ARCHITECTURAL NOTEBOOK

INSPECT

1. PURPOSE

The architecture of the Inspect application will adhere to the criterion of usability, security, availability, recoverability, compatibility, and maintainability/configurability. These requirements will inform the design of a system that will minimise concerns of recovering data, render a user-friendly service that doesn't differentiate between user types, be mostly available for offline use, and be constructed from hardware and software that not only should be available but easy to maintain, modify and repair. The system will employ a loosely-coupled, multi-tier architecture rooted in a client-server based pattern. This will ensure each of the layers of the application's system are able to effectively communicate with each other, allowing for ease of reusability with regards to the use and modification of templates, whether they be new or existing, and offer the development team a greater degree of efficiency when designing, implementing and modifying/updating the distinct tiers that constitute the overall architecture of the system.

2. ARCHITECTURAL GOALS AND PHILOSOPHY

While not limited to, the key non-functional requirements are:

- **USABILITY:** Apply good interactive design practices and concepts such as affordances and familiarity to ensure optimal usability. It is important that the system is easy to use. As such the application should be designed in a way that it is easy to use without or with minimal assistance from tutorials or documentation, especially when taking into account that inspectors of various backgrounds and technological capabilities will be using this application to record faults, courses of action and interact with clients. A consistent and uniform system should be implemented to accommodate this.
- **RELIABILITY:** The main bulk of the system will be designed to work offline as potentially some inspection sites may not have service. Share features of the app (i.e. interactions with the client) will however require an internet connection. The system also needs to be stable to prevent loss of work, especially with a system built to address the time and labour intensive duplication of paperwork.
- **PERFORMANCE:** The system needs to respond promptly to input by the user as in keeping with the streamlining of the inspection process. If the system is busy, loading screens and other prompts will be used to convey status to users.
- **SECURITY:** The application will not store user data, require logins, or transmit data. Any data transfers will be handled by third parties. Any passwords will be handled by third party services (email client for example).
- **AUDIT:** Interactions, errors, and statuses are logged. Not user filled data. This log is held locally. Bug reports will be transmitted via google play store services.
- **CAPACITY:** Only one device is required to use the system. While the application itself will have a small installation size, the devices free space will determine how many templates and pdfs can be stored locally.
- **COMPATIBILITY:** As the target hardware is android mobile phones, the devices need to have touch input and rear facing camera. To minimise compatibility issues with outputs, PDF has been chosen as this format is able to be read by default by many systems.
- **MAINTAINABILITY:** The system will conform to a modular architecture this will be achieved by breaking features down into components that are cohesive and loosely coupled with each other. This approach will make source code easier to read, understand, and debug while also decreasing effort required to update individual components. Taking into account the various technical backgrounds of the users, in-app assistance through tutorials or other supplementary documentation should be made available to minimise the learning curve and allow for a better understanding of the system to those responsible for maintaining it.

3. ASSUMPTIONS AND DEPENDENCIES

DEPENDENCIES

- People to review the system's requirements and answer questions during usability testing. This also extends to users contributing ideas, suggestions and user stories to enhance the overall functionality, usability and features of the application. This process is to be repeated in order to track the improvements and ensure any problems which under the usability and functionality of the app are minimised.
- Availability of hardware and software, i.e. mobile phones which run the Android operating system, as well as ones which have touch input, a rear-facing camera, email client, and are able to view and read the exported documents in the PDF format - whether in Create Template, Edit, Inspection or Share File mode.
- The various technical backgrounds of the inspectors as well as the clients.

ASSUMPTIONS

- There will be no differentiation of user types.
- There will be no account setups, and thus no conditions or requirements on passwords to access the system.
- The system will timeout after a period of 5 minutes of no interaction between the user and the touch screen, especially if they do not have a lock screen. When the user unlocks the device, the system should resume in the same state it was in prior to timeout.
- As far as usability, it will be assumed that the app will only be available in the English language.
- Training (and the potential costs of it) for Inspectors to be able to use the application on site.
- Time and cost associated with the training of developers and auditing, ensuring that coding standards established in the Google Java Style Guide are adhered to during the review process prior to merging in GitHub. Unit testing will also be mandatory.
- Data will only be stored on the local device and emailed to the client when required.
- There will be multiple users of this system, but they will not be able to interact with one another through the application.
- The system will interact with other components on the device such as the keyboard and camera.
- The system should have a reasonably fast start up time, but not necessarily instant.
- Keyboard input should appear instantaneously in the inspection.
- Load time for the keyboard to appear on the screen should be minimal (<1 second).
- If the user is typing information the application should be able to keep up and display that. Likewise, if the user clicks on the area to take a photograph, the camera should load within an acceptable time. This is the same for when creating the template.
- When a picture is taken and saved, the application should refresh and display the image in the template being filled out. Text should be displayed as it is inputted. When creating a template and new elements are added it should refresh to display those.
- Exporting the template to PDF should be carried out as quickly as possible so it can be emailed or saved as required. This is the same for exporting the template once it is created, it should be done as soon as possible. This is so it can also be emailed or saved as required.

- The system should be able to load the template as quickly as possible to allow the user to commence the inspection immediately.

- There will be no delay between loading up the camera, taking photos and saving it to the document. The photo will be cropped to the required size and the image should be automatically displayed in the field used.

- The system should be able to run on one device.

- The system should be able to store data locally on the device.

- There is no time limit on how long the data can be stored on the device. Templates should be stored on the device as long as the user desires.

- The system will be operational 24/7 and outside of the sharing feature with clients, will also be available to use offline.

- During periods of downtime or system failure, errors or bugs should be tracked, logged and dealt with as soon as possible with updates pushed to fix these problems.

- Input should only be touch via the phones screen, text via the phones keyboard and images from the camera on the phone.

- The total file size of photo attachments should be well within the range of the 25 MB limit allowed for Gmail attachments. Photos will have to be compressed to accommodate this. Other data such as templates and PDFs should also be sized and stored correctly.

- To optimise usability, interactive design standards will need to be implemented during the planning phases and enforced during the review process prior to merging in GitHub.

4. ARCHITECTURALLY SIGNIFICANT REQUIREMENTS

- With the exception of the sharing feature between Inspector and Client, the system must be available to use offline, especially in cases where inspection sites may not have service.
- The system will conform to a modular architecture, where it will be broken down into components that are cohesive and loosely coupled with each other. This is significant because the architecture allows for ease of reusability with regards to the use and modification of templates, whether they be new or existing, and offer the development team a greater degree of efficiency when designing, implementing and modifying/updating components.
- The system must store data locally on the device. Allows for offline access of resources for both the user and the system and avoids security issues of storing user data on a server.
- To satisfy system usability, user acceptance testing will be conducted with pilot users and it is expected that they will be able to use the system with minimal assistance. This will ensure the Inspector can effectively focus on communicating the site's faults and key courses of action, unimpeded by any technical difficulties, shortcomings or the occurrences of bugs which have been minimised, to the Client.

5. DECISIONS, CONSTRAINTS, JUSTIFICATIONS

- As part of the design decision, tight-coupling will be avoided as it will render the system a lot more difficult to maintain later, which is why a loosely coupled, multi-tier architecture is favored. This approach allows for different components to be designed and implemented as separate modules that can be reused, making the source code easier to read, understand, and debug while also minimising the need to perform changes on multiple components to update one feature.
- The system must use a common, popular programming language, one that is virtually compatible across a wide variety of platforms such as Java.
- The system must employ common design and architecture patterns that are more readily understood by technicians, especially those working on repairs to the application's source code, i.e., the source code must be readable and commented for clarity for other technicians to follow and thus assist in any debugging/repairs. This, coupled with adherence to coding patterns, can enhance the maintainability of the system.
- Sufferers of color blindness must also be taken into account; colors which are not visible to them must be avoided to enhance interface accessibility. Employing. This would essentially entail taking into account the most common forms of colorblindness, such as red-green color vision deficiency, employing the aid of symbols along with colours, minimising the overall color palette and avoiding undesirable color combinations. In such as case, a blue/white color palette can satisfy the aforementioned criteria. This will be managed during the user acceptance testing phases on the part of the developers as well as the actual testers during the feedback stages.

6. ARCHITECTURAL MECHANISMS

Template Storage / File Management

In terms of capacity and availability, all templates and PDFs are stored locally. These templates should be stored for as long as the user requires and will be loaded from and saved to local storage, either from the application or an email. PDF size will be made to adhere to the 25 MB limit for email attachments. To aid this photos will have options that allow you to edit/crop the images, and/or compress them to lossy file formats such as .jpeg/.png to reduce file size. As all files are stored locally a GUI will be needed to interact with files.

Document / File Sharing

In terms of security and reliability, email will be employed to facilitate document sharing between the Inspector and the Client. Maximum file size for attachments must adhere to the 25 MB limit for email. This feature will require an internet connection, with passwords handled by third party services such as the email client. To satisfy compatibility, PDF will be selected as the default file type due to its readability across many systems.

Attaching/Adding Photos

It will be necessary to take and attach photos to the template in order to better illustrate the nature of any faults or defects that the Inspector may encounter during an Inspection. During the Create Template phase, the user will be able to 'Add Camera', which would allow them a placeholder to take and attach a photo the template during the Inspection phase. Once the Start Inspection phase commences, the user will be able to access the placeholder, which will directly lead them to the device's camera to take a photo and approve of attaching it to the template.

Modifying Templates

It will be necessary to manipulate formatting of a blank template or existing template. A GUI will be employed to add or remove elements of various complexity to the template. This modified template will then be saved locally to the

device and template will retain formatting when loaded. This template will also comply with PDF conversion requirements, especially when fields within elements have been filled by user during off site inspection.

Input / Output

Ultimately the focus of all other architectural mechanisms is to support an on site inspection. As such the application needs to support user input into a template. Text fields, check boxes, and images need to be added to templates during an inspection. At the end of the inspection, the template needs to be output as a pdf retaining formatting of the original template.

7. KEY ABSTRACTIONS

- **Inspector:** Person hired by the Client to perform an Inspection of a site. They are responsible for filling out a template that includes Photo attachments, either ones taken on the spot by them or imported from an existing library/album. They may edit and annotate these photos to better communicate the nature of the faults and the key courses of action to the Client. Once they have filled out the template, they will export it as a PDF and share it with the Client.
- **Client:** A person/company that hires the Inspector to inspect a site. Also responsible for reviewing the inspection document sent to them by the Inspector and carrying out the suggested courses of action to fix the recorded faults.
- **Inspection:** An action/job conducted by the Inspector on behalf of the Client to record any faults that may be present at a chosen site and suggest courses of action to minimise and/or eliminate the faults.
- **Fault:** A feature of the inspection site that may be defective and thus present a safety hazard if left unattended. It is recorded by the Inspector, who will also suggest remedying actions to the Client.
- **Inspection Document/Report:** A document filled out by the Inspector which details the condition of the inspected property, including a record of the faults which it may possess as well as courses of action that need to be taken to repair them. This document is then sent to the Client by the Inspector to review and carry out the suggested courses of action to repair the faults at the site.
- **Inspection Site:** A location where the Inspector has been sent to by the Client to inspect. May contain faults that need to be recorded and fixed.
- **Photo:** A visual record of the fault that not only backs up the textual information provided in the Inspection Report/Document, but further highlights the extent/severity of the fault as well as the mechanics and logistics of the reparative actions to the Client.

8. LAYERS OR ARCHITECTURAL FRAMEWORK

A multi-tier or layered architecture made up of a presentation tier, logic tier, and data tier will be employed. This multi-tier architecture is known to be part of a Client-Server architectural pattern where each layer is distributed between the client and the server. The presentation tier will constitute the user interface, the data tier will manage the storage and retrieval of information from a file/database system, and the logic tier will deal with the communication and coordination between the user interface and the data tier. The separation of these distinct tiers is suitable for applications that are required to access data (e.g. accessing photos from an existing library/album to attach to the template). This type of architectural pattern allows for the following benefits:

- Allows for a greater degree of efficiency on a development level as each team/person can be assigned a separate tier/layer to work on.
- Flexibility through the modification or adding of features due to the separation of distinct layers.

- Ease of reuse by being able to use the same program (in this case, a template, or the app itself) with different data (for a new/different inspection requiring either the same template with different data, or a new template entirely) through the replication of the logic and presentation tiers to create a new data tier.

The sharing feature between the Inspector and Client may also require a Client-Server based architectural pattern that is commonly used in online applications that offer document sharing. It is through this pattern that clients are able to request and receive services from servers.

Multi-Tier Architecture: Lessons Learned During The Elaboration Phase

Examining the main classes:

- **File Manager:** Both File Manager and File Share Manager were integrated, functioning as the logic layer of the security and file accessing/provider system. Templates can be created, loaded and deleted. Integrating the Storage Access Framework (SAF) proved to be quite troublesome while simultaneously learning the Android structure. The ability to navigate through the storage was required for the application. Fortunately we had the ability to utilise this system provided by the Android OS. The SAF also integrated with the share and camera functions, this allowed saving, loading, and sharing files with the application due to its ability to grab the URI of a selected file.

- **File Select Activity:** Functions as the File Share Manager in the logic layer of the application, and is part of the file provider system. Once the inspection has been completed, the templates are converted via the Conversion Manager into PDF files that can be sent to or shared with the client. With the file sharing function there was some difficulty concerning the registration of certain aspects of the class within the Android Manifest, but this was able to overcome this through a combination of extensive research online, assistance from other team members and correctly updating the File Provider permissions within it. Other areas where issues arose included a few of the methods that needed to call variables, which in turn needed to be set up by creating a filepaths .xml with TextView. Trying to run these components proved bothersome as the emulator either kept shutting down or failed to load certain aspects of the application that appeared to run fine for other team members; however, this was soon rectified by creating a new emulator and running the application on it.

- **Log Manager:** As part of the logic layer, it affords classes the ability to display and report errors that may arise from any one of their methods as well as writing messages to the log file on the device. There were some problems initially in ensuring the application was successfully saving and reading log files via the Log Manager. The method itself had to be revised from previous iterations as the buffered writer employed was not creating the log files in the directory. Even though we tried to pinpoint the issue through research and online tutorials and came up short in identifying the issue with the buffered writer, an alternate method in outputFileMethod was employed, which managed to successfully read, save and create the file.

- **Photo Manager:** Also part of the file provider system, alongside the File Manager and File Select Activity class. It is part of the logic layer that allows the user to obtain a photo from the camera, or the gallery as well as save a taken photo to the gallery. Photos can also be altered in terms of their size to accommodate photo attachment capacity limits upon sharing the file/template with the client. There were initial issues regarding the File Provider, mainly in returning the correct values for the URI's in creating a file (and directory) and sharing it. This also lead to other issues such as being unable to take a photo and save it to the photo gallery. We ended up working on alternate implementations, ensuring null values weren't being returned for the URI's, which was part of the initial problem. Through a process of trial and error and extensive research, we were able to correct these errors so that files could be shared and photos saved to gallery.

- **Template Manager:** This class was originally going to be responsible for the creation, loading and deletion of templates, but was ultimately discarded due to the fact that these main functions were implemented in the File Manager class.

- **Template Editor:** This class constitutes the UI lifecycle. Once again, it is part of the logic layer of the architecture that is responsible for the inclusion (and removal) of elements, models, objects and fields in the template upon its creation, which can also be saved. It also provides a view of the template when it is converted as PDF before it is sent to the client via the File Select Activity implementation. The scope of the project was initially compromised as a result of errors associated with the Template Editor that were discovered late in the development process. The device would reset once the view was rotated to accommodate the landscape position. This, however, was able to be remedied by implementing rooms, live data, and view model libraries provided by Android, which in itself provided to be time consuming due to the fact that the associated tutorials for such implementation were executed in Kotlin rather than Java. These could be utilised in such a way that the data was able to be fed into the views rather than being held by them.

- **Conversion Manager:** This was ultimately discarded due to the fact that Android proved sufficient in solely handling the conversion process (from template to PDF) via a view adapter, therefore removing the need to implement a class that handles these responsibilities.

- **PDF Export:** The ViewPrintAdapter handles the PDF export by accepting a view, rendering said view as a bitmap, and placing each bitmap as a page in a PDF document. There was some minor issues around formatting of views, and as we move forward we need to ensure that the view formatting is respected. One issue left unresolved is that the Android Print to Pdf wants to place the bitmap centrally on the page, but as the app progresses this will be solved either directly in the PDF printer or with an element weight system that pads the image to achieve the desired alignment.

- **Inspector:** This class was originally conceived to accommodate the editing of image and text fields. Due to issues with Template Editor, the editor and inspection functionality was moved to the same class. In editor mode, elements could be added and edited, while in inspection mode this editing functionality is omitted and labels are read only leaving only answer fields to be editable by the user/inspector.

Multi-Tier Architecture: Lessons Learned During The Construction Phase

Examining the main classes:

- **File Manager:** Intended to facilitate creating, loading and sharing of templates, the Storage Access Framework (SAF) proved to be troublesome when used to set default launch location of a file. Multiple attempts were made at examining the lines of code in File Manager that were causing the issue but ultimately failed to yield any results. Instead, an alternate implementation was employed in the form of a File Selector library, which managed to solve the issue plaguing the file sharing features of the App.

- **Inspector:** Remained a part of the Logic Layer and expanded upon in functionality to accommodate the following: allowing the user to fill out the template fields during the Inspection phase, allowing the user to save the template (via communication with the Template and Element classes, also in the Logic Layer), allowing the user to navigate back and both between various pages, allowing the user to open the camera and take a photo to attach the template (via communication with the Photo Manager, also in the Logic

Layer) as well as allowing the user to print the inspection form as a PDF file (via communication with View Print Adaptor, also in the Logic Layer).

- **Template:** Template has now been modified to communicate with Template Page and Template Element to add/remove elements and pages.

- **Template Element:** Template Element has now been modified to communicate with Template Page to facilitate the adding/removing of elements and pages during the Create a Template and Edit a Template phases.

- **Template Page:** Template Page communicates through the Template Element to the newly created Element classes - Element Spacer Field (to add spaces to the template), Element Heading Field (to add heading fields to the template), Element Image Field (to add image placeholders to the template, which in turn will allow the user to attach photos during the inspection process), Element Paragraph Field (to add paragraph fields for long and short questions added to the template), and Element Text Field (to add text fields which function as responses to the headings and questions added to the the template).

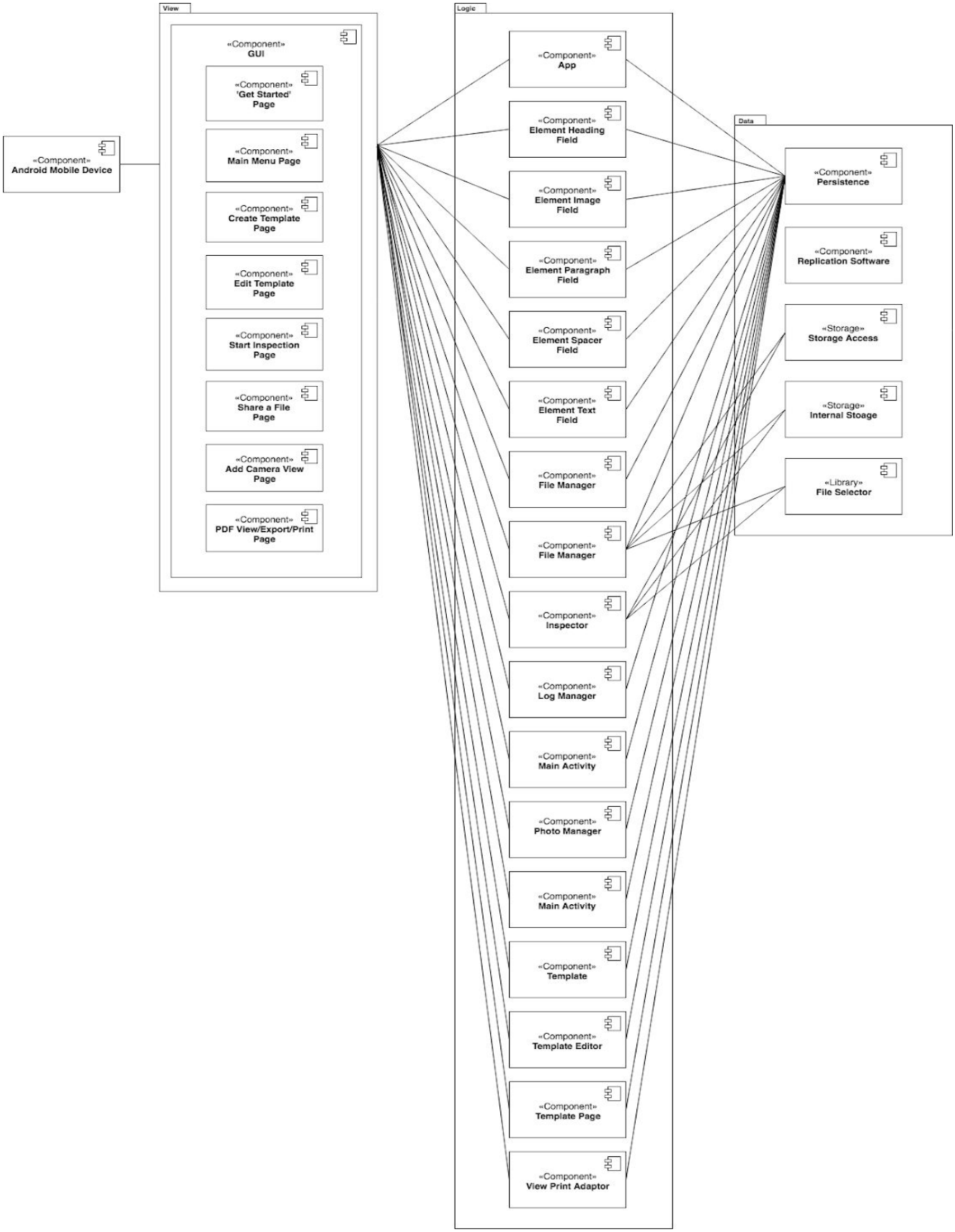
- **Storage Access:** Storage Access exists in the Data Layer of the architecture and was implemented to work in conjunction with the File Selector library employed (as existing within the same Data Layer) to replace the previous SAF implementation that was preventing a default file location to be set within the internal storage to rectify this issue.

9. ARCHITECTURAL VIEWS

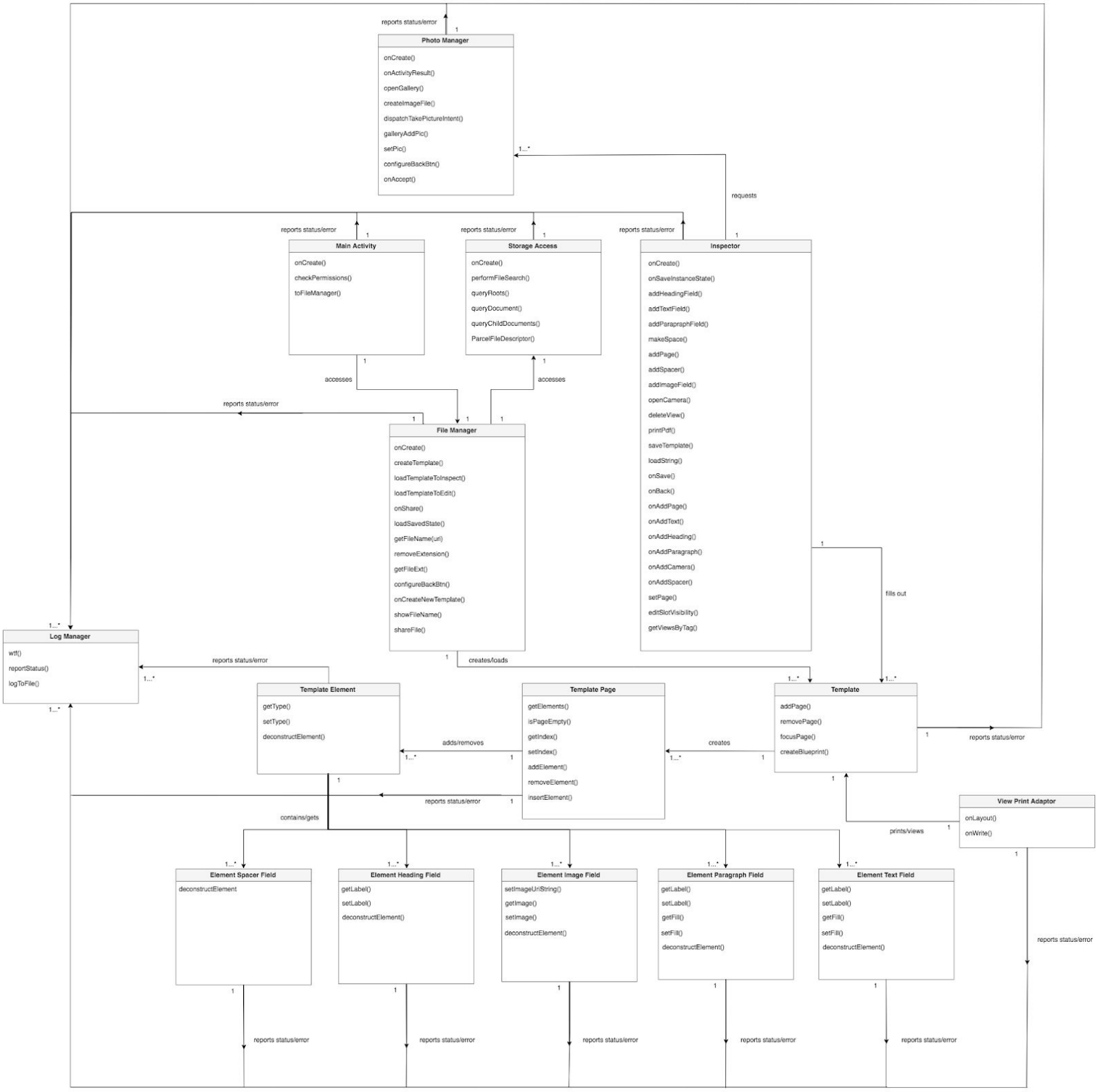
- **Logical Views**

COMPONENT DIAGRAM (Presenting Architecture):
https://drive.google.com/file/d/16miVlBIDFqxWfBUj65mJ3NjH5CSq47CG/view?usp=sharing

(ON NEXT PAGE)


View
«Component»
GUI
«Component»
'Get Started'
Page
«Component»
Main Menu Page
«Component»
Create Template
Page
«Component»
Edit Template
Page
«Component»
Start Inspection
Page
«Component»
Share a File
Page
«Component»
Add Camera View
Page
«Component»
PDF View/Export/Print
Page
«Component»
Android Mobile Device
Logic
«Component»
App
«Component»
Element Heading
Field
«Component»
Element Image
Field
«Component»
Element Paragraph
Field
«Component»
Element Spacer
Field
«Component»
Element Text
Field
«Component»
File Manager
«Component»
File Manager
«Component»
Inspector
«Component»
Log Manager
«Component»
Main Activity
«Component»
Photo Manager
«Component»
Main Activity
«Component»
Template
«Component»
Template Editor
«Component»
Template Page
«Component»
View Print Adaptor
Data
«Component»
Persistence
«Component»
Replication Software
«Storage»
Storage Access
«Storage»
Internal Stoage
«Library»
File Selector

CLASS DIAGRAM (Presenting Physical/Logical Views):
https://drive.google.com/file/d/1b0scetAohVRlhhTY8PtMV_5OnM2ug6QG/view



- **Operational Views: How the Architecture Implements the CCRD Use Cases**

CREATING NEW TEMPLATE (AND SAVING):

1. The user accesses the GUI of the Android Mobile Device by tapping on the Inspect App. Icon.
2. The user arrives at the 'Get Started' page via the GUI and taps the 'Get Started' button to access the Main Menu.
3. The user arrives at the Main Menu page and taps the 'Create a Template' button to begin creating a template.
4. The GUI communicates with the File Manager in the logic layer to create a template.
5. The GUI communicates with Template Element within the Logic Layer to add/remove elements such as heading, image, paragraph, spacer, and text fields as well as template pages. The Template Element itself communicates with Element Heading Field, Element Image Field, Element Paragraph Field, Element Spacer Field, and Element Text Field respectively also within the Logic Layer to achieve this.
6. The GUI communicates with Inspector in the Logic Layer to save the template, which in itself communicates with Internal Storage and File Selector in the Data Layer.

EDITING AN EXISTING TEMPLATE:

1. The user accesses the GUI of the Android Mobile Device by tapping on the Inspect App. Icon.
2. The user arrives at the 'Get Started' page via the GUI and taps the 'Get Started' button to access the Main Menu.
3. The user arrives at the Main Menu page and taps the 'Edit a Template' button to begin editing an existing template.
4. The GUI communicates with the File Manager in the Logic Layer (which in turn communicates with Internal Storage, Storage Access File Selector in the Data Layer) to access and load the template in edit mode.
5. The GUI then communicates with Template, Template Element, and Template Pages in the Logic Layer to make any edits by adding/removing elements through the Element components.
6. The GUI communicates with Inspector in the Logic Layer to save the template, which in itself communicates with Internal Storage and File Selector in the Data Layer.

PERFORMING INSPECTION (FILLING TEMPLATE OUT):

1. The user accesses the GUI of the Android Mobile Device by tapping on the Inspect App. Icon.
2. The user arrives at the 'Get Started' page via the GUI and taps the 'Get Started' button to access the Main Menu.
3. The user arrives at the Main Menu page and taps the 'StartInspection' button to begin editing an existing template.
4. The GUI communicates with the File Manager in the Logic Layer (which in turn communicates with Internal Storage, Storage Access File Selector in the Data Layer) to access and load the template in inspection mode.
5. The GUI then communicates with Inspector in the Logic Layer to fill out the template fields, and in the case of adding a photo, communicates with the Photo Manager, also in the Logic Layer, to take a photo and add it to the template.
6. The GUI communicates with Inspector in the Logic Layer to save the template, which in itself communicates with Internal Storage and File Selector in the Data Layer.
7. The GUI communicates with View Print Adaptor in the Logic Layer to view and/or export the template as a PDF file.

SHARING TEMPLATE WITH CLIENT:

1. The user accesses the GUI of the Android Mobile Device by tapping on the Inspect App. Icon.
2. The user arrives at the 'Get Started' page via the GUI and taps the 'Get Started' button to access the Main Menu.

3. The user arrives at the Main Menu page and taps the 'Share a File' button to begin editing an existing template.
4. The GUI communicates with the File Manager in the Logic Layer (which in turn communicates with Internet Storage, Storage Access File Selector in the Data Layer) to access and load the filled out inspection form (as a PDF file).
5. The GUI then communicates with the File Manager in the Logic Layer, which in turn communicates with the File Selector library implementation in the Data Layer to send/share the PDF file.

ERROR MANAGEMENT:

1. GUI communicates with the Log Manager in the Logic Layer to report an error, report the status of an error, log an error and display an error.

- **Use Case:** A list or diagram of the use cases that contain architecturally significant requirements.