Rough Script for Extra Credit Project

Preston: Hello, my name is Preston Dotsey and I am here with my partner Pankaj Behera. Today, we are going to show you a somewhat in-depth presentation on our exploration of a framework we discovered which we feel represents and utilizes several of the core concepts we have gone over in class. When looking for a framework to explore, there are tons of options out there that would have made great topics, but the one that really piqued our interest is the Lift framework.

Pankaj: The lift framework is a web application framework written in scala that is meant to help scala web applications written in Scala and Java scale their applications and protect their code and data. The framework is used by a variety of applications ranging from gaming services to educational websites and online text editors. Founded in 2007, Lift has been a widely used and successful open source framework for 15 years.

Preston: What especially caught our eyes about Lift is the way that it exemplifies multiple concepts that we have been learning about over the course of our class. The first example is in how Lift intentionally abstracts data to increase the security that their framework provides. It accomplishes this mainly through the way it handles the HTTP request/response cycle. Instead of making the developer handle this cycle on their own code using object wrappers around the HTTP request, Lift handles the request entirely on their side. When a user makes a request, the Lift framework instead represents the action as a GUID in the browser and a function contained within the server. When the GUID is sent as part of the HTTP request, the function is then called with the supplied parameters in the server. By removing the wrappers from the request, data is abstracted and any attempts to attack the process externally are made much more complicated. Because GUIDs are session specific and generated on the spot, attacks such as replay attacks and parameter tampering attacks become orders of magnitude more difficult to accomplish. This use of abstraction ties in directly with the abstraction we have used in class. While Lift abstracts HTTP requests, our labs have abstracted test cases and helper function details. While they have implemented abstraction to help prevent against attacks, the use of abstraction in class has been to increase clarity of code used and to encourage us to debug and think critically to pass test cases which we cannot explicitly see. Through abstraction, multiple goals are accomplished.

Pankaj: Another way in which Lift exemplifies concepts from our class is the use of higher-order functions within it. A higher order function is a function which takes one or more functions as its parameters or returns a function as its result after execution.In class we have used such functions in our labs, such as map and flatMap. When using such functions, we are treating the function within the parameters as values themselves, which are dynamically typed depending on the input of the function parameters themselves. This concept is used quite heavily in the Lift framework in how they abstract data types and objects. In Lift for example, let’s say we are building a web application which allows users to create login ids and passwords. If we want to collect our user data, we can use higher order functions within the Lift framework, such as map, to build a list of user names from our list of user ID’s. The use of higher order functions is vital to

the success of the Lift framework as it allows developers to create more dynamically typed applications and helps facilitate the abstraction of data within the application, as well encapsulates functions to create more loosely coupled code.

Preston: Drawing off of our discussion of the use of higher order functions within the Lift framework, a closely related concept which also presents itself in the framework is the idea of functions as values. We have already discussed how in higher order functions, the parameters of the function are themselves, one or more functions. Now why does this work? In order for a function to execute, we must supply it with values to act on. Now normally we can supply it with any number of value types (string, int, double, etc.) on which to act, so it stands to reason at a basic level that if a function accepts a function as a parameter, then the function within the parameters is itself being treated as a value. It is through this kind of extrapolation that we can see that functions themselves are values, as the values they return are their core characteristic. The use of Javascripts makes heavy use of this fact as these functions as values are repeatedly used in higher order functions. Look at the example we have here:

```
function prefixWordWithUnderscore(word) {
  return `_${word}`
}

const words = ['coffee', 'apple', 'orange', 'phone', 'starbucks']

const prefixedWords = words.map(prefixWordWithUnderscore)
```

In this code, we are using the function prefixWordWithUnderscore as a value in the parameter for the higher order function map. By treating the function itself as a value within the map function, we are able to create cleaner code and allow for variable changes to be dynamically typed.

Preston/Pankaj: Go over code examples and draw parallels

Preston/Pankaj: As a final thought, we hope that we have shown you how the Lift framework incorporates the ideas of abstraction, higher-order functions and functions as values. Through the implementation and incorporation of these elements, Lift presents itself as a powerful yet lightweight framework for web application development.