# TESTING & INTEGRATION PLAN

## DEPARTMENT OF ELECTRONIC ENGINEERING

## UNIVERSITY OF YORK

**MENG YEAR 3**

**SOFTWARE ENGINEERING GROUP PROJECT**

PENELOPE

MAR 2023

## 1. Purpose

The purpose of this document is to showcase the company's testing and integration process, making sure that the product is properly checked and validated, as well as instructing and giving guidance to Penelope's personnel whose actions affect the overall testing phase. This plan also acts as a reference to customers, inspectors or auditors who are interested in what measures and controls are implemented to assure the product is properly tested.

## 2. Scope

Our testing and integration process will comprise a detailed analysis and validation of the key components which make up the product. This includes but is not limited to the main aspects of the product, such as the database server, Android app and administrator desktop application. Following strict testing protocols, we will assess the functionality and performance of each component and its associated functions.

### 2.1. System Overview

Database server. 'Penelope' is a Java SpringBoot server which relies on an underlying PostgreSQL database instance:

**Server controllers**

- API Key controller
- Authentication utility controller
- Bird controller
- Campus controller
- File download/upload controller

**XML elements**

- Bird element
- Campus element
- Campus List element
- Common XML element

**Services**

- File system storage service

**Endpoints**

- Campus endpoints
- Bird endpoints
- Authentication and Identity endpoints

'FaunaFinder' Android application:

**Main application elements**

- Audio element
- Circle element
- Line element
- Rectangle element
- Video element
- Text element
- Image element

**Main application parsers**

- Audio parser
- Circle parser
- Line parser
- Rectangle parser
- Video parser
- Text parser
- Image parser
- Presentation parser

**Various presentation modules**

- Canvas view
- Slides recycler view adapter
- Slide view holder
- List item click action and listener
- UI utilities

‘Icarius’ desktop application, which is designed to be our (Sys) Admin tool which will allow control over the database:

**Graphical User Interface**

- GUI Controller
- Main Tab
- Login Tab
- Campus Tab
- Bird Tab
- Users Tab

**Database Entities**

- Campus
- Bird

**Communication Module**

- GET Request
- POST Request
- DELETE Request
- PATCH Request
- KEY Request

**Authorisation Controller**

- Authentication Service
- User Entity

### 2.2. Out-of-scope Features

The following features will not be tested by our in-house team due to contract obligations with the other team.

**FaunaFinder:**

1. Text element
2. Image element

## 3. Test Plan & Strategy

At the highest level, the key aspects of our testing strategy can be summarised as follows:

- Develop automated and manual tests to cover all the quality risks recognised as needing extensive or balanced testing, focusing on behavioural factors observable at some user interface.

- Add test data or conditions within existing cases or new test cases to cover critical customer usage profiles, customer data, or known defects in our product.
- Use exploratory testing in areas that were not addressed previously and that appear, due to test results, to be at high risk of bugs. Update or define new test cases to cover found bugs.
- Run tests across a "customer-like" environment or similar configurations.
- Repeat all integration and other functionality tests multiple times across each phase to detect regression.
- Possibly, if time and resources allow doing so, use structural coverage techniques to identify untested areas, after which add tests to cover critical test gaps.

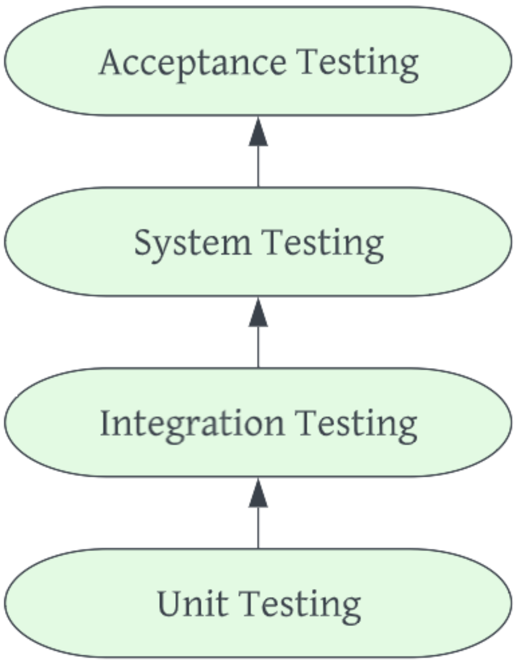

Figure 1: Overall testing topology

## 3.1. Testing tools used

Testing tools used during testing include:

- JUnit: A software testing framework used for unit testing
- Mockito: A mocking framework used alongside JUnit to help write unit tests when mocking is required
- Spring Boot: A framework that allows for unit and integration testing, as well as mocking
- Espresso: An Android testing framework utilised for writing automated End-to-end tests, including GUI and performance tests
- Robolecrtric: An Android testing framework to run Android unit tests on the JVM rather than on an Android device.

## 3.2. Test Types

### 3.2.1. Unit Tests

Unit testing involves testing a small, isolated part of the codebase that does not interact with any external systems, such as databases. When writing unit tests, the JUnit framework is used, specifying the expected input and asserting the expected outputs of the method, whilst Mockito allows for the inputs to be mocked if needed. Once the method is run, the outputs can be verified and compared to the expected outputs.

### 3.2.2. Integration Test (Instrumentation Tests)

Integration testing focuses on maintaining correct functionality of modules that interact with each other. These modules are tested in groups and will follow a bottom-up integration approach, starting with the lowest level modules having to pass integration tests before moving on to the higher level modules. Integration tests are executed utilising Spring Boot test cases, which can carry out methods within the modules under test, and assert the expected outcomes.

### 3.2.3. Automated End-to-end testing (Espresso)

End-to-end testing is used to test the whole application's performance and functionality from start to finish, verifying that everything works as expected. Automated End-to-end testing for FaunaFinder will be carried out utilising the Espresso testing framework, which will test the functionality of the GUI by synchronising written test actions with the user interface. For each view in the GUI under test, an action is performed and the state of the view is asserted, which can then be verified.

### 3.2.4. User End-to-end testing

User End-to-end testing focuses on the user experience and usability of the entire system, ensuring the functionality of the product meets user expectations. This involves having multiple users run through a specified list of actions throughout each view in the application, while the functionality of the application after each action is verified by the tester.

## 3.3. Test cases

## 3.4. Penelope

| Test Id | Entity | Test Description | Test Data | Expected Result |
|---|---|---|---|---|
| **PT01** | **Api Key Controller** | **Create new identity** | Mock loading and storing the key,<br>Authentication key | New user identity created in database;<br>200 HTTP status code |
| **PT02** | **Api Key Controller** | **Remove existing identity** | Mock loading and deleting the key,<br>Non-admin identity,<br>Authentication key | User identity removed from database;<br>200 HTTP status code |
| **PT03** | **Api Key Controller** | **Add Campus permissions to existing Api key** | Mock loading key,<br>Non-admin identity,<br>Campus object,<br>Authentication key | Permissions added to campus;<br>200 HTTP status code |
| **PT04** | **Api Key Controller** | **Remove Campus from existing Api key** | Mock loading key,<br>Non-admin identity,<br>Campus object,<br>Authentication key | Campus is deleted from database;<br>200 HTTP status code |
| **PT05** | **Bird Controller** | **Cannot create bird in a non-existent campus** | False authentication key,<br>Mock loading key,<br>Bird parameters | Bird not created;<br>404 HTTP status code |

| | | | | |
|---|---|---|---|---|
| **PT06** | **Bird Controller** | **Cannot create bird if parameters invalid** | Authentication key,<br>Mock loading key,<br>Invalid bird parameters | Bird not created;<br>4xx HTTP status code |
| **PT07** | **Bird Controller** | **Cannot create bird if no access to campus** | Authentication key,<br>Mock loading key,<br>Fake user identity,<br>Bird parameters | Bird not created;<br>403 HTTP status code |
| **PT08** | **Bird Controller** | **Create bird** | Authentication key,<br>Mock loading key,<br>Bird parameters | Bird created;<br>200 HTTP status code |
| **PT09** | **Bird Controller** | **Delete bird** | Authentication key,<br>Mock loading and delete key,<br>Fake campus object,<br>Fake user identity,<br>Bird ID | Bird deleted;<br>200 HTTP status code |
| **PT10** | **Bird Controller** | **Edit bird contents** | Authentication key,<br>Mock loading key,<br>Fake campus object,<br>Fake bird object,<br>Bird parameters | Bird details changed;<br>200 HTTP status code |
| **PT11** | **Campus Controller** | **Cannot create campus with bad credentials** | Authentication key,<br>Mock loading key,<br>Fake user identity | Campus not created;<br>403 HTTP status code |
| **PT12** | **Campus Controller** | **Cannot create campus with invalid parameters** | Authentication key,<br>Mock loading key,<br>Invalid parameters | Campus not created |
| **PT13** | **Campus Controller** | **Create campus** | Authentication key,<br>Mock loading key,<br>Campus parameters | Campus created;<br>200 HTTP status code |
| **PT14** | **Campus Controller** | **Cannot delete non-existent campus** | Authentication key,<br>Mock loading and delete key,<br>Fake campus object,<br>Incorrect campus ID | Campus not deleted;<br>404 HTTP status code |
| **PT15** | **Campus Controller** | **Delete campus** | Authentication key,<br>Mock loading and delete key,<br>Fake campus object,<br>Correct campus ID | Campus deleted;<br>200 HTTP status code |
| **PT16** | **Bird XML element** | **Bird “Hero slide” is correct** | Bird XML object,<br>Slide, image, audio elements | Slide, image, audio elements are not null;<br>Slide, image, image elements are assigned correctly |
| **PT17** | **Bird XML element** | **Bird “About me” section is correct** | Bird XML object,<br>Slide, URL, text elements | Slide, URL, text elements are not null;<br>Slide, URL, text elements are assigned correctly |

| | | | | |
|---|---|---|---|---|
| **PT18** | **Bird XML element** | **Bird “Diet” section is correct** | Bird XML object,<br>Slide, image, text elements | Slide, image, text elements are not null;<br>Slide, image, text elements are assigned correctly |
| **PT19** | **Bird XML element** | **Bird “Location” section is correct** | Bird XML object,<br>Slide, image, text elements | Slide, image, text elements are not null;<br>Slide, image, text elements are assigned correctly |
| **PT20** | **Common XML element** | **Can create the XML document** | Document object,<br>Info, title, author, date, number of slides elements | Info, title, author, date, number of slides elements not null;<br>Title, author, date, number of slides elements are assigned correctly |
| **PT21** | **Common XML element** | **Can get bytes of an XML document** | XML document<br>Root element | Generate a byte array of the XML document |
| **PT22** | **Common XML element** | **Can increment the amount of slides** | XML document | Increment the amount of slides in the XML |
| **PT23** | **Campus XML element** | **Can format the bird description to be of shorter length** | Bird description | Description is truncated to be less than 50 characters long |
| **PT24** | **Campus XML element** | **Can add a “Bird” object to the campus XML** | Campus XML object,<br>Slide, title, text, image elements | Slide, title, text, image elements are not null;<br>Slide, title, text, image elements are assigned correctly;<br>Only 2 instances of the “text” element, and they’re assigned in the right order |
| **PT25** | **Campus List XML element** | **Can add a “Campus” object to the list of campuses** | Campuses List XML object,<br>Slide, text, line elements | Slide, text, line elements are not null;<br>Slide, text, line elements are assigned correctly |
| **PT26** | **File System Storage Service** | **The *init* method throws an exception** | Base string,<br>Key base string | A Storage Exception is thrown;<br>Base string and Key base string are set to original values |
| **PT27** | **File System Storage Service** | **Check if file system can store image, video and audio files** | MockMultipartFile object,<br>Image, video, audio String elements,<br>Video, audio MIME type objects | Images, videos and audios are stored |

| | | | | |
|---|---|---|---|---|
| **PT28** | **File System Storage Service** | **Cannot store when original file name is empty** | MockMultipartFile with an empty byte array | The file is not stored |

| | | | | |
|---|---|---|---|---|
| **PT29** | **File System Storage Service** | **Can store an image with valid parameters** | BufferedImage object with specified dimensions,<br>Image filename | The image is stored |
| **PT30** | **File System Storage Service** | **Cannot store a bad processed image with invalid parameters** | BufferedImage object with specified dimensions,<br>Invalid filename | The image is not stored |
| **PT31** | **File System Storage Service** | **Can load the correct path with valid input** | Base string,<br>Class under test instance,<br>Image file name | The correct path is returned as a *Path* object |
| **PT32** | **File System Storage Service** | **Can load an existing resource with correct parameters** | Fake test file,<br>Test file path | Return *Resource* object with the correct file name |
| **PT33** | **File System Storage Service** | **Can load a *bird* resource from the database** | Campus entity,<br>Bird entity,<br>Bird entity ID | Load a resource for the bird entity with the given ID from the database |
| **PT34** | **File System Storage Service** | **Cannot load a missing *bird* resource** | Fake bird ID | Receive a *ResponseStatus* exception |
| **PT35** | **File System Storage Service** | **Can load a *campus* object containing *bird* objects** | Campus object,<br>Bird object,<br>Campus object ID | Return a non-null *Resource* object |
| **PT36** | **File System Storage Service** | **Can load a *campus* object without *bird* objects** | Campus object | Return a non-null *Resource* object |
| **PT37** | **File System Storage Service** | **Can load a list of campuses from the database** | Campus object | Return a non-null *Resource* object |
| **PT38** | **File System Storage Service** | **Can store a private key** | RSA key pair,<br>Identity string | The key is stored |
| **PT39** | **File System Storage Service** | **Cannot store a key if path location is missing** | RSA key pair,<br>Identity string | The key is not stored;<br>Throw an *IO* exception |
| **PT40** | **File System Storage Service** | **Can load RSA key from storage** | RSA key,<br>Identity string | Return a non-null byte array |
| **PT41** | **File System Storage Service** | **Can return an empty byte array if no key is stored** | Identity string | Return empty byte array of length zero |
| **PT42** | **File System Storage Service** | **Can remove any stored key** | RSA key | Return a *true* boolean |
| **PT43** | **File System Storage Service** | **Cannot remove a non-existent key** | Identity string | Return *false* boolean |

## 3.5. Icarius

| Test Id | Entity | Test Description | Test Data | Expected Result |
|---|---|---|---|---|
| **IT01** | **Bird** | **Create Bird** | Mock Admin User, POST Request, Bird name and parameters | Created bird ID,<br>POST Request contains:<br>- Penelope Endpoint URL<br>- Bird parameters in headers<br>- Authentication Header |
| **IT02** | **Bird** | **Read Bird** | GET Request,<br>Mock XML Response | Bird object contains parameters from Mock XML, Returns Bird Name, GET Request contains:<br>- Penelope Endpoint Url |
| **IT03** | **Bird** | **Update Bird** | Mock Admin User,<br>PATCH Request | PATCH Request contains:<br>- Penelope Endpoint URL<br>- Bird parameters in headers<br>- Authentication Header |
| **IT04** | **Bird** | **Delete Bird** | Mock Admin User,<br>DELETE Request | Boolean indicating success,<br>DELETE Request contains:<br>- Penelope Endpoint URL<br>- Authentication Header<br>- Bird Id header |
| **IT05** | **Campus** | **Create Campus** | Mock Admin User, POST Request, campus name | Created Campus ID,<br>POST Request contains:<br>- Penelope Endpoint URL<br>- Authentication Header<br>- Campus name in headers |
| **IT06** | **Campus** | **Read Campus** | GET Request,<br>Mock XML | List of Bird Ids, Campus name,<br>GET Request contains:<br>- Penelope Endpoint Url |
| **IT07** | **Campus** | **Update Campus** | Mock Admin User,<br>PATCH Request | PATCH Request contains:<br>- Penelope Endpoint URL<br>- Campus Name Header<br>- Authentication Header |
| **IT08** | **Campus** | **Delete Campus** | Mock Admin User,<br>DELETE Request | Boolean indicating success,<br>DELETE Request contains:<br>- Penelope Endpoint URL<br>- Authentication Header<br>- Campus Id header |
| **IT09** | **GET Request** | **Create GET Request** | Fake URL | GET Request contains:<br>- Full Server URL |
| **IT10** | **POST Request** | **Create POST Request** | Fake URL,<br>Mock User<br>Key Value Hashmap | POST Request contains:<br>- Full Server URL<br>- User<br>- Hashmap parameters |
| **IT11** | **DELETE Request** | **Create DELETE Request** | Fake URL,<br>Mock User<br>Key Value Hashmap | DELETE Request contains:<br>- Full Server URL<br>- User |

| | | | | |
|---|---|---|---|---|
| | | | | - Hashmap parameters |
| **IT12** | **PATCH Request** | **Create PATCH Request** | Fake URL,<br>Mock User,<br>Key Value Hashmap | PATCH Request contains:<br>- Full Server URL<br>- User<br>- Hashmap parameters |
| **IT13** | **Key Request** | **Get Server Public Key** | Mock XML | Server Public Key encoded in base 64 format,<br>Key Request contains:<br>- Penelope Key Endpoint URL |
| **IT14** | **Authentication Service** | **Create encrypted authentication key word** | Mock User | The keyword "username=password=current time" encrypted with the Public Key in base 64 format |
| **IT16** | **GUI** | **Log in** | Mock User,<br>Mock XML response | List of campus Ids if User valid, else Access Denied code |
| **IT17** | **GUI** | **View Campus List** | List of Campus Objects | List of campuses displayed |
| **IT18** | **GUI** | **CRUD Campus** | Campus Object,<br>User | Campus CRUD operations called and input values passed as parameters |
| **IT19** | **GUI** | **View Bird List** | Campus object containing List of Birds | List of Birds displayed under a Campus |
| **IT20** | **GUI** | **CRUD Bird** | Bird Object,<br>User | Bird CRUD operations called and input values passed as parameters |
| **IT21** | **GUI** | **Generate Key** | Admin User, New user credentials | Boolean confirming user creation |
| **IT22** | **GUI** | **Remove Key** | Admin User, user credentials | Boolean confirming user deletion |
| **IT23** | **GUI** | **Add campus to key** | Admin User | Boolean confirming campus addition to key |
| **IT24** | **GUI** | **Remove campus from key** | Admin User | Boolean confirming campus removal from key |

### 3.6. FaunaFinder

| Test Id | Entity | Test Description | Test Data | Expected Result |
|---|---|---|---|---|
| **FT01** | **Audio Element** | **Load audio media player onto canvas** | url, loop, x-coordinate, y-coordinate parent, container slide, id | Audio media player appears onto canvas with specified parameters |
| **FT02** | **Audio Element** | **Start audio media player** | url, loop, x-coordinate, y-coordinate, parent, container slide, id | Audio media player plays sound |

| | | | | |
|---|---|---|---|---|
| **FT03** | **Audio Element** | **Pause audio media player** | url, loop, x-coordinate, y-coordinate, parent, container slide, id | Audio media player stops playing sound |
| **FT04** | **Circle Element** | **Draw a circle onto canvas** | radius, colour, borderWidth, borderColour, x-coordinate, y-coordinate, canvas, slide | Draws a circle onto the canvas with specified parameters |
| **FT05** | **Line Element** | **Draw a line onto canvas** | thickness, fromX, fromY, toX, toY, colour, canvas, slide | A line is drawn onto the canvas from one specified coordinate to another, with specified parameters |
| **FT06** | **Rectangle Element** | **Draw a rectangle onto canvas** | width, height, colour, borderWidth, borderColour, x-coordinate, y-coordinate | A rectangle is drawn onto the canvas with specified parameters |
| **FT07** | **Video Element** | **Load video media player onto canvas** | url, width, height, x-coordinate, y-coordinate, loop, parent, container, slide, id | Video media player appears onto canvas with specified parameters |
| **FT08** | **Video Element** | **Start video media player** | url, width, height, x-coordinate, y-coordinate, loop, parent, container, slide, id | Video media player starts playing video |
| **FT09** | **Video Element** | **Stop video media player** | url, width, height, x-coordinate, y-coordinate, loop, parent, container, slide, id | Video media player stops playing video |
| **FT10** | **Audio Parser** | **Create a new Audio Element** | xmlParser containing Mock XML | Audio element containing url, x-coordinate, y-coordinate and loop boolean from XML |
| **FT11** | **Circle Parser** | **Create a new Circle Element** | xmlParser containing Mock XML | Circle element containing radius, colour, borderWidth, borderColour, x-coordinate and y-coordinate from XML |
| **FT12** | **Image Parser** | **Create a new Image Element** | xmlParser containing Mock XML | Image element containing url, width, height, x-coordinate, y-coordinate, rotation, delay and timeOnScreen from XML |
| **FT13** | **Line Parser** | **Create a new Line Element** | xmlParser containing Mock XML | Line element containing thickness, fromX, fromY, toX, toY and colour from XML |
| **FT14** | **Rectangle Parser** | **Create a new Rectangle Element** | xmlParser containing Mock XML | Rectangle element containing width, height, colour, borderWidth, borderColour, x-coordinate and y-coordinate from XML |

| | | | | |
|---|---|---|---|---|
| **FT15** | **Text Parser** | **Create a new Text Element** | xmlParser containing Mock XML | Text element containing font, fontSize, colour, x-coordinate, y-coordinate, width, height and timeOnScreen from XML |
| **FT16** | **Video Parser** | **Create a new Video Element** | xmlParser containing Mock XML | Video element containing url, width, height, x-coordinate, y-coordinate and loop boolean from XML |
| **FT17** | **Presentation Parser** | **Parse a presentation** | Mock XML, slideType | Iterates through each tag in the presentation and extracts the relevant data from XML |
| **FT18** | **Presentation Parser** | **Add element to presentation** | Mock XML containing an element, slideType | Returns slide arraylist containing elements from XML |
| **FT19** | **Canvas View** | **Draw elements to Canvas** | Canvas, Slide, List of Shapes | Shapes in list are drawn to canvas |
| **FT20** | **Slides Recycler View Adapter** | **Add Slide to Container** | parent ViewGroup, view type int | SlideViewHolder within parent ViewGroup |
| **FT21** | **Slides Recycler View Adapter** | **Draw List of Slides** | SlideViewHolder, position | List of Slides drawn at position on Canvas |
| **FT22** | **Slide View Holder** | **Draw Slide** | Slide | Drawn Slide to canvas |
| **FT23** | **UI Utilities** | **Create Interaction List of Presentation Elements** | Mock XML, Parent view, Slide Type String, List item click action, horizontal margin integer | SlidesRecyclerViewAdapter containing slides contained in XML that will implement specified action on click |

# 4. Integration Plan

## 4.1. Integration Strategy

### 4.1.1. Test-Driven Development

Here at Penelope, we have decided to follow the approach of Test-Driven Development (TDD), which emphasises writing tests before writing the actual program code. Since we aim to operate on a professional level, TDD is a key practice that ensures high-quality software delivery.

Our TDD process will comprise the following steps:

1. Write a test case for a specific functionality or feature
2. Run the test and ensure it fails
3. Write the bare-bone code necessary to pass the test
4. Test again to ensure it passes
5. Refactor and alter code as needed to improve quality

Following this methodology will make it easier for us to catch any possible bugs and issues early in the development cycle, reducing the chance of introducing new bugs in the future. This further helps ensure the software is of high quality and meets the minimum requirements of the system.

Additionally, TDD will promote a faster development cycle by avoiding time-consuming debugging sessions later in the development stage, while also making it easier to maintain and update the code over time. By including a comprehensive set of tests, our software team can easily identify areas of the code that are affected by the recently made changes. This should further help to reduce the likelihood of introducing new bugs when updates are rolled out.

Overall, we at Penelope believe Test-Driven Development is a valuable yet simple practice that will help us save a considerable amount of resources and time in the long run.

## 4.2. Order of integration

### 4.2.1. Bottom-Up Integration

During integration, we will be integrating modules from the bottom-up. While this requires more extensive planning before beginning the integration, it prevents being forced to change high-level modules upon discovering changes that need to be made to accommodate unforeseen issues in low-level modules. Starting from the lowest level, modules will be developed in parallel and must pass the specified tests before beginning to develop modules further up the module tree. A consequence of this strategy of integration is that it will take longer to produce a working first iteration, however the first iteration will be more complete and we can implement feedback from customers quicker as all the tools necessary to make the changes will be already available.

At the highest level, due to the close communication inside the quality assurance and test team, a proposed top-level order of integration has been provided as such:

1. Database server (Penelope)
2. SysAdmin desktop application tool (Icarius)
3. Android application (FaunaFinder)

The following order follows a simple, logical and sequential approach to the components being tested. By doing so, we also take into account the various dependencies and constraints that will occur between said components.

- Integrating the **Database server (Penelope)** first:
  In our project development, the database server will act as the groundwork and represent the crucial data storage for the entire application. Therefore, it’s vital that the database server is correctly set up and functions as initially intended before further integrating components that heavily rely on data storage and retrieval.

- Integrating the **SysAdmin desktop application tool (Icarius)** second:
  The SysAdmin desktop application Icarius was developed by our software team. The main aim and objective of this tool is to allow direct interactions with the database server, allowing our developers and other registered 3rd parties to add and edit anything to do with what’s presented within FaunaFinder. Integrating Icarius after the database server gives the application access to a fully functional system. This includes data stored in Penelope and user interactions managed by FaunaFinder. Due to this manner, it allows the test team to ensure Icarius can effectively manage and monitor database elements to ensure smooth operation of the entire ecosystem.

- Integrating the **Android application (FaunaFinder)** last:
  Since FaunaFinder is planned to be the main user-side component of the project, it requires and depends on constant interactions with the database server in order to store, retrieve and manipulate data. By integrating FaunaFinder with Penelope, the test team will be able to reassure the fact that the application successfully communicates with the database server, performs the required database operations and the user experience is as expected, since the main android application can’t interact with an empty database. This procedure will also achieve results in identifying compatibility and other performance issues between the android application and the database server.

## 4.3. How integration testing will be performed

The following subsections define the factors by which the project management will decide whether we’re ready to start, continue and affirm the complete integration tests.

### 4.3.1. Integration Test Entry Conditions

An integration test shall begin when the following protocols are met:

1. The bug logging system is in place and available for use by the testing team.
2. The software and quality assurance teams have configured the subsystems for integration testing. The testing team has been provided with appropriate access to these systems.
3. At least two separate communicating modules are to be released for integration testing under formal, automated source code and configuration operation control.
4. The software and quality assurance teams have prepared a test release, containing at least two communicating elements, both of which have been subjected to unit testing.
5. The software and quality assurance teams have prepared release notes that validate the functionality in the test release and any given bugs and limitations.

### 4.3.2. Integration Test Continuation Conditions

The integration test shall continue provided the following conditions are met:

1. Each integration test release contains only elements under formal, automated source code and configuration operation control.
2. The testing team prepares test releases from the communicating modules that have each completed their unit testing.
3. The testing team accompanies each release with release notes that validate the functionality of said release and any given bugs or limitations.
4. Certain test releases that were built from fairly up-to-date source code can be installed in the current test setting in such a way that the release functions in a stable manner. Additionally, the test team can follow through with planned or exploratory tests against this test release in an effective manner to further gain meaningful test results.

### 4.3.3. Integration Test Completion Conditions

The integration test shall successfully come to an end provided the following conditions are met:

1. The test team has executed all the necessary expected tests against all the expected integration builds.

2. The final integration test release will comprise all modules that will be part of the stable release for the general public version of the product.
3. The quality assurance and test teams agree upon the completion and implementation of adequate integration testing and further integration testing is not likely to locate more bugs related to this matter.
4. The project management will organise a meeting regarding the ending phase of the integration testing and everyone agrees that all required criteria have been met.

#### 4.3.4. System Test Entry Conditions

System Test shall begin provided the following criteria is met:

1. The bug tracking system is in place and available to all required project personnel.
2. The software and quality assurance teams configured the necessary tools to commence System tests. The test team has been provided with appropriate access to said systems.
3. The software team has completed all planned features and bug fixes in previous sprints and has placed all underlying source elements under formal, automated source code and configuration management control.
4. The quality assurance team has completed unit testing of all planned features and bug fixes that are scheduled for future releases.
5. If needed, re-assess the current amount of identified bugs, if any, that were found during Unit and Integration testing. If the amount of present bugs is too large, refer to performing necessary corrective actions to minimise said amount.
6. Penelope holds an in-house introductory meeting for the system testing phase and agree that the first cycle of system testing is ready to begin.
7. The software team delivers a complete revision-controlled software release to the test team.
8. The quality assurance team then creates release notes documenting the features of the evaluation version and all known bugs and limitations.

#### 4.3.5. System Test Continuation Conditions

The System Test shall carry on given the following conditions are met:

1. All software released to our test team is accompanied by release notes. These release notes should include bug reports that the quality assurance team believes have been fixed in each software release, as well as known bugs and limitations.
2. No modifications are made to the main software, whether in source code, configuration files, or other setup procedures or processes without the accompanying bug reports or as part of a planned feature, or as a subsequent incremental bug fix.
3. The software team provides the testing team with complete, revision-controlled software releases built from relatively updated source code each week, if required. These releases can be installed in a test environment to ensure the release works as intended. Additionally, the test team can perform planned or exploratory testing against this version in a reasonably efficient manner and obtain more meaningful test results.
4. If needed, re-assess the current amount of identified bugs, if any, that were found during Unit and Integration testing. If the amount of present bugs is too large, refer to performing necessary corrective actions to minimise said amount.
5. As frequently as needed, commence in-house bug review meetings with the required personnel until the end of the system testing phase.

#### 4.3.6. System Test Completion Conditions

The System test phase shall successfully come to a finish provided the following conditions are met:

1. The current software has not experienced any crashes, hangs, unexpected process terminations or other disruptions in the past.
2. The test team ran all scheduled tests as indicated by the previous iterations.
3. The software and quality assurance teams fixed all bugs that indefinitely needed to be fixed.
4. The test team has verified that all issues were closed or deferred via “monday.com”, and confirmed through regression and confirmation testing where appropriate.
5. The quality assurance team further analyses the work and establishes that the product has reached acceptable levels of quality, stability and reliability.
6. The project management team agrees that the product defined in the final cycle of system testing will meet the appropriate customer and user quality expectations.
7. The project management holds an in-house meeting regarding system testing and agrees that all of the completion criteria have been met.

## 5. Test Environment

Each application will be tested on a range of environments to ensure correct functionality on different operating systems.

Penelope is intended to run on a single environment so will only be tested on a Java SpringBoot server which relies on an underlying PostgreSQL database instance.

Icarius is intended to run on desktop devices, therefore it must be tested on Windows, Mac and Linux operating systems with a variety of screen widths to ensure responsiveness.

FaunaFinder is intended to run on Android devices. There are a large number of Android devices including both phones and tablets hence, the application will be tested on the following most commonly used devices:

- Samsung Galaxy S22
- Google Pixel 2
- Samsung Galaxy Tab 2

The above systems allow the testing of both android phones and tablets, using different screen widths and brands.

## 6. Test Execution

The following section will provide detailed information regarding the actions taken to make the tests happen and other related activities.

### 6.1. Successful Tests and Regression Plan

A test run consists of running all test suites (modules and elements) defined for a particular test phase. The tests will each require their own respectable time to complete, depending on the significance of the element under test. The test team should repeat these test iterations multiple times over the course of each testing cycle. Due to this, essentially repeating the same set of tests forms our regression strategy for this project. However, the test case coverage for each test suite will improve with each new update release, which results in the number of test cases increasing, therefore allowing our test team to debug specific test objects.

### 6.2. Test Cycles

A test cycle consists of a subset of test passes performed against a single, identifiable test release. Integration testing will run on a very frequent basis (numerous per week), whereas system testing will be performed on rarer occasions (once a week). Tests will start running as soon as the updated software version of the product is properly installed. The test team will run tests abiding these rules:

- **Confirmation tests.** The test team will perform additional tests to check any bugs that were reported as fixed in the update release notes. For fixed bugs, the tester closes the corresponding bug report. Likewise, re-open the report of unresolved bugs.
- **Scheduled tests**. The test team runs all test cases planned for this test cycle, starting with the test cases classified as high risk. Test cases that have never been run before, especially test cases for newly developed features, tend to be the most risky.
- **Exploratory testing**. At the end of each testing cycle, at least one member of the test team runs exploratory tests against the software based on where the present test coverage is low and bugs are believed to be undetected. If testers find a large number of defects in these areas, especially if those defects indicate major gaps in risk-based test design strategies, the test team will add tests to fill the identified gaps.

When the test team completes these tests, the required personnel will run any other test cases not scheduled in this cycle, but not yet run in the current project build. Then, once those test cases are completed, the test team runs exploratory tests until the next version starts in the next cycle. Conversely, if the test team is unable to complete all planned test cases in each cycle due to delivery delays, deadline pressure, high debug rate or other related reasons, they are to notify project management, who will then reschedule the unfinished test cases immediately after the confirmation test, making this a priority in the next cycle.

### 6.3. Test Execution Procedure

The purpose of the integration and system testing phases is to find errors in the product under test, the interfaces between each component, or the system as a whole. Our test team plans to do this by running a series of manual and automated test cases for each build. At the beginning of each test cycle, each member of the test team personnel is assigned their own set of test cases to check. These test cases should all be different for each test cycle and invalid assumptions by testers should not lead to test breakouts. Once these testers are assigned to their respective test cases, they run each case according to the steps listed in the test case, repeating the process until the list is exhausted. If a member of the team has finished all their test cases before the end of the cycle, they can provide support for other test personnel by re-assigning some of their cases to themselves. Once all testing is complete, the test team then moves on to the next cycle and the process repeats.

## 7. Risks and Contingencies

The following table outlines the key risks to success of this project, and contingency and/or mitigations plans to address them.

| Risk | Contingency/Mitigation |
|---|---|
| Breakdown in resolution and escalation processes | 1. Avoid blocking issues and accept inefficient progress and delays in completing plans<br>2. Stop testing on a continuation basis and resolve process issues<br>3. Continue testing with the workaround fix while a resolution is being worked on for the next iteration |
| External 3rd party can’t install the final application (end-to-end user testing) | 1. Thoroughly test the general public release to avoid problems before sending initial installation<br>2. Submit a detailed bug report to the company |

| | |
|---|---|
| A last-minute change that significantly impacts requirements, design, functionality, or other parts of the development plan | 1. Accept increased quality risk from incomplete testing of changes<br>2. Accept increased budget risk by adding or offloading company personnel to run sufficient last-minute tests<br>3. Accept increased scheduling risks by delaying the release date for general public |
| An insufficient test release was detected after the testing cycle has started | 1. Set up automated “smoke” testing into releases to detect any issues<br>2. Halt the test during the “flawed” test cycle, return to the last known good test release to continue testing and accepting that those planned tests are underachieving and loss of efficiency is inevitable |
| The test environment is incomplete on the entry date of the integration testing or system testing phase | 1. Start with the tests that can be run in the environment currently available and accept the limited test execution, inefficient progress and large gaps in test coverage<br>2. Postpone the start and end dates of the testing phase until the environment is available |
| The test environment is incomplete for more than one test cycle | 1. Revise plans to extend phases as needed, remove all tests that depend on missing configurations and identify increased project quality risks |
| Technical support for the test environment is unavailable or inadequate | 1. Accept inefficient progress and planned won’t be executed<br>2. Assign extra personnel for short-term support on certain aspects |
| Erroneous deliverables slow down testing development and reduce overall test coverage | 1. Thorough unit testing by the quality assurance and test teams prevent problems before they escalate further<br>2. Adhere to continuity standards, stop testing if problems worsen and accept schedule delays<br>3. Attempt to continue testing according to the current schedule and accept low quality system releases to the general public |
| Gaps in test coverage | 1. Exploratory testing enables testing in areas not covering by planned testing<br>2. Structural coverage analysis techniques, field-error analysis and customer data may be used to spot gaps that need to be filled. This should most likely be determined during the test design and test development phases based on available resources |
| Deviations in the development plan affect the preparation of entry conditions for the planned dates | 1. Stick to starting criteria, perhaps reducing the number of features deployed by moving timelines across tests and cycles and remove final increments of features<br>2. Accept that possible violations of the entry conditions may occur and run the risk of poor quality due to insufficient time to find and fix bugs. This can include the risk of failure of the entire project |
| Test team personnel absent or unable to perform | 1. Accept a temporary slowdown of scheduled test execution<br>2. Acquire short-term contract testers with the right skills as soon as possible |
| Inability to hire suitable automated test personnel | 1. Use any means available to find the ideal candidate<br>2. Increase the pass duration to slow down the process of test executions |

| | |
|---|---|
| Debugging in test environment | 1. Provide customer-like configurations for testing and manage configurations to ensure error reproducibility<br>2. Accept the slowdown or interruption of test plan execution in addition to the inefficiency caused by having to restore the test environment after each debugging session |