Artifact One: The Emoji App

The artifact that was chosen for this second milestone is the Emoji App that was developed within my Mobile Architecture and Development course. This application uses the built-in android accelerometer sensors, to detect whether the device has been shaken. When the device is shaken, the current emoji that is displayed on the screen is then switched to another. The app has a fun style that is fitting for the theme of the app and showcases the ability to build user-centered mobile applications that take advantages of the many sensors and hardware found within an android device. The enhancements made to this artifact were the most difficult to complete thus far. Originally the app, only had the main activity screen where the user could either shake the phone or press a button to change the emoji. This application, although fun to use, did not exhibit any type of testing that would ensure its stability on the market, so adding android instrumental testing as well as Junit testing was the first enhancement that needed to be made. The second major enhancement that was made to this application was the addition of a new feature. This new feature allows the user to add and store their own images or emojis to the application and then develop more android tests for the new feature, implementing a “test as you build” methodology.

Building the android tests was the most difficult enhancement that I have made thus far. I had never built an android instrumental test and was quickly overwhelmed with the sheer number of available APIS that mock or copy android components so that they can be tested. I had originally tried to use the Mockito and then Roboelectric libraries with little success. Every time, I built a test the compiler would return a null pointer error. So, after watching many tutorials, I was finally able to figure out a way to build robust android instrumental tests that did not rely heavily on the Mockito or Roboelectric libraries and where the Junit and Junit.espresso libraries could be used instead. Once I had finished building a unit test for the Emoji class, I was able to start building tests for the infrastructure. The android tests that were created for this application were heavily focused on layout validation as well as to ensure that the application’s components functioned correctly. Once the main activity’s test was built. I moved on to adding the next application enhancement, the “add your own emoji” feature.

Building and implementing the new “add your own emoji” feature to the application was a straightforward process even though I had never created a program that worked so heavily with image files before. The hardest part of developing this feature was figuring out a way to convert the Drawable type Emojis to strings, so they could be added to the SQLite database and then converting them back to Drawable so that they can be used within the application. Unfortunately, I was unsuccessful in this endeavor and hope to keep working on a solution. A “test as you program” Agile Methodology was implemented while developing this feature. So, with every sub-component of the feature that was developed a new test was created to test it. This followed the two main principles that I had learned in my Software Testing and Automation course as well as in my Software Development Life-Cycle course: “Do not leave testing until the end” and “Build a new test for every component that is added to the program”. Although, the tests for this feature need to be evaluated on a device and not an emulator due to the reliance on the android hardware, every feature was tested as they were implemented.

Lastly, some of the more minor enhancements that were made to the Emoji App also highlights my skills in android app development. The first minor enhancement that added to the application was the background of the application. I exchanged a solid color for a more on-theme emoji wallpaper. The number of emojis originally displayed in the application increased from two to eleven. There was also an enhancement made to the style of the layout, where the original layout was flat the new layout exhibits an elevated frame layout that gives the application a more modern and professional look. A button was also added to the main page of the application to allow the user to travel to the add Emoji activity. Also, navigational enhancements were also made to allow the user to go back to the main page when the “back key” is pressed. The add emoji feature still needs adjusting as I am finding it very difficult to figure out how I would be able to store the images into the SQL database and then turn them back into a drawable. I may have to leave the database idea for this artifact behind as and only use the uploading feature for short term phone memory by using the Intent Bundles built into the application’s activity.

The addition of the android unit tests within this application highlights my ability to not only build fun, user-centered applications but to also thoroughly test them. By following the agile software development methodology of “testing as you build”, also shows my commitment to building stable applications. Unit tests were implemented for every activity within the app as well as for the testing of the database and database functions that were used within the app even though the database feature is not currently in use. The addition of a new features we added in order to build more realistic application tests. However, this was not the original application I was going to use for this project. One of the biggest updates I have made to the original plan was that I was originally going to use the Inventory Management application that I had built to showcase this skill, but I had lost access to the application due to an accident which rendered my computer useless and had to rely solely on programs that I had been saved to an external hard drive or on Git hub. However, the Emoji Application provides the same opportunity to build instrumental android test. Between following the Agile methodology while enhancing the Emoji App and implementing unit tests, the criteria for this artifact enhancement has been met. With the last artifact focusing on data structures and java intermediate programming and this artifact focusing on implementing the Agile Methodology as well as implementing software testing many of the degree outcomes have been met.

Second Artifact: Task Service

The first artifact that I chose to update was a Task Service program that I created in my Software Testing and Automation. This program allowed the user to create tasks and then view or update them at will. The Java Linked List API was used to store, hold, retrieve the accounts. The initial tests that were made for this program were very thorough as they had 99%-line coverage. In order to enhance this project, I decided to implement my own doubly linked list data structure, a user-centered GUI, and implement sorting mechanisms that add to the user’s experience without decreasing the program’s efficiency.

Creating your own data structures is more secure than using an API data structure as it allows the programmer to shore up security vulnerabilities that lurk within the API structure. Use a hand-built data structure is also more secure because it allows the programmer to tailor the data structure to meet their exact needs. These data structures do not have unused methods that are not tested by the programmer where they have the potential to be used to grant access to private structures within the program. Lastly, it is more secure to use a hand-built data structure over an API because any hacker or programmer can view all the API’s methods online. However, when implementing a hand-built structure only the programmer knows what the internal methods are and cannot be easily guessed by the encapsulated user.

The second enhancement that was made to this program was a GUI interface. The GUI interface allowed the user to view their tasks within the GUI’s table and add and delete them, as necessary. This makes using the Task Service easier for the user as they do not have to know how the program works to be able to use it. Implementing a GUI encapsulates the program from the user which also adds another level of security to the program.

Lastly , the final enhancement that was made to the Task Service Program was the ability for the user to sort their tasks by name and then by description. Two added buttons are placed within the GUI, which allows for the user to easily display their tasks alphabetically. Regarding the implementation of the sort methods a hand-built sorting mechanism was built instead of using a Java API library. The sort method within the program, implements two distinct types of sorting algorithms: Quick Sort and Insertion Sort. This was done to enhance the programs efficiency when sorting through considerable number of tasks. Quick Sort has a time complexity of O(logn) in worst case scenarios but a time complexity of O(logn) in best and average case scenarios. The space complexity for Quick Sort is O(n). Insertion Sort has a time complexity of O(n) for best case and O(n2 ) for worst and average cases. This algorithm also has a space complexity of O(1). So, when the program experiences task list that is greater than one hundred, the Task Service program will implement the Insertion Sort algorithm over the Quick Sort algorithm. This will increase the efficiency of the program because Quick Sort’s time complexity of O(logn) grows slower than Insertion Sort’s O(n2 ) within worst case scenarios. Even though Quick Sort has a higher space complexity, the program would need to sacrifice space for speed while sorting through a substantial number of tasks. Due to Insertion Sort’s constant space complexity and it is time complexity of O(n) for best case scenarios, it is faster than Quick Sort for smaller data structures. This was why this strategy implemented when the number of Tasks is under one hundred.

While building this artifact, there were many challenges that I faced. My first challenge was when I switch the program from the Java API structure to my Doubly Linked List. This brought up quite a few errors where they had to be dealt with one-by-one. The biggest hurdle was making sure the logic was correct for the methods within the Doubly Linked List. Every time I thought I had everything converted and working correctly, some other type of logic error would arise, and I would realize that I needed to add a new method or alter my logic. The second biggest challenge I faced was when deciding which sorting mechanism to implement. After studying the available algorithms, it was near impossible to find the perfect algorithm for all situations which is what lead me to implementing two. This helped me ensure that my program remained as efficient as possible when having to sort through large or small datasets.

In conclusion, the enhancements made to the Task Service program highlight my ability to turn the Task Service into a more efficient and secure version. The addition of the GUI also shows my ability to build user friendly programs centered around the user’s experience. These enhancements meet the guidelines of the first artifact as they exhibit skills learned in my Data Structure Class as well as Skills learned in my intermediate Java course. The changes made also meet the course objectives as it is using skills that I learned in my Data Structure class as well as skills I learned within my Intermediate Java class.

Artifact 3: Contact Service App

The last artifact within this project is the Contact Service application. The enhanced version of this application is an android mobile app where the user can add, remove, and update contacts on their device. The original artifact that this application is based on was a simple desktop app that did not have a graphical user interface and was made solely for the purpose of building unit tests. I had originally developed this program while I was taking my Software Testing and Automation course. The original application’s features were the basic CRUD operations: create, read, update, and delete, as well as basic security measures to safeguard against negligent user input. There were a multitude of enhancements made to this project including the addition of an SQLite database, so the contact data is saved after the application closed. This project was included within the portfolio, due to its ability to be enhanced. The original contact application was the absolute bare minimum of what a functioning contact app should be and within this minimalist program was the opportunity to build something great to show off my database skills as well as my secure programming and mobile application skills. I aspire to be a mobile app developer after graduation, so having the chance to take this “bare bones” project and turn it into a power contact android application was an opportunity I could not miss.

One of the first enhancements made to this project was the addition of a graphical user interface, turning the application into an android mobile app. This was not difficult as the project had a decent structure and it had already been thoroughly tested. The next enhancement made to the application was the SQLite database. Following the secure programming protocols, the database was implemented with security in mind. Each of the database’s user methods are package private as to prevent them from being used outside of the database’s package. Every piece of user input is tested before it reaches the database. By utilizing different security approaches in the database’s implementation followed the Defense-in-Depth protocol. This protocol was taught in my Secure Programming course. The next security measure was created to safeguard the application’s database was placing if-statements within the Contact class that specifically checked for potential SQL injections. This was done by checking for the equal sign within each input as well as checking for the word “SELECT”. By implementing these safeguards, a SQL injection that uses the equal sign or the word “SELECT”, will be stopped before it has a chance to reach the database.

The next enhancement that added to the Contact App was the addition of two features. These two new features allowed the user to make phone call or send a text to their contact directly from the application. In order to make a phone call or a text, all the user has to do is click on the desired contact from the list within the main activity, where they are brought to a contact details screen which has a button to make a phone call and another button that triggers a pop-up text window for the user to send a message. Another important security policy that I followed during the implementation of these two features, was the policy of least privilege. This policy is implemented by not garnering unnecessary permissions from the user and only ask for the permissions that are necessary for the application to function. The application does not depend on the permissions to run, the user can retract the permissions at any time and still be able to use the other features of the application.

Lastly, the last major enhancement added to this application was the ability for the user to search for their contacts on the main screen. To find a specific contact within the database, all the user needs to do is to start typing either the contact’s name or phone number within the search bar and any contact that meets the typed criteria will remain in the recycler view list. Implementing features like the search bar enhance the user experience and follow the user-centered approach to application development was taught within my Mobile Architecture course. The user is also able to smoothly navigate back and forth through the application’s pages by either clicking the back button on their phone or but clicking a button within the activity. There is also a separate page to update a contact’s details allowing for the user to update their contacts as necessary.

Nearly every feature within this application relies heavily on the SQLite database, where without the application would be unusable to real-world users. In the original artifact, as soon as the program closed all of the data added to the application was lost, making it unfit for real use. With the addition of the database, the application can be uploaded to the google play store and be used by anyone who would like to download it. Although there are still more features that I would like to add to the program, such as a working menu and the ability to synchronize the application’s database to the device’s built-in contact app. These features would enhance the user’s experience as well as provide the user a secure application to store their contact list.

Although building on the original “bare bones” artifact provided me with a wonderful opportunity, it also meant that enhancing the application would be a lot of work. I am still new to mobile app development and working within android studio, so one of my biggest challenges was configuring my dependencies as well as my manifest. The smallest error in the build.gradle file or in the manifest file can lead to hours of painful debugging causing this application to be my biggest challenge so far. My first challenge in building this application was configuring the build.gradle file. I had unknowingly made a mistake in my configuration of the build file and in attempting to resolve this error, I spent over 6 hours trying to get my program to build correctly. After all of that time, I ended up having to create a new project and move every class, layout, and XML file within my project to a completely new one. From there I was finally able to receive a successful build. Another challenge that I faced in creating this application was creating the pop-up text message window. I struggled to figure out the right approach to building this feature. I had tried to make into a fragment activity, build a layout within the pop-up window API, and build a separate activity for the feature before I stumbled upon the Dialog Builder API. This API allowed me to create a dialog box where the user’s message is entered into the dialog box and sent. Although, I faced multiple challenges in creating this application, I was successful in the end as I met the outcomes set forth for this assignment. An SQLite database was added to a project and where it was implemented in such a way that ensured the security of the database as well as the enhancement of the original artifact.