Agile Practices –

During the development of this application we have made some choices to ensure that its development is consistent. Through making these choices we have ensured that there are minimal conflicts during the development and nobody is acting in such a way that is drastically different and compromising the projects development. The aim of including these standards is to make the code appear that it was developed by a single entity in order to ensure the projects development is easy to follow and understand.

The following standards were implemented in the order they appear below. This order was decided to be the most effective flow of development –

* A class diagram was developed as the first stage in the applications development, this allowed the developers to see what data needs to be included. Through producing a class diagram as a group, all members agreed on what must be included, a vital step to ensuring no members had differing opinions on how development should take place, thus ensuring that all developed code was modular, and would work alongside one another. This is important for one of the final stages when merging all contributions. Additionally, the class diagram was updated during the codes development. When changes were needed, all members of the group were consulted to ensure everyone could make the appropriate changes.
* We also agreed on a method of version control. each user would use the same method of version control in order to ensure that all work that is developed is available to each other. Through having all work available as it is completed, we can all merge and work off of each other’s developments. The obvious choice of version control was the UWE GitLab.
* When coding, we all decided on using the same coding standards, including comments, variable naming, and indentation. We decided on the following
  + Comments will be placed above the line of code, as opposed to beside the line, this helps ensure that lines do not become too long, and keeping all code closer to the red line on the right-hand side of the IDE, keeping the code neat and understandable.
  + Variables and methods will be made with the standard camel case standard, keeping the first letter of the first work lower case, and all first letters of following words upper case. Example: “hallNameArea”
  + Indentation will also follow the standard technique of applying the indent whenever a command takes place within another process.
* After the code was completed, we as a group started development on the use case diagram. This was produced after the code due to the fact that as the code was developing, there were many changes to the class diagram. We decided it would be more efficient to produce this after as it would ensure that it is correct and matches the code, and therefor wouldn’t need constant refinement. Additionally, we also did this for the sequence diagram.
* When testing the program, we agreed on the conditions that should be tested we did this in order to ensure that we are only testing the vital features and not testing redundant features. Testing redundant features would simply add unneeded volume to the testing document. These redundant features would include testing features that use the same code, just under a different area of the user interface, this would be un-needed as we are already sure this works and have proven such.
* Additionally, during the testing, we found that validation such as the phone number were not correctly validated. After finding this issue we decided that we will not validate this input due to the complexity. For example, a phone number has many variables, such as if it is a landline or mobile device, additionally it will have an area code. Therefor it may include characters such as a ‘+’, and have a varying length, this means it can’t be stored as an integer and adding character boundaries would be overly complex. Keeping this as a string is the best choice and we do not believe it will cause significant issues.
* We additionally ran jUnit tests. These tests allowed us to implement them directly into the code. These tests were effective because they would run alongside the functions and check the output against the desired output. If the test failed, the test would pick up on this and instead of displaying the usual error message, it would display a more readable, appropriate message. This made diagnosing issues more effective and allowed us to fix them more appropriately. These tests were used during development, while the other tests were used at the end of the program development.