Inspect
Architecture Notebook

1. Purpose

This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation.
[Always address Sections 2 through 6 of this template. Other sections are recommended, depending on the amount of novel architecture, the amount of expected maintenance, the skills of the development team, and the importance of other architectural concerns.]

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 minimize 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

[Describe the philosophy of the architecture. Identify issues that will drive the philosophy, such as: Will the system be driven by complex deployment concerns, adapting to legacy systems, or performance issues? Does it need to be robust for long-term maintenance?

Formulate a set of goals that the architecture needs to meet in its structure and behavior. Identify critical issues that must be addressed by the architecture, such as: Are there hardware dependencies that should be isolated from the rest of the system? Does the system need to function efficiently under unusual conditions?]

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 to 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 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 do need to have touch input and rear facing camera. To minimize 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 minimize the learning curve and allow for a better understanding of the system to those responsible for maintaining it.

3. Assumptions and dependencies

[List the assumptions and dependencies that drive architectural decisions. This could include sensitive or critical areas, dependencies on legacy interfaces, the skill and experience of the team, the availability of important resources, and so forth]

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.
- 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 read the exported documents in the PDF format.
- 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.
- 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.

- 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 a photo 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.

- 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

[Insert a reference or link to the requirements that must be implemented to realize the architecture.]

- 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.
- The system must store data locally on the device.
- 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.

5. Decisions, constraints, and justifications

[List the decisions that have been made regarding architectural approaches and the constraints being placed on the way that the developers build the system. These will serve as guidelines for defining architecturally significant parts of the system. Justify each decision or constraint so that developers understand the importance of building the system according to the context created by those decisions and constraints. This may include a list of DOs and DON'Ts to guide the developers in building the system.]

- 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 minimizing 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. This will be managed during the user acceptance testing phases.

6. Architectural Mechanisms

[List the architectural mechanisms and describe the current state of each one. Initially, each mechanism may be only name and a brief description. They will evolve until the mechanism is a collaboration or pattern that can be directly applied to some aspect of the design.]

Architectural Mechanism 1

[Describe the purpose, attributes, and function of the architectural mechanism.]

In terms of capacity and availability, databases can be employed to store templates (ones that have been saved, whether they be partially completed, fully completed, or empty templates with specific modules/fields that the user can select from the 'Created Templates List Page' - see Component Diagram in Section 9). 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. Photo file size capacity will be made to adhere to the 25 MB limit for Gmail attachments via options that allow you to edit/crop the images, and/or compress them to lossy file formats such as .jpeg to reduce file size.

Architectural Mechanism 2

[Describe the purpose, attributes, and function of the architectural mechanism.]

In terms of security and reliability, file sharing services such as Google Drive will be employed to facilitate document sharing between the Inspector and the Client. If using Google Drive, maximum file size for attachments must adhere to the 25 MB limit for Gmail. 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.

7. Key abstractions

[List and briefly describe the key abstractions of the system. This should be a relatively short list of the critical concepts that define the system. The key abstractions will usually translate to the initial analysis classes and important patterns.]

- 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 minimize and/or eliminate the faults.
- Fault: A feature of the inspection site that may be defected 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

[Describe the architectural pattern that you will use or how the architecture will be consistent and uniform. This could be a simple reference to an existing or well-known architectural pattern, such as the Layer framework, a reference to a high-level model of the framework, or a description of how the major system components should be put together.]

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 co-ordination 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.

9. Architectural views

[Describe the architectural views that you will use to describe the software architecture. This illustrates the different perspectives that you will make available to review and to document architectural decisions.]

Recommended views

- **Logical:** Describes the structure and behavior of architecturally significant portions of the system. This might include the package structure, critical interfaces, important classes and subsystems, and the relationships between these elements. It also includes physical and logical views of persistent data, if persistence will be built into the system. This is a documented subset of the design.

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

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

- **Operational:** Describes the physical nodes of the system and the processes, threads, and components that run on those physical nodes. This view isn't necessary if the system runs in a single process and thread.

CREATING NEW TEMPLATE (AND SAVING): (1) Inspector accesses the GUI of the Android Mobile Device by tapping on the app icon. (2) Inspector will then access the 'Template Creation Page' via the GUI. (3) GUI communicates with the File Manager to create a template. (4) GUI communicates with the Template Editor to add/remove objects and modules (or fields) that the Inspector requires for the inspection. (5) GUI communicates with the Template Editor to save the template. (6) The template is stored in a database.

LOADING SAVED/INCOMPLETE TEMPLATE: (1) Inspector accesses the GUI of the Android Mobile Device by tapping on the app icon. (2) Inspector uses the GUI to access the 'Created Templates Page' or 'In Progress/Completed Templates Page' to load a template. (3) GUI communicates with the File Manager to load the template from the database.

EDITING AN EXISTING TEMPLATE: (1) Inspector accesses the GUI of the Android Mobile Device by tapping on the app icon. (2) Inspector uses the GUI to access the 'Created Templates Page' or 'In Progress/Completed Templates Page' to load a template (stored in the database). (3) GUI communicates with the File Manager to load the template. (4) Inspector edits the template and saves changes. (4) GUI communicates with the Template Editor to save the changes. (5) This edited template is stored back in the database.

PERFORMING INSPECTION (FILLING TEMPLATE OUT): (1) GUI accesses the File Manager to load the template (stored in the database). (2) Inspector uses the GUI to fill out the 'Title Inspection Page', 'Inspection Map/Location Selection Page', 'Inspection Ground Checklist Page', 'Inspection Fault Notes Page', 'Inspection Fault Image/Photo Attachment Page', and 'Inspection Fault Edit Image/Photo Attachment Page'. (3) GUI communicates with the Photo Editor to take a photo or get it from library, and alter photo if needed. (4) Inspector signs off on the finished template through the GUI's 'Inspector Digital Signature Page'.

DELETING TEMPLATE: (1) Inspector accesses the GUI of the Android Mobile Device by tapping on the app icon. (2) Inspector uses the GUI to access the 'Created Templates Page' or 'In Progress/Completed Templates Page'. (3) GUI communicates with the File Manager to load the template. (4) Inspector selects the 'Individual Template Checklist Page', with the option to 'Delete'. (5) GUI communicates with the File Manager to delete the template.

SHARING TEMPLATE WITH CLIENT: (1) Inspector signs off on the finished template through the GUI's 'Inspector Digital Signature Page' (2) Inspector exports the template to a PDF format through the GUI's 'Inspection/Template Export/Share' page. (3) GUI communicates with Conversion Manager to convert template to PDF, preview the document and save the PDF if desired. (4) GUI communicates with File Manager to open the PDF file. (5) The File Manager attaches PDF object to the File Share Manager and sends/shares it with the client via a file sharing service such as Google Drive.

OR

ALTERNATIVELY: (1) Inspector accesses the GUI of the Android Mobile Device by tapping on the app icon. (2) Inspector uses the GUI to access the 'Created Templates Page' or 'In Progress/Completed Templates Page' to load a template. (3) GUI accesses the File Manager to load the template. (4) Inspector exports the template to a PDF format through the GUI's 'Inspection/Template Export/Share' page. (5) GUI communicates with Conversion Manager to convert template to PDF, preview the document and save the PDF if desired. (6) GUI communicates with File Manager to open the PDF file. (7) The File Manager attaches PDF object to the File Share Manager and sends/shares it with the client.

ERROR MANAGEMENT: (1) GUI communicates with the Log Manager 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.