Software Architectural Design Approval:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Issued by:  DEV |  | < Author > |  |  |  | < Date > |  |
|  | Name printed |  | Signature |  | Date |  |
| Revision No.: |  | | | | | | |
| Approved by  PL: |  | < Name > |  |  |  |  |  |
|  | Name printed |  | Signature |  | Date |  |
| Approved by  TEST: |  | < Name > |  |  |  |  |  |
|  | Name printed |  | Signature |  | Date |  |
| Approved by  QA: |  | < Name > |  |  |  |  |  |
|  | Name printed |  | Signature |  | Date |  |
| Approved by  IT: |  | < Name > |  |  |  |  |  |
|  | Name printed |  | Signature |  | Date |  |
| Approved by OE |  | < Name > |  |  |  |  |  |
|  | Name printed |  | Signature |  | Date |  |

< Project Number, Project Name >

Knee Balancer application

< Version Number >

< Document ID >

BLUE TEXT is provided for explanation, description or examples for each section, as applicable. All BLUE TEXT is to be deleted prior to the approval of the document.

Table of Contents

1 Introduction 6

1.1 Purpose 6

1.2 Scope 6

1.3 Business Context 6

1.4 Definitions, Acronyms, and Abbreviations 6

1.5 References 7

1.6 System Context 7

2 System Decomposition 8

2.1 Hardware Decomposition Not applicable 8

2.1.1 Overall System 8

2.1.2 Sub System 1 8

2.1.3 Sub System 2 8

2.2 Software Decomposition 8

2.2.1 Overall Software System 8

2.2.2 Knee Balancer App-iOS 11

2.2.3 Knee Balancer Java Backend - Cloud 12

2.2.4 Layer 1 12

2.2.5 Layer2 12

2.3 Support for Manual Operations of the System 12

2.3.1 Human-equipment Interface 13

2.3.2 On-line Help Menus 13

2.3.3 Speech Recognition 13

2.3.4 Voice Control 13

3 Risk Control 13

4 Deployment View 13

5 Runtime View 13

5.1 Runtime View 1 13

5.2 Runtime View 2 14

6 Architectural Key Aspects 14

6.1 Safety 14

6.2 Accuracy 14

6.3 Extensibility 14

6.4 Configurability 14

6.5 Maintainability 14

6.6 Testability 14

6.7 Security 14

6.8 Performance 14

6.9 Scalability 14

6.10 Reliability 14

6.11 Workflow Control 14

6.12 Error Handling and Recovery 14

6.13 Logging and Tracing 15

6.14 Parallelization and Threading 15

6.15 Internationalization 15

The application shall support only English language. 15

6.16 Localization 15

The application shall support English language only 15

6.17 Communication between Distributed Components 15

6.18 Migration 15

7 Design Decisions 15

7.1 Design Decision 1 15

MVVM (Model-View-ViewModel) 15

7.2 Design Decision 2 15

7.3 Design Decision 3 17

8 List of Software of Unknown Provenance (SOUP) 17

9 Development Environment 17

9.1 Standards 17

9.2 Methods 18

9.3 Object Oriented Design and Analysis with UML 18

9.3.1 Swift and Java programming 18

9.3.2 Continuous Integration/CD 18

9.3.3 Code Reviews 18

9.3.4 Unit Tests 18

9.4 Tools 18

9.4.1 Integrated Development Environment: XCode 12.5.x, Eclipse, Spring tool-suite 18

9.4.2 UML Modelling Tool: Visio 18

9.4.3 Configuration Management System: Not applicable 18

9.4.4 Continuous Integration System: Azure 18

9.4.5 Refactoring Tool: Not applicable 18

9.5 Supporting Items 18

10 Document Revision History: 18

# Introduction

## Purpose

This document describes the overall architectural design of the Knee Balancer Application system. It depicts from a high-level perspective the system’s context and its static as well as dynamic structure. Where details go beyond the scope of this document the reader is referred to lower level architecture and design documents. The intended audience for this document is development, regulatory affairs and quality assurance.

## Scope

Develop the Knee Balancer app which is a clinical decision support software tool which provides on demand, an automated intra operative plan to the Orthopaedic surgeon based on pre-operative data, captured soft tissue information and surgeon preferences.

* The Knee Balancer app will be designed for iPhone and iPad
* The app provided to Stryker via Stryker App Store
* Product security

App includes

* Case input
* Case solve
* Surgeon/User preferences
* Case identifier

Out of scope

* Surgical navigation system screen recognition
* Digital connection to the other system (Bluetooth)
* Medical image data intake
* Download of case information and send email
* Record position page

## Business Context

Develop the knee Balancer iPhone/ iPad application is intended to improve the efficiency that is involved with calculating the implant movements required during TKA intra-operative balancing. The Knee Balancer app which is a clinical decision support software tool which provides on demand, an automated intra operative plan to the Orthopaedic surgeon based on pre-operative data, captured soft tissue information and surgeon preferences.

## Definitions, Acronyms, and Abbreviations

Following definitions, acronyms, and abbreviations are used throughout this document:

|  |  |
| --- | --- |
| Terms | Definition |
| MPS | Mako Product Specialist |
| MAKO | Robotic-Arm Assisted Surgery |
| MA | Mechanical Alignment |
| FA | Functional Alignment |
| IA | Individualized Alignment |
| HKA | Hip Knee Ankle Alignment |
| STR | Soft Tissue Release |

## References

|  |  |  |
| --- | --- | --- |
| ID | Title | Doc. No. |
| SDP | SGTC-QFM-DLC-001-01\_SOFTWARE\_DEVELOPMENT\_PLAN\_rev\_05-draft |  |
| URS | KB SGTC-QFM-DLC-001-22\_USER\_REQUIREMENTS\_SPECIFICATION\_rev\_04 | SGTC-QFM-DLC-001-22 |
| SRS | SGTC-QFM-DLC-001-02\_SOFTWARE\_REQUIREMENTS\_SPECIFICATION\_rev\_07 | SGTC-QFM-DLC-001-02 |

-

Risk Analysis - < Risk Analysis >

## System Context

Knee Balancer application collects the pre-operative data or the initial plan along with the Surgeon preference from the MPS user manually referred from the Mako system. The application also collects the intra-operative data during the surgery and generates the list of solutions which will help the Surgeon to select the appropriate solution which will be the final plan (or final solution) for the surgery.



The main users of the system are

|  |  |
| --- | --- |
| Users | Description |
| MPS | User who refers the Mako system and inputs the data to knee balancer application.  Follows the commands as instructed by the Surgeon.  Enters the patient information. |
| Surgeon | User who creates Surgeon/ user preference cards. |
| Service Engineer | Stryker user uses it to pull data for analysis |

The other system which is used for the Knee Balancer application

|  |  |  |
| --- | --- | --- |
| ID | Neighbouring system | Description |
| NA | Mako system | Provides the plan and the input data to the MPS user. |
|  |  |  |

# System Decomposition

## Hardware Decomposition Not applicable

### Overall System

Not applicable

### Sub System 1

Not applicable

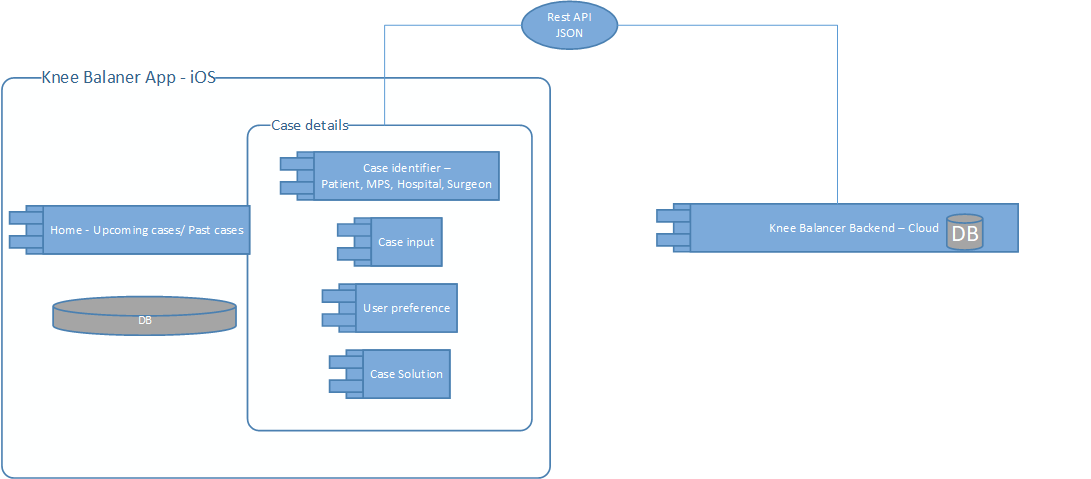
### Sub System 2

Not applicable

## Software Decomposition

### Overall Software System

The software system is client server architecture which consists of two components. Frontend Knee Balancer app - iOS application and backend Knee Balancer - Cloud. Data communication is through Rest APIs using JSON.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SAD ID | Units | Description | SRS ID | Safety Class |
| SAD100 | Home | Upcoming case will show added case details of patients. Past cases will show the solution details and the summary of the case | SRS-1.0.0 |  |
| SAD101 | Case input | Case input screen allows the user to input femoral pre-operative plan data including medial distal femoral, lateral distal femoral, medial posterior femoral, lateral femoral posterior resection depths, femoral component varus/valgus and internal/ external rotation, and tibial pre-operative plan data including medial proximal and lateral proximal tibial resections and tibial component varus/valgus.  Allows to add intra operative data like medial extension gap, lateral extension gap, medial flexion gap and lateral flexion gap Or User can input through camera capture All values except mHKA parameters which needs to be entered manually. | SRS-1.1.0 |  |
| SAD102 | Case solution | The automated plan will be generated considering the following inputs: pre-operative data, Surgeon Preferences (Ranges and Targets), Intra-operative data (HKA, FFD, initial gaps), within the minimum and maximum gap ranges set in the surgeon preference card.  Adjust solution parameters itself manually either by key entries or camera capture | SRS-1.2.0 |  |
| SAD103 | Surgeon/ User preference | Allows user to input the following planning ranges for the femoral component to be used during generation of automated implant plan:  coronal alignment (Varus/Valgus),transverse alignment (I/E Rotation),medial lateral distal and posterior resections.  Allows the user to input planning targets and ranges for the tibial component to be used during generation of automated implant plan:  Coronal alignment (Varus/Valgus)  medial lateral tibial resections.  Allows user to input Planning targets and ranges for the minimum and maximum final gaps (range) and ideal final gaps (target). | SRS-1.3.0 |  |
| SAD104 | Case identifier | Application allows to add the following patient details Patient first name, Patient last name, Patient Id, Date of birth, sex, Hospital, Surgeon, Surgery ate, Surgery time and Notes. | SRS-1.4.0 |  |
| SAD105 | Knee Balancer app – iOS database | To enter data required for the knee balancer referring Mako system and generate appropriate solutions with the help of algorithms for the surgery. Store the case details in the iOS database until the data is uploaded to cloud | SRS-1.0.8 |  |
| SAD106 | Knee Balancer Backend-Cloud | Dump the case data and the log files (errors, exceptions etc) from the iOS app over internet on specific interval and for further analysis and retrieval | SRS-1.0.9 |  |
| SAD107 | Knee Balancer Authentication | Enables developers to acquire tokens from the Microsoft identity platform in order to authenticate users and access secured web APIs. | SRS-6.5.0 |  |

The block diagram of the architecture is shown

Diagram

Description automatically generated

### Knee Balancer App-iOS

The frontend Knee Balancer application diagram is as shown



|  |  |
| --- | --- |
| Units | Description |
| App View/UI | Module is responsible for displaying the view to the end users and enables them to perform user interface events. The UI layer is created using storyboard provided by apple. And each of the UI element is associated with UI controller. |
| ViewModel | This module is responsible for handling all business logics, converting data in model into human readable format that can be presented in the view-by ViewController.  Updates to View data would not go directly to the Model, rather they would  be triggered by the view controller talking to the View Model which would then talk to the model. |
| HTTP communication manager | This module is responsible for interacting with web server for sending /receiving (JSON) data using the REST API and communicates back to view model, data repository |
| DataRepositories | This module is acts as a wrapper for all database services, so that your application can work with a simple abstraction that has an interface. |
| DatabaseServices | This module is responsible for handling the data base operation implementation and sending back the result into ViewModel again. |
| CoreData | Core Data is a framework that use to manage the model layer objects in application. It provides generalized and automated solutions to common tasks associated with object life cycle and object graph management, including persistence. |
| Validator | This module is responsible for validating the input provided and sending back the validated output into the view model. |
| Utility | Used to assist in providing some reusable functionality (Colour literals, custom fonts etc..) |
| Database | Local database within the iOS for the local storage to store data temporarily when executed offline until the data is sent to cloud database |
| Algorithm | This module contains all of the logic for solving an initial position within the set limits and boundaries. Takes an InitialPosition object, preference object stores results as an array of Solution objects. |

### Knee Balancer Backend - Cloud

The Knee Balancer Backend – Cloud diagram is as shown

Diagram

Description automatically generated

|  |  |
| --- | --- |
| Units | Description |
| Azure Storage Account | Azure storage account contains all of your Azure Storage data objects i.e files.The storage account provides a unique namespace for your Azure Storage data that is accessible from anywhere in the world over HTTP or HTTPS. |
| Azure File Service | To store the data as json file in Azure cloud-azure blob storage for future retrieval and analysis |

## 

## Support for Manual Operations of the System

Not applicable

### Human-equipment Interface

Not applicable

### On-line Help Menus

Not applicable

### Speech Recognition

Not applicable

### Voice Control

Not applicable

# Risk Control

This section identifies components which are segregated to ensure effective risk control. It is required for safety class C components (Optional for Class A & B) and software systems (IEC 62304). For example, one designated component may comprise contributions to risk control measures.

<provide a reference to the risk control either an identified hazard or risk control measure which links to the risk analysis. An example of segregation is to have SOFTWARE ITEMS execute on different processors. The effectiveness of the segregation can be ensured by having no shared resources between the processors. >

|  |  |  |
| --- | --- | --- |
| Units | Reference to Risk control | Statement how to ensure that the segregation is effective |
| Software Unit 1 |  |  |
| Software Unit 2 |  |  |

# Deployment View

<This section describes, the hardware environment in which the system is supposed to run. It describes the geographical distribution and the structure of the individual hardware units and defines which software units are hosted on those hardware units. The deployment view describes the system from an operator’s/administrator’s point of view. Typical information you should put in here include computers, processors, network topologies, other units of the physical system’s environment.>

# Runtime View

< This section describes, how the system’s units collaborate in terms of processes, tasks, activities, threads, …. Focus on the most interesting runtime aspects, such as:

* How are the most important use cases realized by the system’s units?
* How do the system’s units collaborate with the external units?
* How does the system start up?

Use collaboration, sequence diagrams, flow charts to highlight the runtime aspects… >

## Runtime View 1

< Runtime diagram and explanation of the particularities of the collaboration of the units involved >

## Runtime View 2

< Runtime diagram and explanation of the particularities of the collaboration of the units involved >

# Architectural Key Aspects

## Safety

…

## Accuracy

…

## Extensibility

MVVM makes any of components more reusable. It has also the ability to replace or add new pieces of code that do similar things to the right places in the architecture.

## Configurability

…

## Maintainability

…

## Testability

MVVM makes each components more loosely coupled so makes testing easier.

## Security

App uses Microsoft authentication for login, Camera for Vision Kit, Core Data to store the value in encrypted mode. App will be released over Intune store.

## Performance

…

## Scalability

NA

## Reliability

## Workflow Control

NA

## Error Handling and Recovery

Two things considered under Error handling - Exception and Error

Using **NSSetUncaughtExceptionHandler** inline function we can log the exceptions.

**Handling signals**

At some point of the code, we must implement a way to handle the signals in the stack trace to catch the crash. I've found out that there are two ways of doing that:

* **UNIX Signals:**Since there are more developers that are more familiar to Unix Signals, you will find more stuff googling about this. I suggest starting from this one, but be aware that there are some events of crash in swift that are not directly translated to UNIX Signals.
* **Mach Exception Handler:**This is the default error handling mechanism used by Apple crash reporter. You might even think that using this is a good idea, but this mechanism is quite complex to understand.

## Logging and Tracing

Mix panel is used for Logging and Tracing, allows to analyse how users interact with Internet- connected product, allowing everyone to analyse user data in real-time to identify trends, understand user behaviour, and make decisions about your product.

Graphical user interface, application

Description automatically generated

## Parallelization and Threading

NA

## Internationalization

### The application shall support only English language.

## Localization

### The application shall support English language only

## Communication between Distributed Components

NA

## Migration

…

# Design Decisions

The Knee balancer iOS app using MVVM design pattern. The MVVM pattern introduces a fourth component, the **view model**. The view model is responsible for managing the model and sending the model's data to the view via the controller.

## Design Decision 1

The decision was to choose MVVM over MVC for the frontend architecture due to following

|  |  |
| --- | --- |
| MVVM (Model-View-ViewModel) | **MVC** (Model-View-Controller) |
| Makes the view controller simpler by moving a lot of business logic out of it. | All business logic and UI related logics will be in ViewController class. |
| The view model better expresses the business logic for the view. | There is no ViewModel. |
| Breaks the coupling between the application logic and the UI and so make testing more accessible. | All business logic and UI related logics will be in ViewController class makes testing difficult. |
| The responsibilities of the view controller are reduced to controlling the interaction between the view layer and the model layer. | ViewController will be responsible for communicating with View and Model. |

## Design Decision 2

The decision was to choose Vision Kit over Opensource OCR for the frontend architecture due to following.

One of Vision’s many powerful features is its ability to detect and recognize multilanguage text in images. You can use this functionality in your own apps to handle both real-time and offline use cases. In all cases, all of Vision’s processing happens on the user’s device to enhance performance and user privacy.

Vision’s text-recognition capabilities operate using one of these paths:

**Fast:**

The fast path uses the framework’s character-detection capabilities to find individual characters, and then uses a small machine learning model to recognize individual characters and words. This approach is similar to traditional optical character recognition (OCR).

**Accurate:**

The accurate path uses a neural network to find text in terms of strings and lines, and then performs further analysis to find individual words and sentences. This approach is much more in line with how humans read text.

Graphical user interface, application, table

Description automatically generated

## 7.3 Design Decision 3

The decision was to choose Microsoft oAuth over other oAuth for the frontend architecture due to following.

Table

Description automatically generated

# List of Software of Unknown Provenance (SOUP)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Title | Manufacturer | Version | Release Date | License Type | Maintenance procedure |
| SOUP 1 | Mix panel | 3.1.5 | 19-02-2022 | Open source | Cocoa pods |
| SOUP 2 | Vision Kit(iOS) | iOS 14.5 | 07-03-2020 | Available from iOS 13 | XCode |
| SOUP 3 | Microsoft Authentication | 1.1.3 | 21-01-2022 | Open Source | Cocoa pods |

# Development Environment

## Standards

< Mandatory for software safety class C. Optional for class A and B. Include or reference development standards which are used in addition to those described in our Quality Management System. >

## Methods

< Mandatory for software safety class C. Optional for class A and B. List design methodologies and notations, programming language, system integration procedure, and so on. List quality assurance practices including methods of technical peer review, unit testing, stepping through code in a debugger, system testing, automated regression tests, and so on. The following list provides standard methods, modify or extend the list as appropriate >

## Object Oriented Design and Analysis with UML

### Swift and Java programming

### Continuous Integration/CD

### Code Reviews

### Unit Tests

Unit testing is a level of software testing where individual units/ components of software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It is performed by white box testing method. This is the first level of testing performs before integration testing which is performed mainly by the developers. All unit tests in iOS use Apple’s XCTest framework. Every test case you write will import it.

## Tools

The following list provides standard tools used for development of the software described in this document.

### Integrated Development Environment: XCode 12.5.x, Eclipse, Spring tool-suite

### UML Modelling Tool: Visio

### Configuration Management System: Not applicable

### Continuous Integration System: Azure

### Refactoring Tool: Not applicable

## Supporting Items

Not applicable

# Document Revision History:

|  |  |  |  |
| --- | --- | --- | --- |
| Revision  Level | Revision Date | Effective Date | Reason and Description of Revision |
|  |  |  |  |
|  |  |  |  |