:** | Daria Vekic (DV), Chairperson |
| **Minute Taker:** | Daria Vekic (DV), Minute Secretary |

1. **Minutes of Previous Meeting**

There were no issues highlighted in the minutes for meeting 2.

1. **Matters Arising**

DV explained that the project team would like to assign identification numbers to employees and clients. The client stated that they do not have a format preference for employee and client ID numbers. As this is a new system, the client is happy for the project team to make this decision.

1. **User Interface (5 mins)**

The client was presented with user interfaces that represent the intended aesthetics of the evolving system. DV also made reference to the main use cases when presenting the user interfaces.

DV highlighted that there is a link in the Log In interface to indicate that a new employee must press this to set their password. The client confirmed that this complements their business process of a new employee setting their own password as opposed to the system generating a password. DV highlighted that the system will ensure the new employee’s password meets the minimum password criteria that the client specified in meeting number 1 held in January.

The remaining user interfaces were discussed and DV explained how the user will navigate the system. DV drew attention to various fields that require user input and indicated where the user will be able to select a value from a dropdown list. The client is happy with the design of the user interface and no changes have to be made.

1. **Any Other Business (2 mins)**

No other business.

1. **Date of Next Meeting (<1 min)**

To be confirmed.

#### Summary

The meeting held on 8 March 2023 with the client confirmed that there is no preference with regard to the format of employee and client ID numbers. As the client is also happy with the user interface designs, it is now a good point to provide extra details of the data binding design.

## Data Binding Design

The data binding design aims to illustrate where data will go in the back‑end of the evolving system. We have designed various user interfaces, but these don’t operate by themselves alone. The graphical user interface doesn’t take care of data storage for us. We have already touched on the data binding design earlier in Section 1 – Business Model, where some discussion regarding potential data structures was provided. Now, we can map the data from the GUI to its data structure in order to connect the view model to the business model.

I have split the data binding model up so that mapping is shown for each of the user interfaces presented to the client in meeting number 4. There are relevant data dictionaries included also for clarity.

### Data Mapping – Log In

1

2

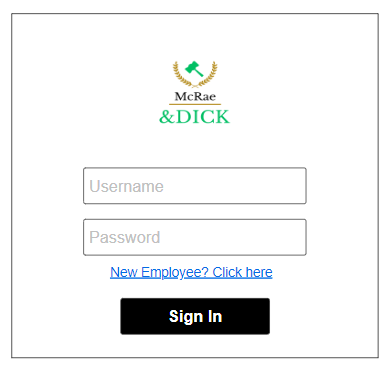


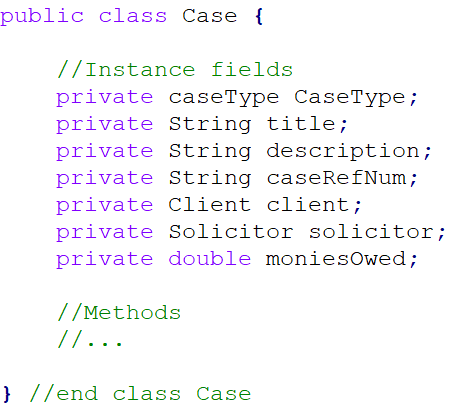
Figure Data Binding - Log In

|  |  |  |
| --- | --- | --- |
| **Data Dictionary** | | |
| **Num** | **Data** | **Description** |
| 1 | username | Current design shows username will be stored as part of an Employee object.  Needed for Employee to log in successfully. Included as String for now, but will probably change this in implementation. |
| 2 | password | Current design shows password will be stored as part of an Employee object.  Needed for Employee to log in successfully. Included as String for now, but will probably change this in implementation. |

### Data Mapping – Add New Case (Case Details)

For purposes of clarity, I have cut the relevant fields from the “Add New Case” interface and included them separately. The data dictionary refers to all three subdiagrams.

1



4

3

2

Figure Data Binding - Case Details

|  |  |  |
| --- | --- | --- |
| **Data Dictionary** | | |
| **Num** | **Data** | **Description** |
| 1 | caseType | Current design shows caseType will be stored as part of a Case object. Type is CaseType, making use of enumeration as we know all possible values of this field. |
| 2 | title | Current design shows case title will be stored as part of a Case object. Type is String as alphabetical data expected. |
| 3 | description | Current design shows case description will be stored as part of a Case object. Type is String as alphabetical data expected. |
| 4 | CaseType | The possible values of a caseType. |

### Data Mapping – Add New Case (Client Details)

3

5

6

8

7

4

2

1

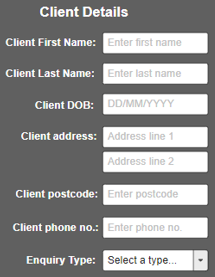
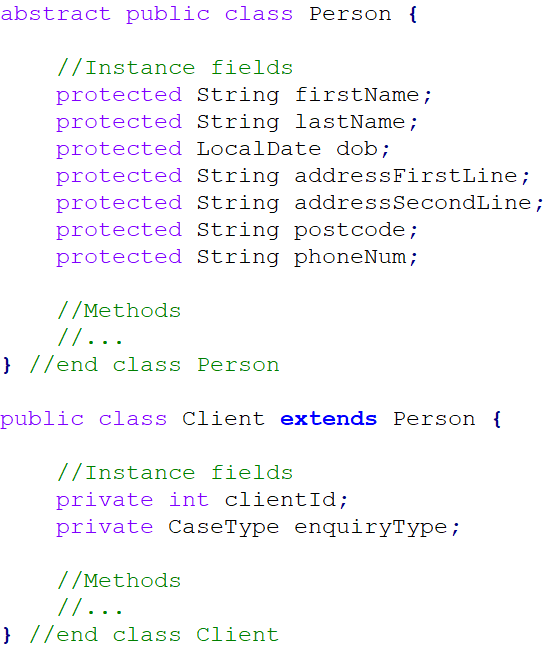
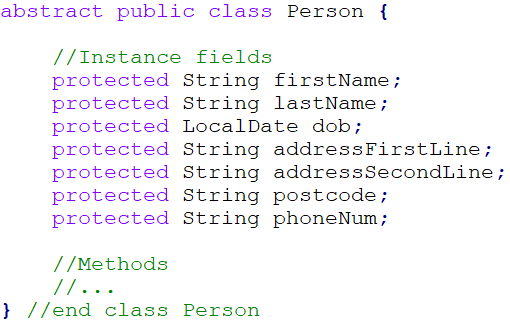


Figure Data Binding - Client Details

|  |  |  |
| --- | --- | --- |
| **Data Dictionary** | | |
| **Num** | **Data** | **Description** |
| 1 | firstName | Stored in firstName instance field of superclass Person as type String. |
| 2 | lastName | Stored in lastName instance field of superclass Person as type String. |
| 3 | dob | Stored in dob instance field of superclass Person as type LocalDate. |
| 4 | addressFirstLine | Stored in addressFirstLine instance field of superclass Person as type String. |
| 5 | addressSecondLine | Stored in addressSecondLine instance field of superclass Person as type String. |
| 6 | postcode | Stored in postcode instance field of superclass Person as type String. |
| 7 | phoneNum | Stored in phoneNum instance field of superclass Person as type String. |
| 8 | enquiryType | Stored in enquiryType instance field of subclass Client as type String. See row 4 in Data Dictionary table for Figure 55. |

### Data Mapping – Add New Case (Solicitor Details)

1



2

Figure Data Binding - Solicitor Details

|  |  |  |
| --- | --- | --- |
| **Data Dictionary** | | |
| **Num** | **Data** | **Description** |
| 1 | firstName | Dropdown list will show the Solicitor’s firstName (String) + lastName (String). |
| 2 | lastName | Dropdown list will show the Solicitor’s firstName (String) + lastName (String). |

### Data Mapping – Add New Case

The figure below shows what will ultimately complete a Case object, omitting the arrow numbers to avoid repetition:

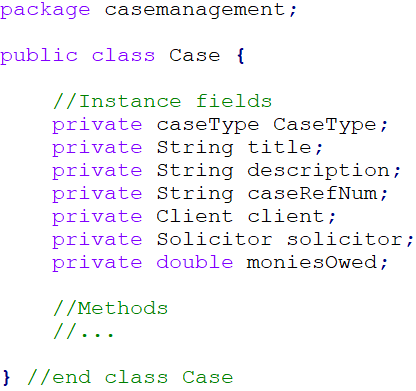
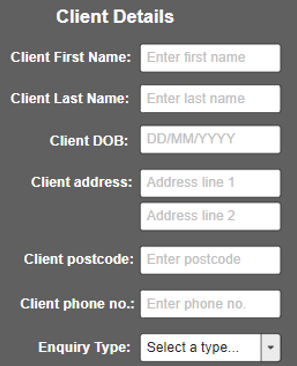


Figure Data Binding - Case Details

### Data Mapping – Recording Billable Activities

3

1

2

6

5

4

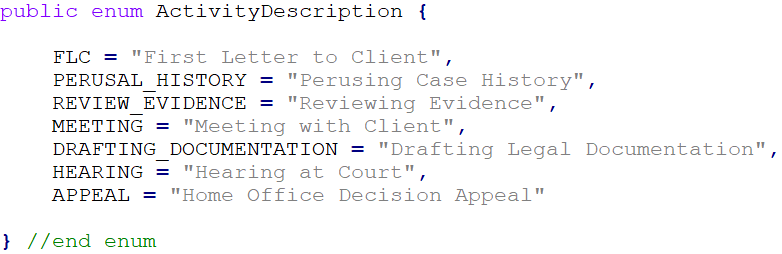
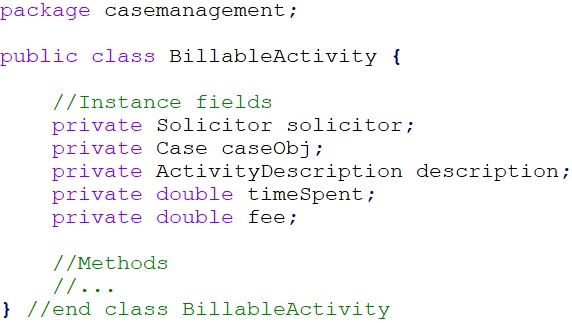
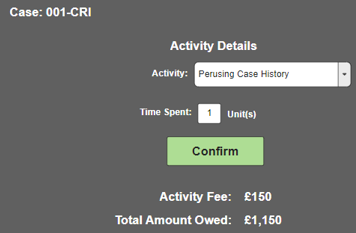
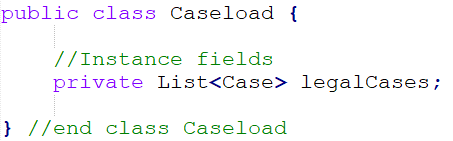
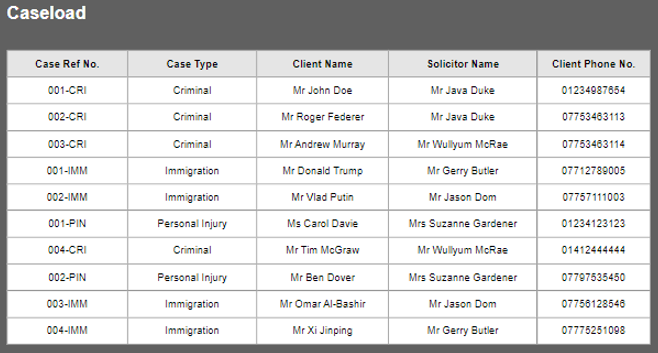


Figure Data Binding - Record Billable Activity

|  |  |  |
| --- | --- | --- |
| **Data Dictionary** | | |
| **Num** | **Data** | **Description** |
| 1 | caseRefNum | Accessed via the Case object, needed to link the moniesOwed to the given instance of a Case object. |
| 2 | activityDescription | Stored in BillableActivity as type ActivityDescription. See row 5. |
| 3 | timeSpent | Will be received in units (type: double) and converted to minutes for calculation of activity fee. |
| 4 | fee | The fee for this activity (type: double). Will use solicitor’s hourly rate to compute this value, hourlyRate accessed via solicitor attribute. |
| 5 | ActivityDescription | We know all possible values of a BillableActivity, so system will make use of enumeration here. |
| 6 | moniesOwed | Each Case object will contain a running total of how much is owed to the firm. Every time a fee is calculated, the fee will be added to the running total. |

### Data Mapping – View Caseload



1

Figure Data Binding - View Caseload

|  |  |  |
| --- | --- | --- |
| **Data Dictionary** | | |
| **Num** | **Data** | **Description** |
| 1 | legalCases | A List of all Case objects. Current design shows this as a List, but data structure may change during implementation. |

### Data Mapping – Add New Employee

If the firm hires a new employee, their details will be entered into the system in order for a username to be generated. If the jobTitle field is Secretary, as shown below, there will be no need for a specialism and hourlyRate value to be supplied. I have not included a data dictionary for this data map because it is very similar to the data dictionaries for each of the Add New Case figures, particular in Client Details and Solicitor Details given both Client and Employee extend the same class.

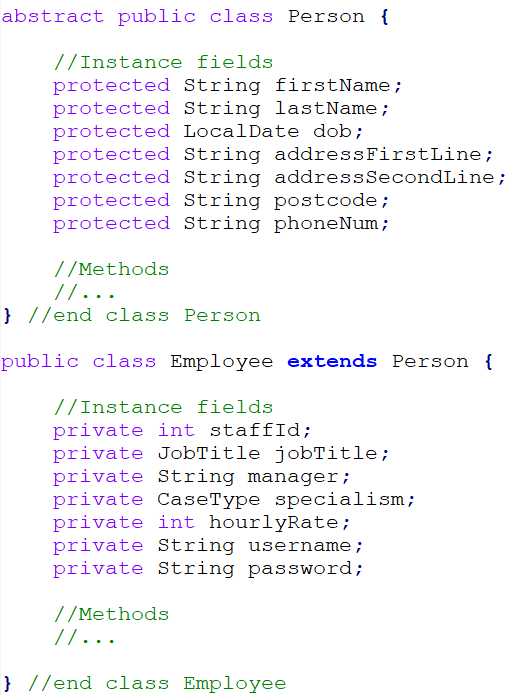
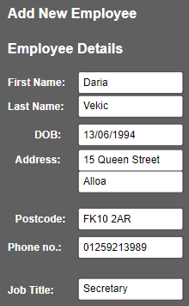


Figure Data Binding - Add New Employee

# Amendments History

A few minor amendments were made to the view model when developing the user interface. It highlighted that some labels had been overlooked and missed out in the wireframes. Details are provided below.

|  |  |  |
| --- | --- | --- |
| **Date** | **Amendment** | **Comments** |
| 05/03/2023 | Added in three components to user interface for Recording Billable Activities.  Use case description also updated to include this detail. | The Record Billable Activity wireframe only showed that the figure for the total amount owed for the case would be displayed. However, it would make more sense to also display the figure owed for the activity the user has just entered, and then display the total amount owed. I have also added in a label to this screen to make it clear what case the activity is being added for.  The agenda for meeting number 4 included the sample user interface that reflects this addition, but it did not seem worthwhile to update the wireframe because the user interface is more meaningful to the client. |
| 05/03/2023 | Added in component to user interface of Log In screen to allow for a new employee to set a password. | Client meeting 3 – discussion around how a new employee would be able to generate a username. Client meeting 1 – specifies the system should only generate the new employee a username, suggesting the user should be able to set the password themselves. They will now be able to do this through the link in the Log In interface. Presented to client in meeting number 4. |

# Bibliography

Babich, N. (2020). *Information Architecture Guide for UX Architects & Designers | Adobe XD Ideas*. [online] Ideas. Available at: <https://xd.adobe.com/ideas/process/information-architecture/information-ux-architect/>.

Goodrich, M.T., Tamassia, R. and Goldwasser, M.H. (2014). *Data Structures and Algorithms in Java*. Wiley Global Education.

Goyvaerts, J. and Levithan, S. (2012). *Regular expressions cookbook*. Beijing; Cambridge; Farnham: O’Reilly.

Larman, C. (2005). *Applying UML and patterns : an introduction to object-oriented analysis and design and iterative development*. Upper Saddle River, N.J.: Prentice Hall Ptr.

Mackenzie, S. (2013). *Human-computer interaction: an empirical research perspective*. Amsterdam Etc.: Morgan Kaufmann.

Mclaughlin, B., Pollice, G. and West, D. (2007). *Head first object-oriented analysis and design*. Beijing: O’Reilly.

mcleblanc (n.d.). *association end multiplicity - ADO.NET*. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/association-end-multiplicity [Accessed 2 Mar. 2023].

Miles, R. and Hamilton, K. (2006). *Learning UML 2.0*. Sebastopol, Ca: O’Reilly.

Sidnei, R. (2014). *Object-Oriented Analysis and Design for Information Systems : Modeling with UML, OCL, and IFML*. Burlington: Elsevier Science.

www.ibm.com. (n.d.). *Messages in UML diagrams*. [online] Available at: https://www.ibm.com/docs/en/rsm/7.5.0?topic=diagrams-messages-in-uml [Accessed 12 Mar. 2023].

www.thebusinessresearchcompany.com. (n.d.). *Personal Computers Market Size 2022 And Growth Analysis*. [online] Available at: https://www.thebusinessresearchcompany.com/report/personal-computers-global-market-report.

1. As mentioned these have been omitted from the agenda to avoid cluttering this document, but the bullet points list the visual aids that were presented. [↑](#footnote-ref-1)
2. I have not included a conceptual class for Solicitor details. This is because a Solicitor is an employee, and as such their details are already stored in the system. [↑](#footnote-ref-2)
3. This data structure is subject to change by the development team during implementation. [↑](#footnote-ref-3)
4. These have been omitted from the agenda to avoid cluttering this document, but the bullet points list the visual aids that were presented. [↑](#footnote-ref-4)