Play with Scala

Functional Programming Course

Martin Zuber, B00066378

Project Supervisor: Luke Raeside

April 13, 2016

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Degree of Honours B.Sc. in Computer Science in the Institute of Technology Blanchardstown, is entirely my own work except where otherwise stated, and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Acknowledgements

In performing my assignment, I had to take the help and guideline of some respected persons, who deserve my greatest gratitude. The completion of this assignment gives me much Pleasure. I would like to show my gratitude to the project supervisor Dr Luke Raeside, Institute of Technology, Blanchardstown for giving me good guidelines for the thesis throughout numerous consultations. I would also like to expand my deepest gratitude to all those who have directly and indirectly guided me in writing this thesis, especially Dr Markus Hofmann.

Abstract

This paper is dedicated to computer programming with the focus on programming styles found in modern application development. The idea for the project came to life when I, the author and computer science student realized there are programming styles other than imperative and object-oriented programming. Namely functional programming paradigm. The lack of a study material dedicated to the functional programming withing the ITB Computer Science course curriculum and my curiosity toward the subject I decided to research and to learn the paradigm. I decided to use the course final project as the vehicle for this endeavour.

This paper is discussing functional programming paradigm with Scala programming language and web application development process using Play! Framework. The target audience is any computer science student with knowledge of a programming language such as Java and with an understanding of some of the object-oriented programming principles such as inheritance and polymorphisms.

Please keep in mind that I had no knowledge of a functional programming, Scala programming language or Play! framework prior to this endeavour. This document is basically a report of a student learning basics of these concepts and tools. I can only hope that I would be able to communicate clearly what I learn and in this document's usefulness.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Introduction

In the world of multi-core processors, distributed systems and big data, we are witnessing a paradigm shift in the computational model used within application development industry. It's a shift from imperative programming style toward declarative programming and the rise of the functional programming languages [1]. JVM ecosystem is no exception.

Even though Java being still most used JVM language and one of the most used programming languages overall, JVM languages such as Scala, Groovy and Clojure are getting more popular every day. Java itself with recent SE 8 update brought limited support for functional programming as well.

Scala programming language [2] is getting a big momentum in the last few years [3]. Companies such as Twitter or LinkedIn [4] had switched to Scala for their web applications development. Scala is being a big player in the realm of Reactive

programming, Distributed Systems and Big Data [5].

1.1.1   Some of the reasons for Scala success

- Scala is elegant, scalable, purely object-oriented and fully functional programming language with the performance comparable to Java.

- Scala compiles to JVM byte code .class files which mean that Scala is fully compatible with existing Java libraries or frameworks and vice versa. Mixed Java / Scala projects are quite common.

- Existence of powerful, open source and award-winning frameworks [6], toolkits, platforms and build tools written in Scala, such as Akka, Spark, Play, SBT [7].

1.2   Aims and Objectives

The idea is to develop a web application for a computer science student who would like to learn Scala programming language and functional programming paradigm. The application will be developed using MVC Play! Framework. I decided to work on this project for the following reasons:

- Strong personal interest in functional programming.

- Lack of modules dedicated to functional programming in ITB curriculum.

- Solid background in Java platform.

- Desire to pursue a Java / Scala development professional career.

1.2.1   Web Application Proposed functionality

- A student can register the account.

- The authenticated student will gain access to the functional programming with Scala lectures and exercises.

- The student will have access to Scala interpreter where he could carry out coding exercises.

- The student will be able to communicate with other users through a chat window or other messaging functionality.

- The student will be able to upload the files, for example an exercise solution.

1.3   Main Research Questions

- What exactly is functional programming and what benefits does it bring to application development?

- In what ways Scala supports the paradigm?

- What exactly is Play framework and what kind of applications we can develop with it?

- What are the advantages and disadvantages of Play framework in terms of productivity, performance and scalability and in comparison to other frameworks?

1.4   Justifications / Benefits

As mentioned earlier, there is not a module dedicated to functional programming in ITB Computer Science course curriculum. I can actually claim that there is no

mention of this programming style in any module overall and the paradigm is totally ignored by the curriculum. In comparison, there are 8 networking modules delivered to students over the course duration. In my opinion, this makes the course unbalanced and creates an 'educational gap' in the curriculum. Personally, I feel that the topic belongs to the course curriculum and it is the natural progression from object-oriented programming and design patterns which had strong coverage in the third year of the course.

- **Filling the 'educational gap' in the Computer Science course curriculum by creating learning web resource.**

The preliminary research on the current state of the application development industry is suggesting that fluency in a functional programming language is a really valuable skill for any developer to possess right now. Developers able to code using languages such as Scala, Clojure, F# or Lisp dialects are in demand. Reasoning about the functional programs requires different 'mind-set' or 'thought processes' in comparison with object-oriented programming. Even if one would never use purely functional programming professionally, understanding of the concepts will make one a better programmer overall and give one deeper insight into other programming styles.

- **Acquiring valuable skills and deeper insides into different programming paradigms and design patterns.**

Also, Scala seems to be very elegant language and takes the good design ideas from many other programming languages. The language was designed with the scalability of the syntax taken in the consideration, which allows the language to 'shape-shift' towards the needs of its users. In my opinion, the Scala programming language would be the right tool for me to learn functional programming paradigm.

- **Learning Scala programming language and Java Virtual Machine (JVM).**

Even though learning functional programming with Scala is the main aim of this project, the additional work will be carried out by exploring Play framework and developing the web application. The preliminary research suggests the framework is a powerful tool for rapid web development. Play seems to be heavily inspired by Ruby on Rails. It supports reactive web application development and seems to be a cleaner alternative to legacy Java Enterprise stack [8].

- **Building valuable skill set in contemporary web application development.**

1.5   Feasibility

Project requirements

- Access to relevant study material and tutorials dedicated to Scala programming language and Play. There is no material available in ITB library, but there is a number of books published and available over the internet. *Estimated cost is 100 euros.*
- Access to personal computer. I'm the owner of the laptop computer, which should serve as the project workstation without too many constraints. Additional research must be done to decide on the operating system and other tools. Using free software is most desired.*Estimated cost is 0 euros.*
- Access to the internet, deployment, DNS and versioning control services, printer and thesis binding. *Estimated cost is 300 euro.*

- At least 14 hours per week to carry out the project related work for the whole project duration. This estimation is based on personal experience I gained by working on two academic projects during my studies and on experience working on a research project as part of the summer internship.

In my opinion, all the requirements could be met to assure this project feasibility with the minimal cost associated. With proper planning, use of the personal assets and free software when possible, I should be able to meet the project goals within the given time scale.

1.6   Proposed Methodologies

1.6.1   Literature Review

The first step is to carry out the research on functional programming [9], Scala programming language [10], MVC design pattern and Play Framework [11]. I'm planning to use dedicated books I acquired recently, plus online tutorials and blogs. I will analyze, summarize, and carry out the coding tutorials. The thorough literature review at the beginning should lay a foundation for following web application development. I will continue to research relevant materials for the whole duration of the project.

1.6.2   The Web Application Development

The real challenge lies in the fact that I have a little experience developing web applications using MVC framework and no skills in technologies I decided to use during this project. I have only general ideas how to design such an application. Only

layers I can design reasonably well in advance are the database relations and user interface.

Therefore, I must approach this problem using some kind of adaptive methodology and avoid 'a big design in advance' approach. I would argue that Prototyping SDLC is the best methodology to carry out the development. In this model, the developer basically re-analyzes, re-designs, re-implements and re-test application prototype until the product is accepted by the client. Instead of creating 'a big design in advance',
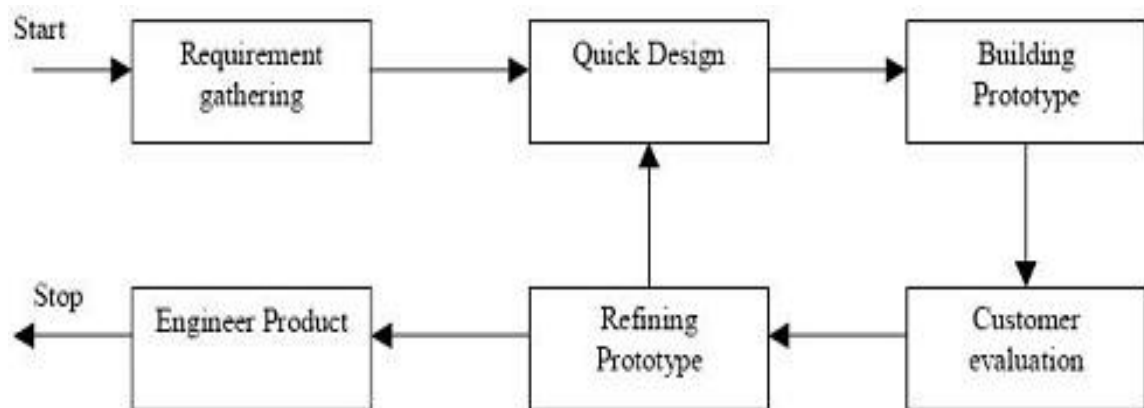
Figure 1.1: *Prototyping SDLC diagram*

prototyping methodology is allowing the developer to change the design with each iteration of the design / development / evaluation / refining circle. I should expect major changes in design based on my lack of experience in web development and technologies I decided to use. The prototyping methodology will allow me to have a simple, but functioning prototype reasonably early. As more insights are acquired from the research, the prototype can be re-designed to implement additional functionality or re-evaluate the design.

1.7   Project Plan

Work Breakdown Structure (WBS)

1. **Research.**

    (a) Research on Scala programming language design ideas and syntax, MVC design pattern, Play framework, user interface and database design.

    (b) Literature Review of papers dedicated to the functional programming paradigm and test driven rapid web application development with Play.

    (c) Building development platform.

2. **Front-End design.**

    (a) Creating wire-frame design for new view.

    (b) Constructing the view.

3. **Database Design.**

    (a) Adding the database entity for new view.

    (b) Designing the relationships with existing entities.

4. **Prototype Development.**

    (a) Creating the controller object.

    (b) Creating unit tests.

    (c) Implementing method bodies using test driven development methodology.

    (d) Repeating the steps 1, 2, 3 with additional functionality until the final product is build.

5. **Quality Assurance.**

    (a) Performing integration tests of the whole MVC pipeline.

(b) Performing user tests.

6. **Post Implementation Maintenance.**

(a) Making prototype production ready and deploying the prototype.

(b) Monitoring deployed application and adding more content.

(c) Adding content to the application.

(d) Working on project Report.

(e) Repeating steps 5.(a), 5.(b), 6.(a), 6.(b), 6.(c), 6.(d)

(f) Report Binding.
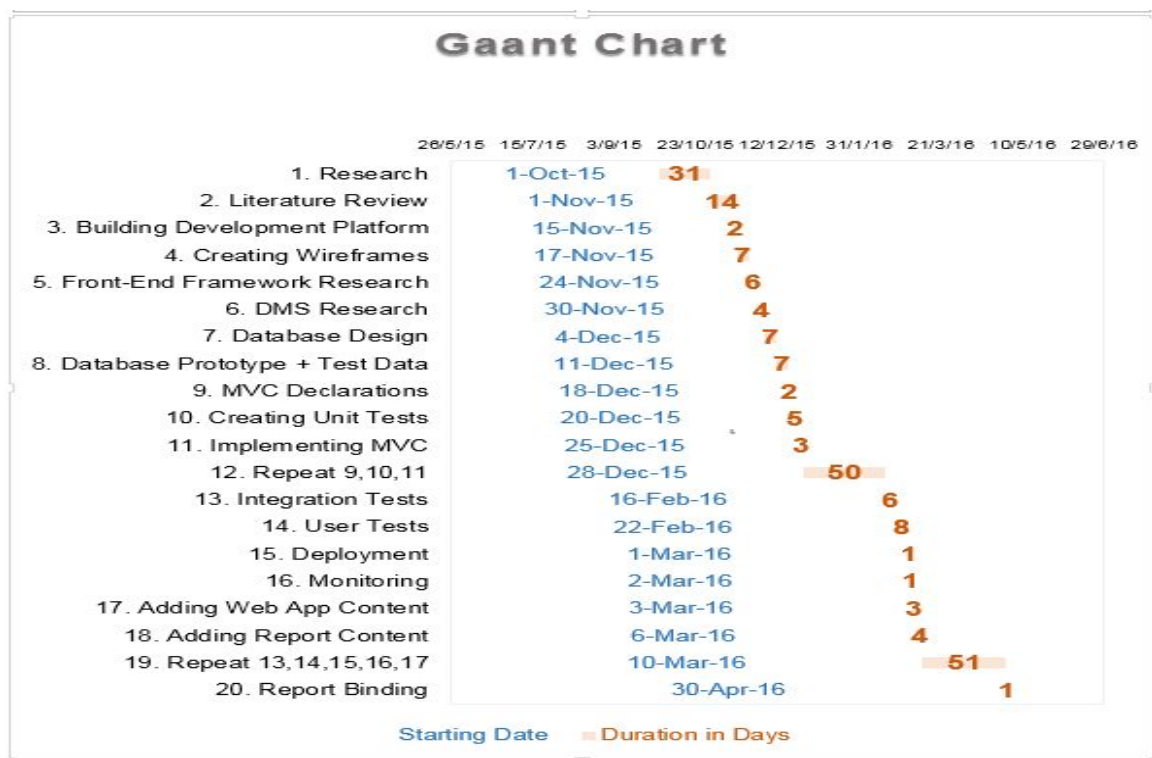
Figure 1.2: *Gaant Chart*

1.8   Expected Results

The project would be considered successful if it would meet at least two following criteria and goals. In the case of three or more goals accomplished I would consider project outcome to be very successful.

1. The gain in Scala coding skills and creation of enough learning material content for Scala course web application.
2. The comprehensive research on the functional programming paradigm and design patterns performed and gained the ability to code in the declarative style.
3. The delivery of fully functioning prototype of Scala course learning web application.
4. The acquisition of skill-set in rapid web development using Play framework.

1.9   Conclusion

At the end of this project, I will hopefully introduce some of Scala dedicated learning material in the form of an interactive web application. It can be used in a fictional course module for a fictional college. The application main purpose is to promote the interest in Scala programming language and functional programming paradigm. Especially on the personal level. After three years of my studies with ITB, I didn't learn anything about functional programming. I chose this project so I can gain a knowledge and skills which are really missing in my skill-set and which could prove very useful in the future. I hope as a fourth year student I am ready to face this challenge.

This research project will not contribute at all to the discipline area. Maybe

only in a sense that it will hopefully bring one more student with the passion for programming languages to the functional programming paradigm. And maybe if other students or lecturers will see how elegant and declarative functional programming really is, the contribution could be a bit more significant.

Chapter 2

Literature Review

2.1 Introduction

Computer programming is a fascinating and vast subject to study. Programming languages began as an attempt to translate the human language and the way of human thinking into the language of a computer.

The idea was to make it easier, more efficient for programmers to write the programs. With the evolution of computers, as the problems to solve became more complex, the programs became larger and more sophisticated. The programming languages quickly evolved into the forms commonly referred to as high-level programming languages. In these forms, the programming languages are hiding the internal hardware details and offer a higher level of abstractions allowing programmers to write the programs using familiar terms with an ability to model real life objects.

The purpose of this paper is to review some of the studies dedicated to concepts and design ideas behind the high-level programming languages. The research done in this

field is immense. It is out of the scope of this paper to cover every concept,paradigm, or language and rather study some of the most important concepts in general. Then I will continue to study research papers dedicated to two most popular paradigms - object-oriented and functional programming. I will try to identify and discuss their key concepts, strength, and weaknesses and outline their history in short. I chose reading materials with a secondary intention to lay down a foundation for further studies toward a deeper understanding of the paradigms and transitions between them.

2.2   General Concepts

In *The Conception, Evolution, and Application of Functional Programming Languages*[12] author defined a programming paradigm as an approach to a computer programming based on a coherent set of principles or a mathematical theory. The purpose for a paradigm existence is to solve a specific type of a problem. Each paradigm consists of a number of concepts. Any programming language can support one or more paradigms and the language which support more than one paradigm is called the multi-paradigm programming language. Different languages can interpret the concepts of a paradigm differently and often the implementations differ from language to language as well. The author listed around 30 useful programming paradigms implemented by modern programming languages. Author then identified the two most important properties which differentiate the programming paradigms:

1. *Observable nondeterminism* is when a program is not completely determined by its specification. In other words with each execution the same program can produce different results. Observable nondeterminism is caused by the run-time scheduler and its usual effect is a race condition, which is often used as the synonym for Observable nondeterminism.

2. *Named state* is an ability of a program to store values in time. This ability is highly influenced by the paradigm it contains it.

As mentioned before, the programming paradigms are based on a number of concepts. Author identified four most important programming concepts:

1. **Record** is a data-structure. Every programming language should be able to work with records. Arrays, lists, strings, trees and hash tables are derived from records.

2. **Lexically scoped closure** is a record storing a function together with an environment referenced at the time of that function definition. Closure is a very
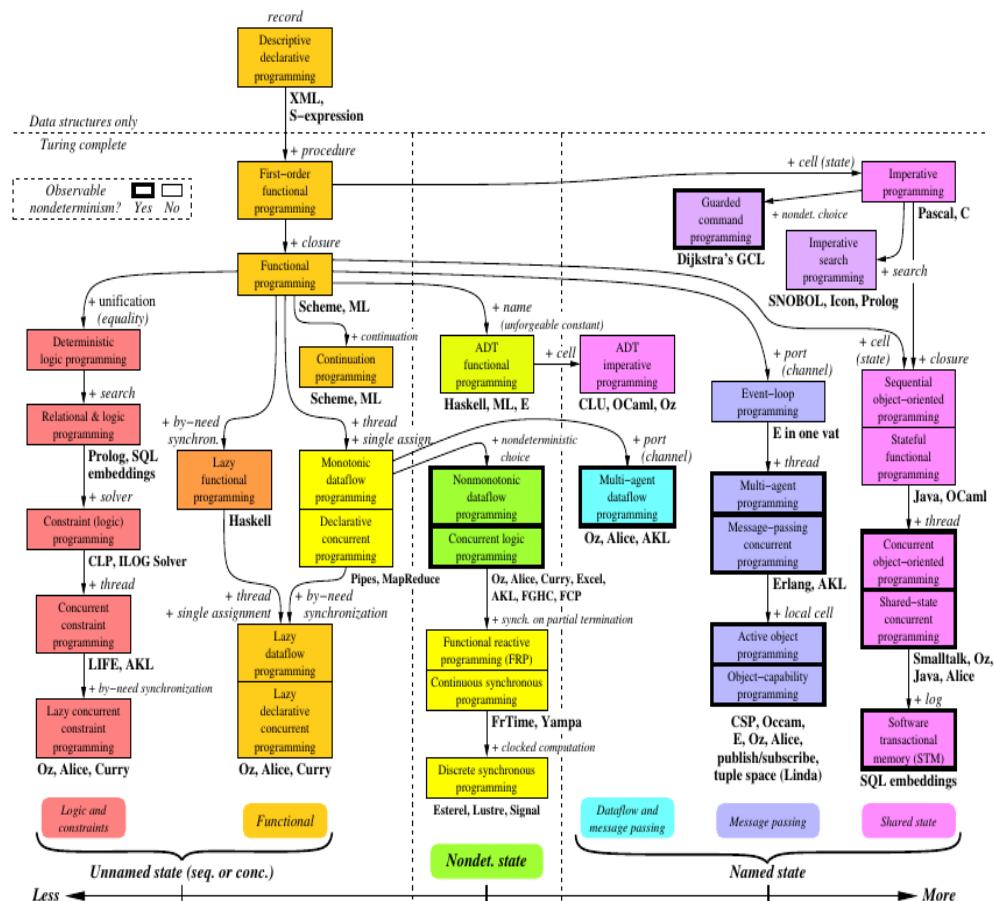

Figure 2.1: *Taxonomy of programming paradigms (Van Roy, page 15)*

powerful concept and constructs such as *objects* or *control structures* have been implemented with closures in many programming languages.

3. **Independence** is when a program is constructed from independent parts. When parts don't interact with each other we call them *concurrent.* When an order of the execution is given, the parts are called sequential. The interaction between the parts is called communication. Author recognizes three levels of concurrency:

   (a) *Distributed system,* where concurrent activities (parts) are computers.

   (b) *Operating system,* where concurrent activities are processes. This level of concurrency is often called competitive concurrency because the processes are competing for access to system resources.

   (c) *Inter Process,* where concurrent activities are threads. This level of concurrency is called *cooperative concurrency* because the threads are cooperating to achieve the result of the process. There are two popular paradigms for inter process concurrency: *Shared state concurrency*, where shared data are accessed by threads using control structures called *monitors.* The second approach is *message-passing concurrency*, where threads are communicating by sending messages to each other.

4. **Named State** Author showed how named state could be implemented with a help of internal memory (variables).

2.3   Specific Characteristics

2.3.1   Object Oriented Programming

In *A Survey of Object Oriented Programming Languages*[13] authors argued that consensus on what key concepts of Object Oriented Programming still don't exist.

They argue that the most fundamental concepts behind the paradigm are as follows:

1. **Class** is a mechanism which group together attributes and methods with common properties. The class describes the run-time behavior of the objects instantiated from it. The well-designed class would have an immutable interface clients can use.

2. **Abstraction** is a simplified view of reality. Presented to clients by class methods and attributes.

3. **Inheritance** is a mechanism to create hierarchical class designs by creating a child class of the original class. A child class inherits the parent class behavior which can be then extended. Multiple inheritance allows for a class to inherit the behavior of multiple parent classes.

4. **Encapsulation** is hiding implementation details within the class. Only the interface is presented to the clients.

5. **Polymorphism** is allowing to similar looking structures to handle a variety of objects.

Authors then discussed inheritance and polymorphism in detail. I will outline the important key points.

Inheritance

As mentioned earlier the inheritance brings the hierarchical relationship into the class model. Authors pointed out that many languages define the most generic class that is an ancestor for all the classes in the language. This applies that any class can be downcast to the pointer of that ancestor. Authors pointed out that inheritance provides the ability to represent an *"is a"* relationship in software. This relationship

can be violated if child class extends the inherited code the way which changes the code semantics. It is the reason for the inheritance to be used with caution.

Inheritance also allows to represent generalization /specialization relationships with method redefinition. This functionality is breaking the encapsulation because the child class has access to parent class hidden methods and attributes. This problem can be addressed by defining a well-defined interface for the descendants. Authors argued that literary every object-oriented language provides the ability for a child class to invoke parent class methods. Even those methods which were redefined in the child class.

Some of the languages support the feature which restrict method redefinition. A method is marked *'frozen'* or *'final'* and no child class can redefine this method. Authors warned that inheritance has also an adverse effect on synchronization requirements of a concurrent object. This problem is usually named *'inheritance anomaly'*. The problem arises when a class with concurrent code is derived. The careful redefinition of the inherited methods is necessitated to preserve the synchronization requirements. This necessity denies *'reuse'* benefits of inheritance. The multiple inheritance potential risks were then discussed in details. Authors explained various problems which could arise such as directed acyclic graph in a class hierarchy, method name collisions or repeated inheritance problem.

Authors outlined various solutions to deal with the problems and outlined the alternatives to multiple inheritance used by some of the object-oriented programming languages such as interfaces, mixins or delegation.

Polymorphism

As mentioned earlier, polymorphism allows programmers to write functions and classes which work uniformly with different types. Authors listed four distinct types of polymorphisms grouped into two categories. *Ad-hoc Polymorphism* and *Universal Polymorphism.* The difference between the categories is based on the fact that ad-hoc polymorphic functions execute code only for a small set of potentially unrelated types while universal polymorphic functions execute the same code for an infinite number of types.

Authors then defined the two types of ad-hoc polymorphism: The first type is *Overloading Polymorphism*, where the polymorphic function has the same name, but a different signature. Within this type authors distinguished between a *method* overloading and *operator* overloading and discussed each in detail.

The second type of ad-hoc polymorphism is *Coercion Polymorphism.* With coercion, the value of an argument can be converted to the value of another argument from the list of a method arguments. Authors pointed out that the difference between those two of ad-hoc polymorphism types is often blurry, especially in untyped and interpreted programming languages. The universal polymorphism is divided to two kinds as well. The first kind is *Parametric Polymorphism* aka generic programming. Generic programming uses type parameters to determine the type of a method argument. Authors pointed out that parametric polymorphism is only relevant for statically typed languages because dynamically typed languages infer types at run-time and hence have generic programming built into them. The second kind is *Inclusion Polymorphism*, which gives different classes the ability to handle the same functionality.

Inclusion Polymorphism is what we call *Inheritance* in object oriented programming. Inclusion polymorphism is implemented through *dynamic binding* or sometimes

called *late binding* because a method is bind to the message at the run-time. It is relevant for situations when a child class has an overridden method of the parent class and it is not obvious which method is being invoked. The search for the right method is then performed by the compiler (interpreter) at run-time. The dynamic binding uses the most specific version of a method.

Authors mentioned that some of the languages implement static binding where methods are bind to messages at compile time and it will always bind to a base class method version.

2.3.2   Functional Programming

In *The Conception, Evolution, and Application of Functional Programming Languages*[12] the author distinguished four key characteristics of modern functional programming languages.

Higher-Order Functions

In functional programming, the functions are treated as *'first class values'*, which means that they can be stored in data structures, passed as other function arguments and returned as results. The author pointed out that the function is the primary abstraction mechanism over values. He showed with examples how to compose higher-order functions.

Lazy Evaluation

Often called Non-Strict Semantics or call-by-need. Its primary feature is that arguments in a function call are evaluated at most once. In some cases, it does no evaluation at all. The author explained with the help of an example how lazy evaluation frees a programmer from concerns about evaluation order and pointed out the ability of the lazy evaluation to compute with infinite data-structures.

Data Abstraction Mechanisms

The author pointed out that a data abstraction improves modularity, security and clarity of programs. He explained that modularity is improved because one can abstract away from implementation. Data abstractions prohibit interface violations which improves security and that programs are clearer because of the self-documenting quality of the data abstractions. He argues that *strong static* typing eliminates type violations and run-time errors. He continued the discussion by describing *algebraic* (concrete) data types, *type synonyms*, and *abstract data types*.

Pattern Matching

The lack of side-effects in functional programs allows to apply pattern matching or sometimes called equational reasoning. The author explained the basics behind the feature with the help of code examples and outlined some of pattern matching shortcomings.

In *Why Functional Programming Matters*[14] author summarizes functional programming characteristics and advantages as they are usually used in the literature as follows:

- The functional program consists of functions.

- The functional program uses no mutable variables which, in general guarantee that program contains no side-effects. A program without side-effect is free of a major source of bugs.

- Since a function call produces no side-effects and for given arguments produces the same calculation result independently on when it is evaluated, the order of execution is irrelevant and functional programs are referentially transparent. This freedom makes functional programs easier to reason about.

The author argues that this often used list of strengths of functional programs is describing what functional programming is not (no assignments, no side-effects, no flow of control) and fails to emphasize what functional programming actually is. The list fails to emphasize the modularity as probably the most powerful characteristic and advantage of functional programs. He convincingly argues the ability of higher-order functions and lazy evaluation to increase the modularity by serving as a 'glue' for the program fragments. He provides a number of examples with code to support his claims.

2.4   Short History

First generally accepted object-oriented programming language is Simula (1967). With the introduction of Smalltalk (1962-1980) the paradigm gained some momentum. Most of the concepts of object-oriented programming were implemented in Smalltalk. In early 1980 the concepts were integrated into C programming language and resulting language was called C++. In the 1990s, the similar language was developed called Java by Sun Microsystems. Java became soon one of the most popular object-oriented

languages. Then in 2000, Microsoft announced .NET platform and C# programming language. C# is in many respects similar to Java. [13]

Functional Programming is heavily influenced by lambda calculus invented by Alonzo Church in 1936. First of the programming languages implementing lambda calculus was Lisp specified in 1958. Next significant language in terms of contributions to functional paradigm was Iswim introduced by Peter Landin in 1966. Probably the first functional language which received wide-spread attention was FP specified in 1978. In mid 70's several research projects related to functional programming emerged in the UK. Specifically the work of Gordon, Milner, and Wadsworth. They developed ML programming language which brought the invention of the type system (Hindley-Milner Type System). In early 80's David Turner at the University of Kent developed three languages which most faithfully characterize "modern school" of functional programming: SASL, KRC and Miranda. [12]

In later 1990s and after 2000 the functional programming has gained a great momentum and penetrated mainstream programming. Haskell, F#, Clojure, Scala are some of the examples of functional programming languages. Many object-oriented languages added functional features and become multi-paradigm languages. Java, Python, C# are examples of such.

2.5   Interesting Ideas for Further Study

2.5.1   A Definitive Programming Language

In *Programming Paradigms for Dummies: What Every Programmer Should Know*[16] the author is presenting four research projects, each trying to solve a very

different problem, but all four project considered language design as a key factor to achieve success. Turned out that programming languages invented in each project have very similar structure supporting same paradigms: strict functional programming, declarative concurrency, asynchronous message passing and global named state. The invented languages are Erlang, E, Distributed Oz and Didactic Oz. One could infer ideas for a design of 'perfect language' and it could be quite interesting to study those four programming languages.

2.5.2   Artificial Intelligence

In [14] the author of the paper is using examples of composing programs from lazy evaluated higher-order functions. The final example is the alpha-beta heuristic algorithm. This algorithm is often used in computer games to estimate how good a player position is in the game. The author used Miranda programming language syntax, but it would be very interesting to implement the algorithm in some other functional programming language such as Scala or Haskell.

2.5.3   LambdaFicator

Java 8 update brought a few functional programming features in the language. Namely functions as first class values, lambda expressions and closures. Also fluent Stream API which uses monads, lazy evaluation, and higher-order functions. In *Crossing the Gap from Imperative to Functional, Programming through Refactoring*[15] authors presented the analysis, design, implementation and evaluation of LAMBDAFI-CATOR, the automatic refactoring tool, which converts old-style code prior to Java 8 update into the functional style. Namely tool does two refactorings:

- Anonymous inner classes to lambda expressions

- External iterators to internal iterators (from *for loops* to Stream API *higher order function chains*)

In the paper, authors discussed their motivations, outlined the implementation algorithm in the detail and determined the usefulness with a thorough evaluation. They applied the tool to four open source projects (ANTLRWorks, Apache Ivy, Junit, Hadoop) with very successful results. For example, the first type of refactoring reduced the number of source code lines by 2213 with 100 percent accuracy.

The tool is open source and available for download and would be very interesting to study it deeper and implement it using some other programming language such as Scala or Haskell.

2.6   Conclusion

The main purpose of this research was to study two most popular programming paradigms: object-oriented and functional programming. To identify and understand the key concepts and to find the similarities and differences. The first step was to understand general programming concepts such as record, closure, state and concurrency. Then I focused on studying characteristics of object-oriented programming languages. I learned that main building blocks of object-oriented programs are classes organized into hierarchies based on inheritance. I learned about polymorphism, the powerful feature of programming languages in general. Then I studied papers dedicated to functional programming and I found out that functional programs are built from functions, which are then composed together to modules and programs. I was shown the elegance of lazy evaluation and pattern matching. I learned that functional

programs are trying to eliminate state and to stay referentially transparent.

Chapter 3

Analysis and Design

3.1 Introduction

In this section, I will discuss the approach to the content and web application design. The main aim of this project is to explore functional programming methodologies, develop a learning material content dedicated to the functional programming with Scala programming language. The secondary aim is to develop a simple web application which would deliver the learning material to a potential consumer.

3.2 Proposed Methodology

3.2.1 Learning Material

The lectures will have a form of static text with code examples and coding exercises, The references to cited paragraphs and the source study material will be included. There will be a link to online Scala REPL[18] available in each lecture,

offering to a student an environment to test the code examples and carry out the exercises. Lectures are proposed to be short, covering only one concept or a feature at the time. Lectures will be formatted using Markdown. Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML)[20].

Scala is an elegant but complex programming language. I realized that to be able to study functional programming with Scala, the student will need to learn the language syntax first. Even though Scala programs resemble Java programs in many ways and they can seamlessly interact with code written in Java[17], it doesn't mean that Scala is just Java with slightly different syntax. In contrary, Scala is purely object-oriented, has a different object model, a richer type system with the type inference, supports multi-inheritance through mixins, pattern matching, implicit references, partial function applications, operator overloa and much more. Basically, Scala is much richer language than Java and considered by many to be the most evolved programming language.

Some of these concepts and features must be understood before a study of functional programming can be carried out. Therefore the student must have an access to the learning material dedicated to the Scala language syntax itself. I have decided to produce a batch of *'From Java to Scala'* lectures in outlined form. These lectures will attempt to explain the concepts using the contrast between the languages with the help of code examples in both languages. Meaning a student familiar with Java should be able to learn Scala by comparing the code solutions.

The second batch of lectures will cover the functional programming paradigm using Scala programming language in proposed form. These lectures will not cover the

Scala syntax, but rather how to design functional programs and how to reason about them. Coding examples will be written in Scala and exercises will have a form of short Scala programs partially finished with missing parts of the code to be designed by the student.

3.2.2   Web Application

Secondary aim of this project is to develop a simple web application which will publish the learning material over the internet to the potential students. Idea is to develop this web application using Scala and functional programming style. This could be quite a challenge, because I must first learn and language and the style, which is actually the main aim of this project. If I fail to grasp the concepts I will be not able to build the application. I chose Play Framework to carry out the implementation of the application. **Play Framework Key Features Overview**[11]
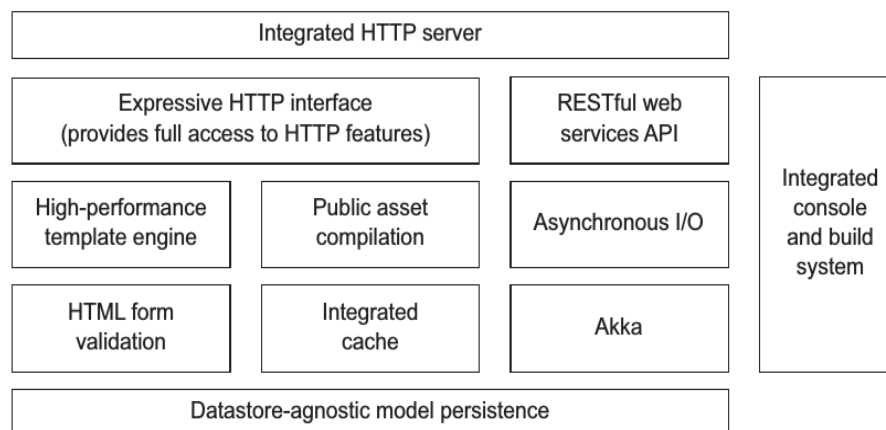

Figure 3.1: *Play Framework Stack*

- Declarative application URL scheme configuration

- Type-safe mapping from HTTP to Scala API

- Type-safe templates

- Embraces HTML5

- Live code changes when you reload the page in your web browser

- Full-stack framework (fig. 3.1), including persistence, security and international-ization

Play versus Java EE From fig. 3.2 is apparent how the framework embraces the
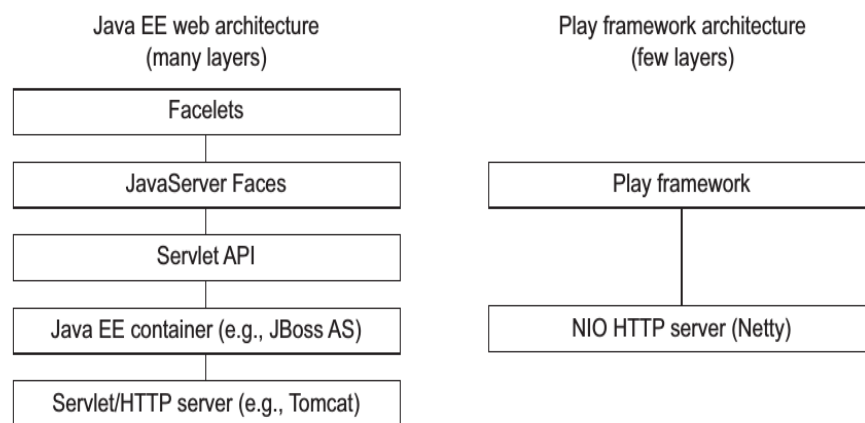

Figure 3.2: *Java EE 'lasagna' architecture versus Play architecture*

simplicity in comparison with layered Java Enterprise Edition architecture. Whole framework stack is build on top of Netty[19] client-server framework and doesn't require an application server container. Netty assures high performance of the stack and simple deployment of the application. Play has both, Java and Scala APIs (Application-Programming Interface) which will allow me use functional programming with Scala. I will discuss the framework in more depth in Back End Desing section 3.8 and Implementation chapter 4.

SDLC - Prototyping Model

As discussed in Proposed Methodologies for The Web Application Development section 1.6.2, I will use Prototyping Software Development Life-Cycle Model to develop the web application. Each step in development cycle is already listed and explained

in Work Breakdown Structure section 1.7 of Project Plan.

3.3   Proposed Tools

3.3.1   Hardware

The project will be developed on DELL Latitude E5540 laptop machine lent to me by School of Computer Science, UCD where I work part-time as a research assistant. The laptop has following specification and should be sufficient to support this project development.

- **Procesor:** Intel Core i5-4310 CPU 2.00GHz x 4
- **Memory:** 8GB RAM
- **Graphics:** Intel Haswell Mobile
- **Hard Disk:** 463 GB

3.3.2   Software

I decided to use an open source, or at least free software tools. The thesis report will be written in LaTeX instead of an office suite. Learning material will be written using the Markdown format, which require just a simple text editor. For the web application development I will need Java and Scala Development Kits with an IDE Studio. Additional tools will be needed to draw diagrams, take and edit screen shots, CSS, JSON or XML parsers and validation tools. For that I can utilize the operating system tools or online tools.

- **Operating System:** Ubuntu 14.04 LTS 64-bit[21]

- **Word Processing:** LaTeX suite witt Gummi simple LaTeX editor[22]

- **Markdown editor:** Remarkable[23]

- **IDE:** IntelliJ IDEA[24]

- **Web Development Framework:** Play Framework 2.4[28]

- **Distributed revision control system:** Git[25]

3.3.3   Deployment

Play Framework has very good deployment model. The application doesn't require a Java application server container, which greatly simplifies the deployment process in comparison to Java EE applications. There are various deployment options available, for example the application can be deployed as a standalone package and just copied to the deployment machine. Some of the cloud providers have built-in support for Play application deployment. Additional dependencies are handled by Simple Build Tool (SBT)[29] which is included with the framework. For example, during the development, the framework is utilizing memory H2 database, but for the deployment the database management system can be replaced with MySQL for example. All need to be done is to edit the web application configuration file and SBT will handle the rest automatically.

- **Heroku:** The Heroku Cedar stack natively supports Play framework applications. A Postgres database is automatically provisioned for Play framework applications.[26]

- **Azure:** Microsoft Azure Cloud supports Play applications deployment on a web server with *Azure Toolkit for Eclipse*. Another option is to dedicate a Vitual Machine (VM) to the application. Azure has numerous Linux images available, Ubuntu Server 14.04 included. The deployment should be then a simple question

of moving the packaged application to the VM after all the required technologies are installed on the VM, such as Java and Scala runtime and database server.[27]

3.4   Web Application Content Design

In preparation for learning material writing, I collected some of the guidelines and recommendations of ways to structure the learning content with some tips on the writing style. Following material is only a short extract from the research conducted on numerous online resources dedicated to e-learning.

3.4.1   The Lecture Structure

- **Introduction**
    - Title
    - Lecture content list
    - Requirements (previous work or expected level of knowledge)
    - Learning outcome
    - Expected time requirements

- **Ending**
    - Summary of main points
    - Optional extra work

- **Signposts**
    - Route the lecture by headings, which are related to objectives and emphasize the the structure of the lecture
    - Indicate beginning and ending

     − Emphasize important points

     − Show relationships within the subject matter

3.4.2   Writing Style Tips

- Keep it simple

- Keep it short

- The 20 words per sentence is the maximum

- 5 - 7 lines per paragraph

- Maintain the variety in the writing style

- Use positive rather than negative expressions

- Use visuals and examples often
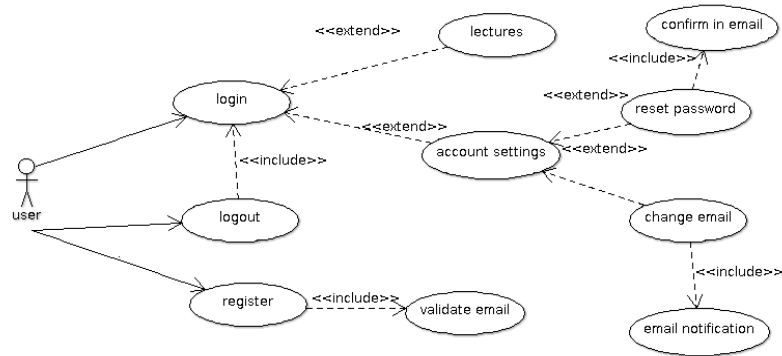
3.5   Use Cases

3.5.1   Prototype 1.0

Prototype 1.0: Login Use Case

1. User logs in.

   (a) User reads the lectures.

   (b) User adjusts the account settings.

     i. User resets password.

       A. User confirms password reset in email.

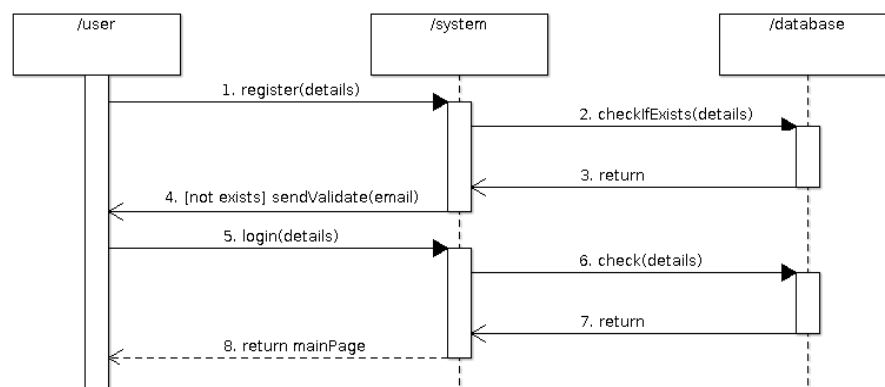   ii. User changes the address.

2. User logs out.

3. User registers the account

    (a) User validates the account in email.

Figure 3.3: *Prototype 1.0: Login Use Case*

3.6   Sequence Diagrams

3.6.1   Prototype 1.0

Register Sequence Diagram

Figure 3.4: *Prototype 1.0: Register Sequence Diagram*

Account Settings Sequence Diagram
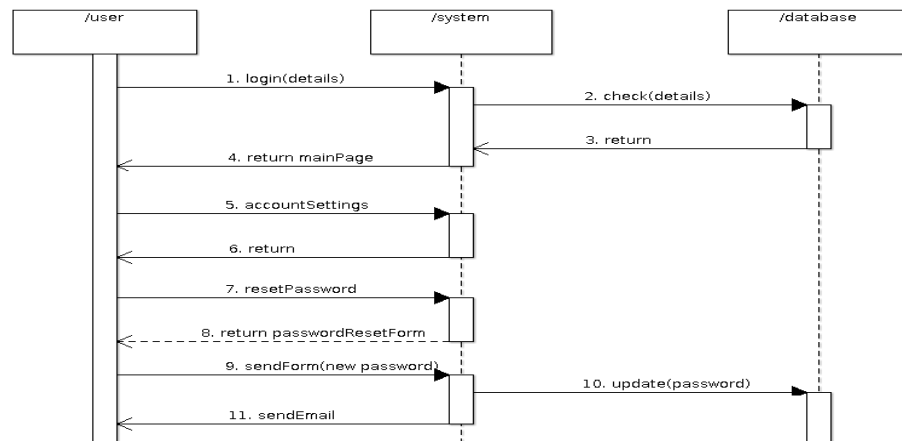

Figure 3.5: *Prototype 1.0: Account Settings Sequence Diagram*

3.7   User Interface Design

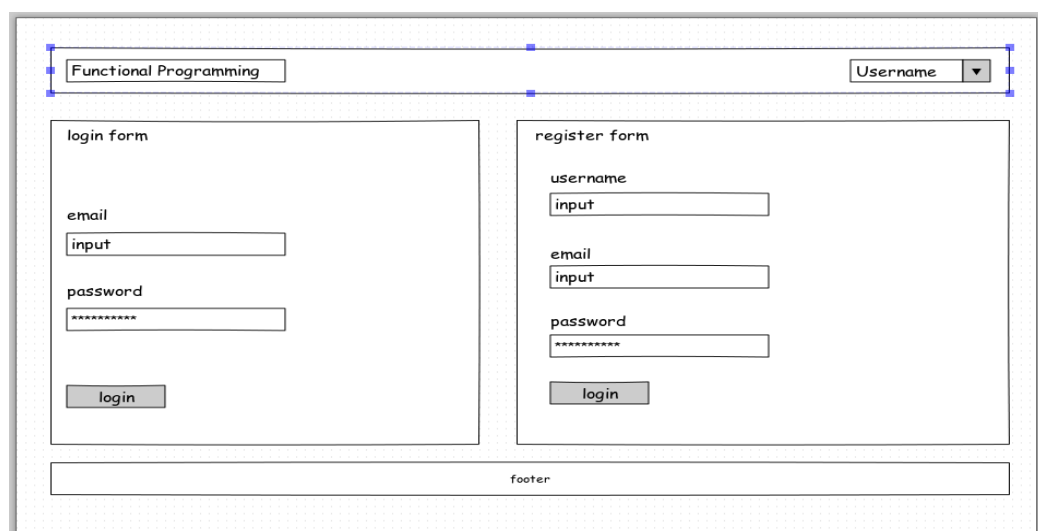## 3.7.1   Prototype 1.0

Main Page Wireframe

Figure 3.6: *Prototype 1.0: Main Page Wireframe*

On the main page, the user can log-in or register a new account. If the user attempts to log-in without the account, the attempt is rejected. The successful log-in will result in redirection to the dashboard page. Upon successful account registration, the user will receive the email with a confirmation link which he must click to validate the registration.

Required Elements

- header (label, hidden drop-down menu)

- login form (labels, text input, password input, button)

- register form (4 labels, text inputs, password input, button)
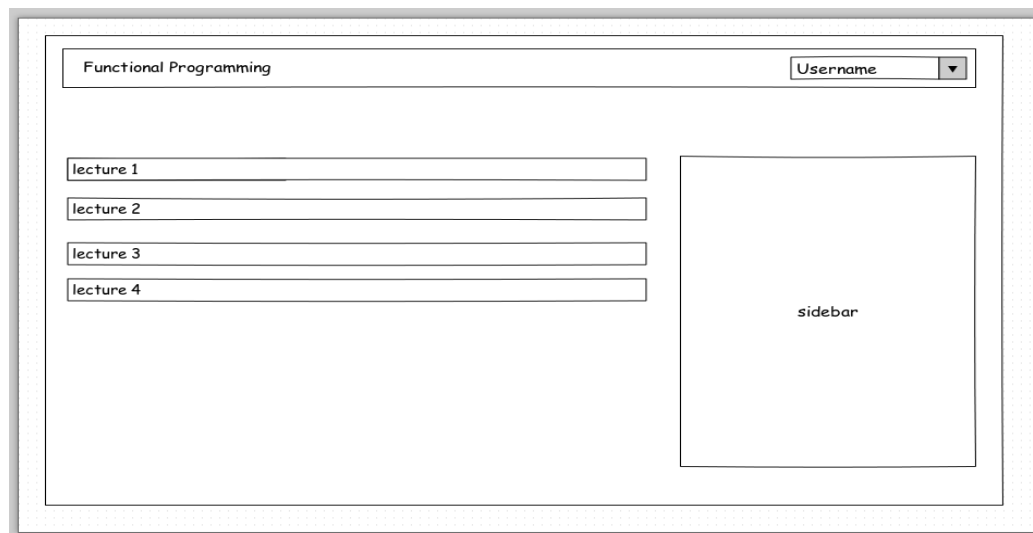
- footer

Dashboard Wireframe

Figure 3.7: *Prototype 1.0: Dashboard (lectures) Wireframe*

On the dashboard page user will have access to the lectures. Also, the drop-down menu in the header will be visible and will allow the user to request the account

settings page or log-out. The account setting request will result in the account settings page redirection and the log-out request will result in the redirection to the main page.

Required Elements

- header (label, visible drop-down menu)

- main area (links to the lectures)

- side area (for future functionality)
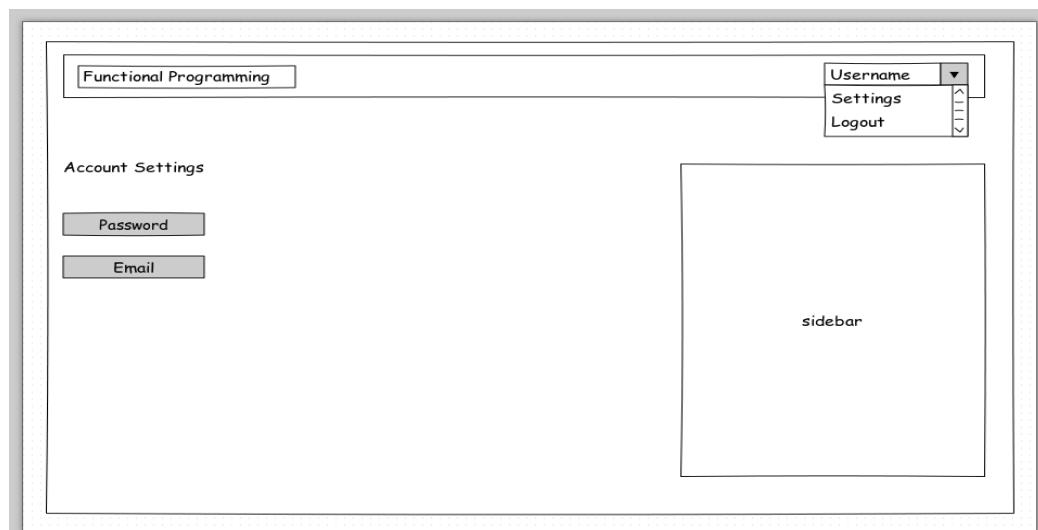
- footer

Account Settings Wireframe

Figure 3.8: *Prototype 1.0: Account Settings Wireframe*

On the account setting page, the user can change the password or the email address. Upon the password change request, the email will be sent to the user with the confirmation link. Upon the successful email change, the user will receive the notification email.

Required Elements

- header (label, visible drop-down menu)

- main area (2 buttons)

- side area (for future functionality)

- footer

3.8 Database Schema Design

3.8.1 Prototype 1.0

For the prototype 1.0 functionality, I will need the entity which will represent the user and the entity which will represent a random token. I will generate a token and associate it with the user every time a user requests registration, password change or email change. These tokens will then be embedded into an URL which will be sent to user's email. It's basically a way of identifying the user actions. The user then must click the URL to validate (confirm) the action. There should be a general expiration period associated with each token. Let's say one day. If the user doesn't confirm the requested action the action will be cancelled. For this functionality, I will need to save the token creation time-stamp as well.
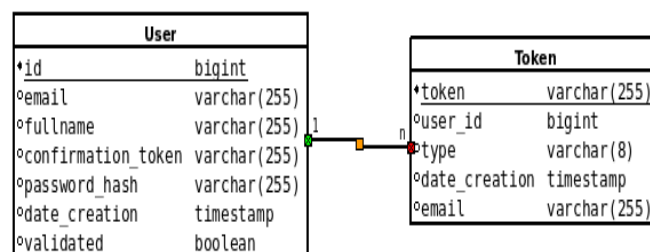
Figure 3.9: *Prototype 1.0: Entity Relations Diagram*

3.9   Back End Design

3.9.1   Prototype 1.0

In Play! Framework the user interfaces are represented by views. The view consists of HTML, Javascript and embedded Scala[30] code. The views are compiled to Scala classes and stored in /views folder. The views are rendered on demand by the server.

The user interacting with the views creates the requests to the resources. The incoming requests are handled by controllers. Each request is routed to the one controller method. Controller methods are processing the request and returning responses which in turn render the views. Controllers are stored in /controllers folder.

The requests are routed to the controllers by rules stored in /conf/routes file. In this file, the mappings form URIs to controller methods are defined. Play Framework uses REST like approach to routing. The /conf directory serves as a storage for all configuration files.

Figure 3.10: *View -> Request -> Controller -> Response -> View Diagram*

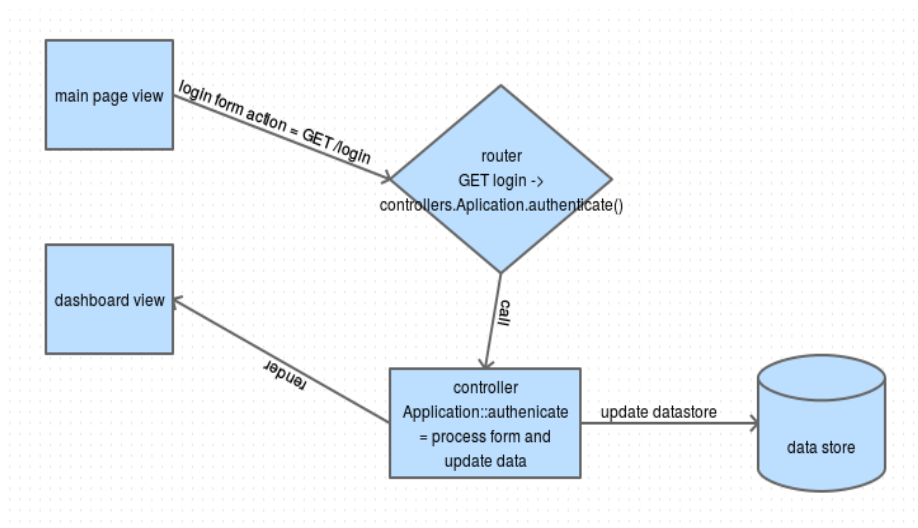Play Framework is persistence agnostic, but it provides Java Ebeans ER mapper and H2 in memory database by default. H2 can be changed for other database implementation for the production mode and any other ER mapping framework can used instead of Ebeans. In Ebeans, the each entity is represented by a POJO annotated class called model. Models are stored in /models directory.

The layout of a Play application is standardized to keep things as simple as possible. After a first successful compile, a Play application looks like this:

```
1   app                       → Application sources
2    └ assets                 → Compiled asset sources
3       └ stylesheets         → Typically LESS CSS sources
4       └ javascripts         → Typically CoffeeScript sources
5    └ controllers            → Application controllers
6    └ models                 → Application business layer
7    └ views                  → Templates
8   build.sbt                 → Application build script
9   conf                      → Configurations files and other non-compiled resources (on classpath)
10   └ application.conf        → Main configuration file
11   └ routes                  → Routes definition
12  dist                      → Arbitrary files to be included in your projects distribution
13  public                    → Public assets
14   └ stylesheets            → CSS files
15   └ javascripts            → Javascript files
16   └ images                 → Image files
17  project                   → sbt configuration files
18   └ build.properties       → Marker for sbt project
19   └ plugins.sbt            → sbt plugins including the declaration for Play itself
20  lib                       → Unmanaged libraries dependencies
21  logs                      → Logs folder
22   └ application.log        → Default log file
23  target                    → Generated stuff
24   └ resolution-cache       → Info about dependencies
25   └ scala-2.11
26      └ api                 → Generated API docs
27      └ classes             → Compiled class files
28      └ routes              → Sources generated from routes
29      └ twirl               → Sources generated from templates
30   └ universal              → Application packaging
31   └ web                    → Compiled web assets
32  test                      → source folder for unit or functional tests
```

Figure 3.11: *Play Framework Project Anatomy*

3.10   Project Repository Design

Project repository will be hosted on GitHub page: https://github.com/zubidlo/
itb_honours_project. I propose following directory structure:

```
 1   /lectures_markdown
 2        - 1. lecture
 3        - 2. lecture
 4        ...
 5   /poster
 6   /thesis
 7        /diagrams
 8        /figures
 9        /tables
10        - thesis.latex
11   /web_app (Play Framework Project)
```

Figure 3.12: *GitHub Repository Anatomy*

3.11   Conclusion

//TODO

Chapter 4

Implementation

4.1 Introduction

//TODO

4.2 Prototype 1.0

//TODO

4.3 Conclusion

//TODO

Chapter 5

Testing and Evaluation

5.1 Introduction

//TODO

5.2 Back End Unit Tests

//TODO

5.3 User Interface Tests

//TODO

5.4   Conclusion

//TODO

Chapter 6

Conclusion and Further Work

6.1 Introduction

//TODO

6.2 Achievements

//TODO

6.3 Personal Gain

//TODO

6.4 Further Work

//TODO

6.5 Conclusion

//TODO

Bibliography

[1] Neil Ford, *Functional thinking: Why functional programming is on the rise*, http://www.ibm.com/developerworks/library/j-ft20/, (2013-01-25)

[2] Typesafe, *Scala, Object-Oriented Meets Functional*, http://www.scala-lang.org/, (2002-2015)

[3] Typesafe, *Scala in the Enterprise*, http://www.scala-lang.org/old/node/1658, (2012-01-19)

[4] Brikman, Yevgeniy, *The Play Framework at LinkedIn: Productivity and Performance at Scale*, https://www.youtube.com/watch?v=8z3h4Uv9YbE&ab_channel=NewCircleTraining, (2013-06-26)

[5] Wampler, Dean, *We Won! How Scala Conquered the Big Data World.*, https://www.youtube.com/watch?v=AHB6aJyhDSQ&ab_channel=NewCircleTraining, (2015-03-01)

[6] theotown, *Akka wins 2015 JAX Award for Most Innovative Open Source Technology*, https://www.typesafe.com/blog/akka-wins-2015-jax-award-for-most-innovative-open-technology, (2015-03-23)

[7] Wikipedia, *Free software programmed in Scala*, https://en.wikipedia.org/wiki/Category:Free_software_programmed_in_Scala, (2015-01-05)

[8] Wikipedia, *Play Framework*, https://en.wikipedia.org/wiki/Play_framework, (2015-10-04)

[9] Paul Chiusano, Runar Bjarnason, *Functional Programming in Scala.*, Shelter Island, NY 11964, Manning Publications Co., 2015. ISBN 9781617290657.

[10] Dean Wampler, Alex Payne, *Programming Scala, Second Edition.*, 1005 Gravenstein Highway North, Sebastopol, CA 95472, O'Reilly Media, Inc., 2015. ISBN: 978-1-491-94985-6.

[11] Peter Hilton, Erik Bakker, Francisco Canedo, *Play For Scala.*, Shelter Island, NY 11964, Manning Publications Co., 2014. ISBN 9781617290794.

[12] Hudak, P., *The Conception, Evolution, and Application of Functional Programming Languages.*, s.l.:Yale University Department of Computer Science, 1989

[13] Hristakeva, M., Vuppala, R., *A Survey of Object Oriented Programming Languages.*, Santa Cruz: Univ. of California. 2009

[14] Hughes, J., *Why Functional Programming Matters.*, Goteborg Institutionen for Datavetenskap, Chalmers Tekniska Hogskola, 1984

[15] Goyri, A., Franklin, L., Dig, D., Lahoda, J., *Crossing the Gap from Imperative to Functional, Programming through Refactoring.*, Saint Petersburg: s.n., 2013

[16] Roy, P. V., *Programming Paradigms for Dummies: What Every Programmer Should Know.*, s.l.:IRCAM/Delatour, 2009

[17] Odersky M., Altherr P., Cremet V., Dragos I., Dubochet G., *An Overview of the Scala Programming Language*, s.l.:IRCAM/Delatour, Second Edition, 2006

[18] Wikipedia, *Read-Eval-Print Loop*, https://en.wikipedia.org/wiki/Read-eval-print_loop, (2015-11-21)

[19] Netty is an asynchronous event-driven network application framework ..., *Netty Project*, http://netty.io/, 2016

[20] Markdown, *Daring Fireball*, https://daringfireball.net/projects/markdown/, 2002-2016

[21] Ubuntu 14.04.3 LTS, *Canonical Ltd.*, http://www.ubuntu.com/download/desktop 2016

[22] Gummi, The Simple LaTeX editor, *alexandervdm*, https://github.com/alexandervdm/gummi, 2016

[23] Remarkable, *Jamie McGowan*, http://remarkableapp.github.io/, 2015

[24] Intellij IDEA, *JetBrains s.r.o*, https://www.jetbrains.com/idea/, 2000-2016

[25] Git, *Linus Torvalds, Software Freedom Conservancy*, https://git-scm.com/, 2005-2016

[26] Heroku Play Framework Support, *Heroku.com*, https://devcenter.heroku.com/articles/play-support, 2016-01-22

[27] Deploying Play Framework Apps with the Azure Toolkit for Eclipse, *Kirk Evans[MSFT]*, https://blogs.msdn.microsoft.com/kaevans/2015/05/12/deploying-play-framework-apps-with-the-azure-toolkit-for-eclipse/, 2015-05-12

[28] The High Velocity Web Framework For Java and Scala, *Typesafe*, https://www.playframework.com, 2016

[29] The Interactive Build Tool, *Typesafe*, http://www.scala-sbt.org/, 2015

[30] Scala Twirl Engine, *Play Framework*, https://github.com/playframework/twirl, 2016

[31] Ebean ORM for Java/Kotlin, *Avaje*, http://ebean-orm.github.io/, 2016

Appendix A

The Learning Material

A.1 Functional Programming with Scala

A.1.1 Preface

As a Computer Science student at ITB, I was introduced to computer programming through Java programming language. First, we learned the syntax of the language. Next, we have been introduced to the object-oriented approach to designing software and how to work with libraries such as Swing. Later in the third year, we learned the basics of concurrency and some design patterns for solving reoccurring problems. Around this time I crossed the initial steep part of the programming learning curve and I was comfortable enough with Java to learn some additional language features and Java Standard Edition platform libraries. Programming with Java became my hobby and I found a time to read additional material such as "Thinking in Java" from Bruce Eckel or "Effective Java" from Joshua Bloch.

At the same time, Oracle introduced important update number 8 to Java platform which, among other things introduced the syntax for anonymous functions (lambdas) and a number of libraries dedicated to functional programming style, such as Stream API. I was interested in this new functionality and started to learn to implement these tools and ideas into my code. Playing with lambdas, streams, and optionals I started to realize the power and elegance behind and I got really curious about the theory behind the functional programming.

Any student trying to learn about functional programming within JVM ecosystem will very soon become aware of programming language called Scala. The reason is that Scala has by design more richer support for functional programming than Java and hence is better suited as a vehicle for the study. At some point of the study one will become familiar with other programming languages such as Haskell or ML. There

are various reasons for this, main being that these language's syntax is designed to emphasize the functional programming ideas.

The following material is an overview of what I learned about functional programming in the period of 2 semesters in time dedicated to honors degree final project and my own spare time. I used Scala syntax to communicate ideas with code.

A.1.2   Pure vs Impure Functions

Well, functional programming is a programming style where programs are constructed using only *pure functions*. What are these pure functions you may ask?

Well, pure function is any function with following properties:

1. The function *always* evaluates to the same result given the same argument. The word always is emphasized here. The pure function can return the same result for different arguments, but it can never return different results for the same argument.

2. Evaluation of the result does not cause any semantically observable *side effect*. Well, what are these side effects then? In Computer Science, a function or expression is said to have a side effect if it modifies some state or has an interaction with calling functions or the outside world. Basically, a function has a side effect if it does something else than returning a result. Some examples of side effects:

   - Modifying a variable or a data structure in place
   - Throwing an exception or halting with an error
   - Writing to or reading from the console or a file
   - Drawing on the screen, updating a database, printing on a printer

Example: Common Pure and Impure Functions

```
1   sin(x)
2   length(list)
3   random(seed)
4   printf(string)
```

Functions *sin(x)* and *length(list)* are pure functions because they always return the same result for the same argument and do nothing else. Function sin(0) is always 0 and length of the same list is always the same result.

Functions *random(seed)* and *printf(string)* are impure functions because random(1) will not always return the same result. It will return a random number from 0 to 1.

Function printf("hello world") will print the string literal to the console and return no result. Even that is not guaranteed. Sometimes it can throw an exception if the console is not available for example. Impure functions are called *procedures* in computer science.

Example: Function definitions in Scala

```scala
def add1(a: Int, b: Int): Int = return a + b;
def add2(a: Int, b: Int): Int = a + b
def add3(a: Int, b: Int) = a + b
val add4: (Int,Int) => Int = (a: Int, b: Int) => a + b
val add5 = (a: Int, b: Int) => a + b
def add6(a: Int)(b: Int) = a + b
val add7 = Int => Int => Int = a => b => a + b
```

1. On line 1 is a function definition where I defined a function *add1* which takes two arguments of type Int and returns Int result which is just the addition of the arguments.

2. *add2* defines exactly same function without *return* word and *semicolon*

3. *add3* is the same function with return type left out

4. Line 5 is not a function definition statement, but assignment to an immutable value named *add4*. The value of an anonymous function described by *(a: Int, b: Int) => a + b* was assigned to the name *add4*. This form of function description is called a *function literal* or a *lambda expression*. It's the same function with emphasis on it's type: *(Int, Int) => Int*. It's a type of a function which takes 2 Int arguments and returns Int result.

5. *add5* is again a value which holds a function literal. This time the type is omitted and the function is only expressed by the lambda expression.

6. *add6* is again the same function defined in *curried form*, which we will discuss later.

7. *add7* is again the same function value, but with it's type *Int => Int => Int* emphasizes the curried form and the fact that every function is a function which takes one argument and returns one result. We will discuss these ideas later as well.

Here are the the function call statements where 4 and 5 were passed as arguments. All seven functions will return 9 as the result.

```scala
add1(4,5); add2(4,5); add3(4,5); add4(4,5); add5(4,5);
add6(4)(5); add7(4)(5);
```

Here are some important points to realize:

- All seven functions represent the same function which adds two Int values and returns the result.

- This function is a pure function because it always returns the same value 9 for the same arguments 4, 5 and doesn't do any side effects.

- In Scala and functional programming in general, functions are values. The same way as objects are values. We can assign functions to variables, pass them as arguments to other functions, create them inside the functions and return them as results of function computations. Later we will see that as long as the functions are pure they hold other very useful characteristics we can utilize in our functional programs.

- The functions can be expressed as literals. The same way we can create anonymous objects or a string literal "hello world", we can create a function literal in a form of a lambda expression.