QA Portfolio

Pavel Vjalicin
Softar Consulting Limited

Table of Contents

Company

Softar Consulting Limited consists of only two people. Me and my manager Arunas.

Arunas initially created the company to do contract software development work and software development consulting. Later on, I was hired to develop a new ambitious product from scratch.

Softar consulting focuses on developing complex software that requires years of experience to develop. Our small team size allows us to be flexible with our design without any unnecessary management overhead. Our mission objective is to simplify the lives of our clients by introducing software that automates or simplifies difficult tasks by using software.

Culture

Softar has a horizontal corporate culture. Everyone is encouraged to give their own personal input during discussions. You are generally free to do your own thing as long as there are no outstanding problems or emergencies. Creative thinking, going an extra mile to make the product better are highly encouraged. Work schedule is flexible unless meetings are scheduled during that day. If any additional input is needed you can freely arrange a meeting with anyone and talk to them directly.

My role

Currently, most of my time is spent developing our future product. It is a web application for financial advisers that helps managing assets and creates projections based on the input data. I am a lead developer for the product. I am responsible for product design and implementation.

Our current project will store a lot of user financial data, which is a major security factor that I am responsible for. I spend a lot of time to make sure that all of the authorisation methods that we implement are consistent and minimal to reduce the risk of potential human error that would lead to data breach. I have to make sure that all of our input fields are not vulnerable to SQL and cross-site scripting (XSS) injections.

I also need to make sure that our project complies with regulations. General Data Protection Regulation (GDPR) was introduced recently. The regulation puts many restrictions to the way we can use users' personal information. I constantly get questions from the project owner about things like: How do we store data? Do we store data of different organisations together? What are our data deletion policy? Can we track/provide the stored data on demand?

Regulations are no joke. Not complying with regulations can have dire consequences, even lead to bankruptcy.

Role requirements

My job requires me to have a variety of different skills:

Technical skills

Hands on experience with our development stack. In this case, having experience in HTML, CSS, JavaScript, CoffeeScript, SQL, Scala.

A broad understanding of User Experience design to make sure that the product we make is easy to use.

Critical thinking and problem solving abilities. Although, we have plenty of meetings where we can discuss new features, most of the time I am on my own figuring out
the best way to achieve our goals.

Soft skills

Great communication. Making sure that everyone knows what you are doing at all times can save lots of time and eases the process of development. I make sure to document and inform both my manager and the product owner of my progress and I am constantly compiling lists of TODOs and questions to ask people I am working with.

Managing relationships is also important. Bad relationships can cause problems for the business. People who have bad relationships might not want to communicate between one another as much as they need, not take other persons criticisms in a good way.

During our meetings I make sure to behave myself in a way that maintains and improves relationships. I make sure not to be too hard on people if they make a mistake, providing constructive criticism. I try to be patient, respect other peoples time.

When talking with people you should make sure that you are involving them as much as possible. Other people can view the same problem from a completely different perspective. Different people have different life experiences and skills. I always try my best not to overwhelm the discussion and look for people that are not engaged to ask for their opinion on the matter. This way you can make better, well informed decisions.

I also try to get to know people I work with. Having a casual discussion once in a while is a good way to get to know your co-workers, learn about their strengths and weaknesses. Not only can you make a new friend, this also can be useful during problem solving. Knowing peoples strengths, weaknesses or preferences can improve the overall productivity of a team.

When working with people social barriers may emerge. Some of the people you work with might come from different cultural backgrounds. This might impede your ability to communicate with that person and impact team's productivity in a negative way. Those kinds of things have to be resolved as soon as possible. The longer you let that sit, the harder it is to resolve. The way I would approach any social issue of this kind is by being honest, asking questions, engaging that person as much as I can and checking to make sure that the person with whom I speak understands the purpose of the thing being discussed.

I also try to be sensitive to the dynamics of the situation. Being aware of things that might disrupt the effectiveness of the communication is very important. Examples of those things are the ones that I mention above or things like past history and status within the company. You should always put things into perspective and be aware of the overall situation at hand.

Managing conflict in a work place is also crucial. Eventually, you will find yourself in a conflict with somebody and you need to have appropriate skills to deal with that. Conflicts can be caused by numerous of things such as: difference in perspective, different work approaches, miscommunication. The way you resolve the conflict is by being empathetic to the person. You must try to understand where they are coming from. Perhaps, propose a compromise to resolve a conflict where needs of both parties are at least partially met.

Time management. I am constantly trying to prioritize things to make sure that we have all the essentials done, before moving onto the more trivial tasks. Making sure that we always have something new to show to our product owner during our meetings.

The ability to learn fast. Software business is constantly evolving. It is essential to follow the general trends and learn from others.

Communication

There are many ways to communicate and choosing the right way is important. Depending on the situation you might want to use a different communication method in response to cues from the other person. For example a person can be in a loud environment at the moment so calling him would not be appropriate since he will not be able to hear you. Sending an e-mail is better in this case.

Writing an e-mail is a great way to inform someone about what you are doing in a very detailed manner. You have all the time in the world to construct a precise set of sentences that describe anything you want with as much detail as you need.

Calling someone is great for time essential activities. If you need to get hold of someone that precise moment in time. It is hard to discuss something complex while on the phone, in that case you should meet that person instead.

Presentations are great to explain complex structures and abstractions.

The kind of communication method I choose depends on the circumstance. If the topic is not time essential I usually just write an email, so the person can read it in their free time. On the other hand, if I need something fast, I usually call to get an immediate response.

Here is an example of an email I wrote to our project owner before our weekly meeting, discussing the topics I want to go through during the meeting:

Hi ______,

Since there are a lot of things I want to discuss during our Wednesday meeting I made a list of things me and Arunas want to go through.

Check those out if you have any spare time.

Organisation:

I do not have a clear idea of what kind of features we want to have when it comes to Organisation.

What kind of information (if any) should we include?

We probably should include some kind of license information. To extend their license or update their license.

Should we have a page for all IFA Members?

Can you reassign an IFA Member to different IFA?

Can you unassign a Member from IFA?

IFA Removal:
What happens to IFA's members when we remove IFA from our organisation?

In this case, are IFA members are still a part of an organisation and can be accessed by someone?

Do we have to reassign IFA Members to different IFA's?

Assistants of IFA:

Can an assistant have multiple IFA's to manage?

Can an assistant belong to a different organisation?

How do we manage assistants UI?

My suggestion is: An assistant has a different UI where he can see IFA's he is working with. From there on he is taken to a similar to IFA UI that we have at the moment but with restrictions.

If assistants can belong to a different organisation how do we add an assistant to IFA?

Licenses:

Our current idea for licensing
Have different license packages:
1) A stand alone package that is only covering one IFA.
2) An organisation focused package.
This license permits an Organisation to have X amounts of Users.
3) A network focused package.
This one is complicated and the terms of the licenses are going to be done individually.

How do we handle the amount of licenses when it comes to the organisation package?

Currently we either have or going to have those roles:
1) IFA
2) IFA's assistant
3) Organisation super admin - Has all organisation powers.
4) Organisation admin - Can manage organisation accounts. Has no access to trade secrets (Member Information). Can't add super admin role.
5) Organisation supervisor - Can manage organisation IFA assumptions. Has access to members. Can relocate assign members. Create members.

Do we set the amount of total users without considering roles or do we have a set amount of each role per license?

How do we handle IFA's assistants if they come from other organisations?

- Pavel Vjalicin

When communicating with people it is important to think about the risks of miscommunication. You need to make sure that your communications can't be misinterpreted. Expressing yourself clearly and being brief helps with that, but at the same time do not over-simplify. To avoid miscommunication I try to be as direct, concise and unambiguous. I write short, use simple words and go straight to the point.

Over-confidence in your communication skills can lead to problems, people come from different social backgrounds and have different experiences and because of that you should confirm if the other person understand what is being expressed. In my case, after writing an e-mail about something important I try to confirm if person understood my intentions the next time we meet with that person.

Transparency and full disclosure is also important while communicating with people. While responding to queries, I try to include as much detail as possible and produce a complete response to any given question.

You should always be polite while speaking to people. Make sure to that the person you are speaking with knows you are listening, occasionally make eye contact, nod. Speak clearly and try not to shout, this makes spoken communication easier for everyone. Ask questions when it is appropriate.

My background

Before working for Softar Consulting limited, I was a free-lance software developer / graphics designer. During this time I have acquired multiple skills that I use today. I learned a lot about computer science, business, communications and design. Working by myself have taught me to be disciplined, to be objective oriented and to never stop learning new things.

My prior work experiences greatly help me with my current work. Working as a free-lance developer requires a lot of creative thinking and great communication skills. You have to adjust to the given circumstances and come up with good solutions while at the same time communicating with the client about what they need and why. All of this helps me during the meetings where I can come up with creative solutions to problems and explain the idea behind those solutions in a clear way.

I feel like my past have left me with gaps in my skills that I have to improve on. Primarily, I don't think that I have great technical communication skills. I only worked in small teams before, where technical communications were not important. I find it difficult to explain the rationale behind my design decisions and defending my reasoning. My long term goal is to work on high-end computer systems such as machine learning. I have been studying a lot of mathematics and developing Machine Learning projects in my spare time in preparation for that.

Product

The focus of the product is a simple approach to retirement. We want to make sure that IFAs can explain and optimize retirement plans for their clients in an easy way. Saving Independent Financial Advisers (IFA) time. Improving the relationships between IFAs and their clients.

There are two main stakeholders. The product owner, who is interested in getting the product out into the market as soon as possible, with features that our target audience needs to make their jobs easier. And my manager who hired me to develop the software and to to develop a modern systems architecture that is going to be used for the future projects.

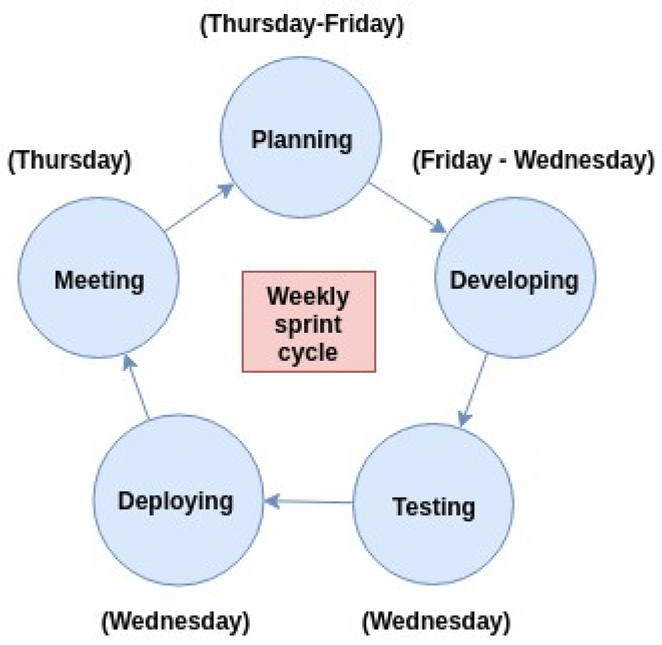
Prioritizing stakeholders is key to success and my biggest priority from day one was getting the project to a state of Minimum Viable Product so we can start selling as soon as possible and start bringing in profits from my work. At the same time I am also focused on making sure that the development system I am developing is well designed and flexible, fit for use in future projects.

I work directly with the product owner. I make sure that the product owner is informed on current status, any changes made to the product. We usually have a meeting or two every week, those are usually arranged by me or our product owner using a phone or email. Our meetings usually consist of 2-4 people (Me, my manager, product owner, external software engineers). We go back and forth on the design decisions. During meetings we review the changes that I made, discuss future plans, next week objectives. We use an Agile software development life-cycle, iteratively go through the changes made and discuss making changes on top of that.

On the right is a diagram showcasing my typical work week. We usually aim to get all of the week's work done by Wednesday. To make sure that everything is working on Thursday.

After implementing as many changes as I can, I begin testing new features. Testing is being done manually for trivial things, for more complex systems we use automated tests.

We generally don't write many unit tests because we are using a strongly-typed functional programming language Scala.



Our code is usually designed to be compile-time safe (errors are being caught by compiler), because of that we don't need to test our software as much.

After testing is done, I deploy the program to our testing server to make sure that other people can review the changes and prepare for the Thursday meeting.

Meetings

We have meetings on Thursday where we review the latest changes and discuss things that we need to implement and establish short-term goal for next week's meeting. After the meeting is done I plan the next week development cycle. Me and my manager think about what kind of systems we need to develop and how to achieve the features that we discussed during the meeting.

During the meetings we discuss the ways we can improve our product. Pension modelling is a very complicated thing and our goal is to minimize the complexity of our tool as much as possible. We make sure that our tool is easy to use for an IFA and is appealing, visually pleasing and informative for the IFA's client.

I am constantly looking for ways of improving the project outside of the specifications and if I come up with something new and exciting I prepare a brief presentation (accompanied by design documents when appropriate) and pitch my idea to stakeholders. After discussing pros and cons we either agree to implement that new feature or just move on to something that we think is more important based on the outcome of our discussion.

We generally try to tackle one problem at the time to avoid overwhelming ourselves with excessive amounts of information and we try to go back and fourth between people to avoid selection bias.

Since the project that I am currently work on is quite complex, a large part of my job is dealing with changes. One of the meetings we've had was about user permissions. During this meeting we discussed what kind of user permissions should we implement, how to implement them and comparing all of that to our current system. Soon after we began the discussion, I have noticed that our purposed user permission implementations might be too hard to follow for the end-user, who might not be tech-savvy. I made sure to bring up the complexity of our system and explained, in detail, about possible complications (the system is too hard to use). The project owned agreed and we started to talk about ways to simplify things for our users. By the end of the meeting, after we have agreed on the changes, the project owner wrote a specification document for me to use as a basis for user permission system.

Here is a part of the specification document I have received:

IFA Modeller Sign-up Process and Permissions 8 February 2019
Comments: 8 February 2019
Managing Advisers
Managing Advisers do not automatically have access to all clients.
The “Clients” section in Tools should only see clients where the Managing Adviser has been given permission.
However, the Managing Adviser can give themselves access to particular clients using the “Manage Users” or “Manage Clients” section.
Manage Users – adviser view
- remove the “profile” and “Advised Client” tabs that appear when you click on a User.
When you click on a user it should be made clear which user you are managing.
The text at top that says
“Username” has the following client permissions:
Manage Clients - adviser view
The two table are my clients and other clients that I have permission to see.
When you click on a client it should be made clear which client you are managing.
User permissions for “Client name”
Remove the two tables: organisation managing advisers and organisation website administrators

In the above report, the project owner outlined how our user roles should behave. In this case the request was to change the way we display data on our “Manage Clients” and “Manage Users” pages while while current user has a role of Managing adviser. So I made sure to implement things that are mentioned here the following week.

While implementing things I make sure that I set reasonable expectations of what can be done and how much time will it take. There is a saying: “Under promise and over deliver” and while I do agree with the general premise I am not sure if this can be applied in all of the cases. In my case, I usually evaluate the task based on my prior experiences and if I am doing something that is new to me I try to think of a reasonable time-scale and double that.

Generally, we try to push for an update every week. It is essential for our business to follow strict schedule. Sometimes updates do get delayed, because of unforeseen circumstances. Although I try to be as productive as I can, I make sure to get enough sleep, move a little, have a rest once in a while. You can’t make shortcuts when it comes to your health. But at the end of the day, the business I work for has a limited budged and I am responsible for making sure that we make progress in a timely manner.

During that same week, we went through a process of transferring our database to a production database. We used to store all of our data with tools that were designed only for development purpose. I migrated our software from development database to a proper MYSQL database.

We have also implemented a L2 Cache on our database queries to optimise the performance of our application. L2 Cache automatically stores all the necessary data in RAM rather then HDD. You only pull the data from database once and after that you only store the changes.

Generally speaking, L2 Cache helps us improve the performance of our application on a generic level (without any maintenance overhead). Increase in performance means that I do not have to spend as much time optimising our software to get the same results, drastically improving my ability to produce code. It gives me more time that I can spend on things like systems design and user experience.

Because of those fundamental changes to our application we have started to encounter bugs that would brake our software. It was up to me to diagnose and solve those issues.

By using the Intellij debug tool and looking at error logs I was able to determine that those bugs were indeed caused by either L2 cache system and our database transition. Based on my research I have figured out that the development database that we used was not as strict on enforcing SQL restriction such as “not null” (variable can’t be empty). This was easy to fix. I just went through all the troublesome models and made sure that they are being initilised properly and all of the required variables are filled in. We have also encountered an error where our L2 cache was not updated properly (in some cases) where the model relationships were a bit more complex. This was solved by manually updating our L2 cache after a complex operations are executed on our codebase.

Our product owner does not come from technical background. Because of this, sometimes the suggestions that he proposes does not quite fit into our system making it difficult to implement and not being very useful. During those kind of moments I usually insist on either postponing the suggested feature or suggest that we discuss this with my manager, who does come from technical background, or I suggest an alternative that does a similar thing but is a lot easier to implement.

Creative Thinking

At times some solutions require you to let go of your technical knowledge and immerse yourself into creative thinking.

From the beginning of the development of this project we decided that we wanted our website to load dynamically. Dynamic web-site or web application is a web-site that doesn’t load the whole page but loads only the necessary bits. This is done with javaScript by dynamically handling XMLHttpRequest (XHR).

Initially we started to implement the dynamic side of our website with JavaScript and Jquery, but down the line we discovered the following problems with this method:

1. JavaScript is a dynamically type language which in turn means that it is easy to make mistakes and you have to spend a lot of time debugging bugs.

2. The client-side and server side of application can't talk to each other because they are written in different languages, meaning that some times we have to write the same code for both the client and server side.

3. Jquery doesn't use OOP to get a specific DOM element. This combined with the dynamic typing system of javaScript means that any changes made to DOM can potentially break your program, without you ever knowing about it.

So based on that I wasn't really happy with our current dynamic request handling and was looking for the alternatives.

I started the way you normally start: by looking for solutions made by other developers. I've looked into Front-end framework like Angular, React and Vue. All of those framework do exactly that, dynamic XHR handling.

Although those were great options, we are not there yet. Those framework split your front-end into objects called "components" that make your app a lot easier to manage. None of those framework address the problem of dynamically typed JavaScript. Angular tries to address this problem with TypeScript, but it's not very good. TypeScript is soft typed (you have to assign types to DOM Elements explicitly if you want to use them properly).

During my further research I stumbled upon a Scala library called ScalaJS. The premise of this library is simple: It takes your Scala code and compiles it to JavaScript with all of the type validation on compile. Essentially you can write type-safe Scala code and execute it on the browser.

An idea was born. What if we can write a front-end framework with all of the Object Oriented Programming and Functional Programming systems provided by Scala?

I really liked react when I first learned about it. It was designed to be maintainable, flexible and fast. The maintenance part comes from the component system it has. The speed comes from the virtual DOM that react creates. And the flexibility comes from the way it was designed: You can implement react into your website step by step (You can use as little or as much React as you want. Components can be used to display a single element or the whole website).

So that was my idea. To take the principles that make React great and to express all of that in a functional programming/type-safe way using Scala. Not only did this approach solve the type-safety issue, it also gave us the ability to use Scala code on both client and server sides. Also, now we could use the same code for both client and server sides of our application.

Technology

Innovation and technology are the key factors in gaining an advantage for our company, especially in a fast paced business environment such as software development. Failing to keep up with the changes in the industry may lead to dire consequences.

One of many ways we try to gain advantage in the industry is by using modern tools and software development techniques described below. The combination of those allow us to spend less time on maintenance, writing tests, code refactoring.

Server side

Our web applications is build on Play Framework.
It is fast, it is simple and it supports Scala. It is a Model-View-Controller based framework.

Our primary language of choice is Scala. It is a functional programming language based on JVM byte code that supports object oriented programming language features.

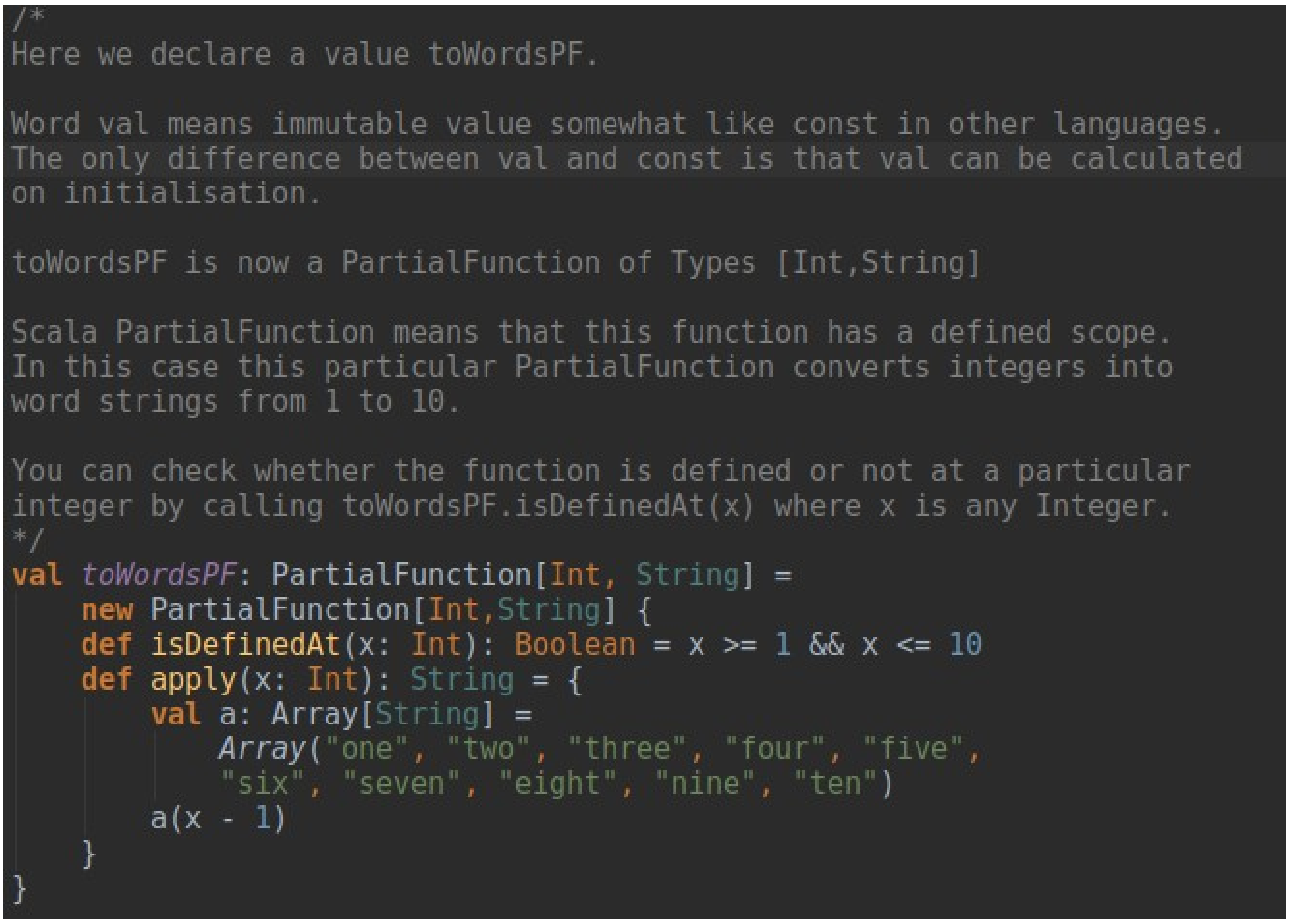
The functional side of this language provides us with the ability to structure our code in a simple way, it is easier to work with data, allows us to write less code. Less code = less bugs. Easier maintenance.

Object oriented side lets us construct complex inheritance structures that, again, lets us minimize our code base.

JVM byte code element provides us with power of Java libraries. You can use most of Java libraries with Scala without any integration.

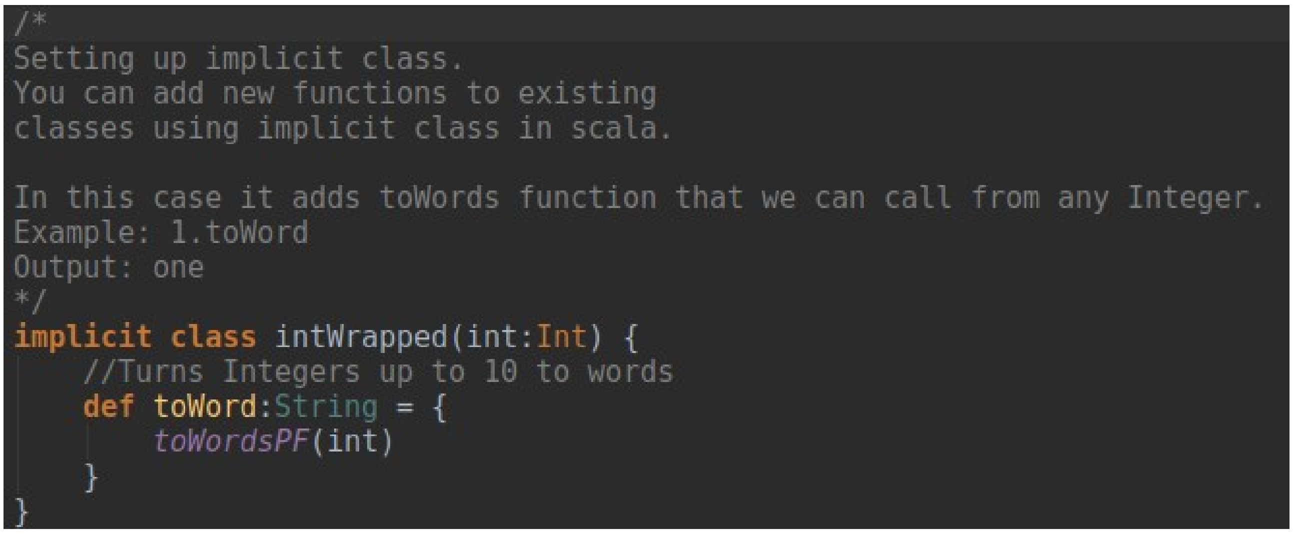
Here is a small demo that I have prepared to show some of the Scala language features.

A partial function example:



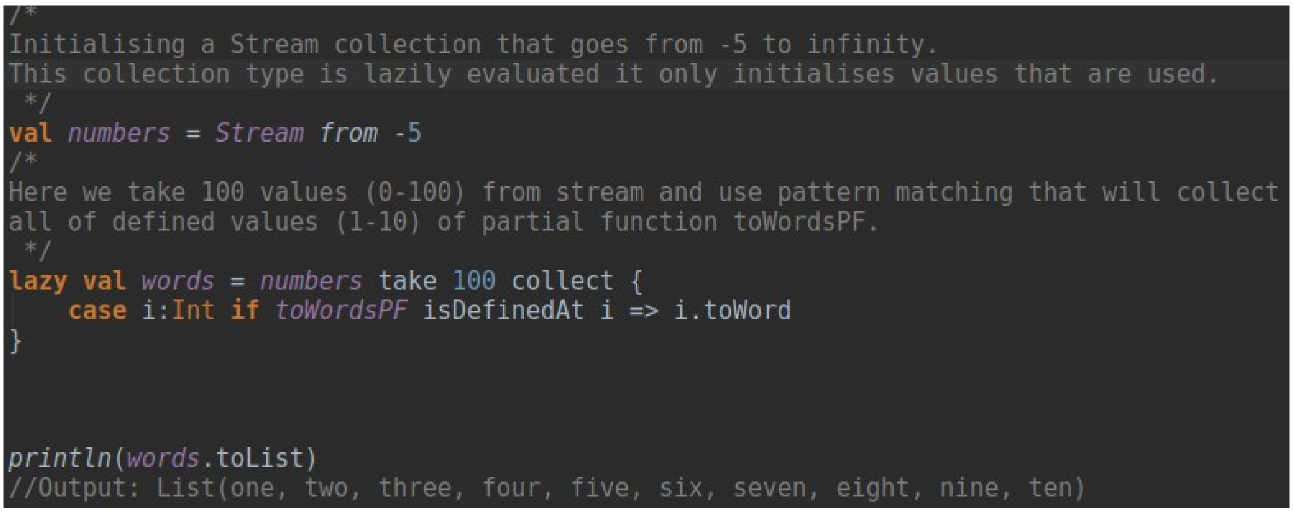
```scala
/*
Here we declare a value toWordsPF.

Word val means immutable value somewhat like const in other languages.
The only difference between val and const is that val can be calculated
on initialisation.

toWordsPF is now a PartialFunction of Types [Int,String]

Scala PartialFunction means that this function has a defined scope.
In this case this particular PartialFunction converts integers into
word strings from 1 to 10.

You can check whether the function is defined or not at a particular
integer by calling toWordsPF.isDefinedAt(x) where x is any Integer.
*/
val toWordsPF: PartialFunction[Int, String] =
    new PartialFunction[Int,String] {
    def isDefinedAt(x: Int): Boolean = x >= 1 && x <= 10
    def apply(x: Int): String = {
        val a: Array[String] =
            Array("one", "two", "three", "four", "five",
            "six", "seven", "eight", "nine", "ten")
        a(x - 1)
    }
}
```

Implicit class example:

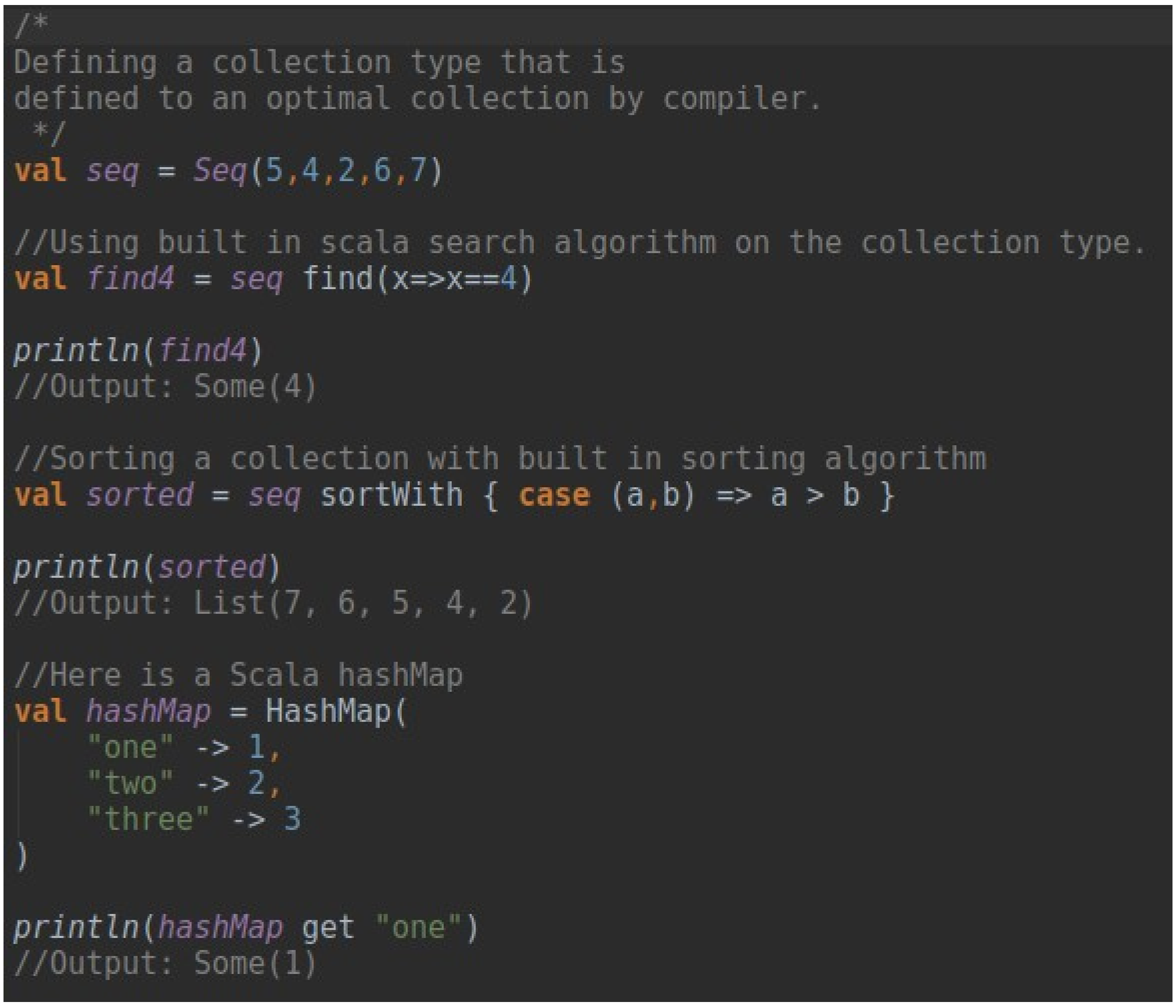


```scala
/*
Setting up implicit class.
You can add new functions to existing
classes using implicit class in scala.

In this case it adds toWords function that we can call from any Integer.
Example: 1.toWord
Output: one
*/
implicit class intWrapped(int:Int) {
    //Turns Integers up to 10 to words
    def toWord:String = {
        toWordsPF(int)
    }
}
```

Example of using both implicit class and partial function together to convert a list of numbers to words if they are defined by the partial function:


```
/*
Initialising a Stream collection that goes from -5 to infinity.
This collection type is lazily evaluated it only initialises values that are used.
 */
val numbers = Stream from -5
/*
Here we take 100 values (0-100) from stream and use pattern matching that will collect
all of defined values (1-10) of partial function toWordsPF.
 */
lazy val words = numbers take 100 collect {
    case i:Int if toWordsPF isDefinedAt i => i.toWord
}



println(words.toList)
//Output: List(one, two, three, four, five, six, seven, eight, nine, ten)
```

Example of using different data types, search and sort algorithms:


```
/*
Defining a collection type that is
defined to an optimal collection by compiler.
 */
val seq = Seq(5,4,2,6,7)

//Using built in scala search algorithm on the collection type.
val find4 = seq find(x=>x==4)

println(find4)
//Output: Some(4)

//Sorting a collection with built in sorting algorithm
val sorted = seq sortWith { case (a,b) => a > b }

println(sorted)
//Output: List(7, 6, 5, 4, 2)

//Here is a Scala hashMap
val hashMap = HashMap(
    "one" -> 1,
    "two" -> 2,
    "three" -> 3
)

println(hashMap get "one")
//Output: Some(1)
```

Based on the above examples, you can see that I try and follow the coding standards of our company. We use 4 space indentation. We make sure that our code is easy to read and if for some reason it is not, we make sure to comment our code. Follow function programming practices:
1. Use immutable data structures.
2. Make sure that our code is flat. (Avoiding unnecessary class definitions)
3. Write compact functions that can be re-used in a generic way.

At the moment we use h2 database engine for our data storage and we use object-relational mapping tool eBean. It makes our jobs easier by removing the need to pass SQL queries directly. Instead we just use Model objects to assign and remove data from our database.

The build tool that we use is SBT which a standard build tool provided for Scala development. SBT lets you determine the way you compile your project.

We use a combination of Java and eBean to create our database tables.

Client side

We use CoffeeScript compiler. It takes functional coffeeScript code and transforms it into JavaScript that runs on all browsers. CoffeeScript improves the syntax of JavaScript and provides me with language functionality that may or may not be supported by any given browser.

Some of our code is written using Scala.js which is a compiler that compiles our Scala code to javaScript. This is useful because we can you Java libraries in our javaScript and we can retain the type-safety of Scala.

We use a front-end framework for asynchronous page loading that was developed by me. It provides us with an easy, maintenance free way of partially loading web page. So instead of reloading the whole website page, we only load the bits that are necessary. This greatly improves user experience by drastically reducing the loading time of pages. It is especially noticeable while accessing our web application via a mobile device.

We use bootstrap to make our website responsive. This greatly increases user experience of our web application.

We use jQuery to simplify syntax and reduce our code base. Jquery also manages browser compatibility since it provides a common interfaces between function implementations of all modern browsers.

We also use different kinds of poly fills to meet our browser specifications and many different small javaScript libraries to, again, reduce our code base.

We use a template engine called Twirl. It lets us to use Scala to convert server side logic to html and display all the data we need.

Other tools

We use Visual Studio and IntelliJ IDEs. Only two IDE's that support our chosen languages IntelliJ and Eclipse, we went for IntelliJ because it has a minimalist design in comparison with Eclipse.

IntelliJ IDE is a great tool that greatly improves productivity. It lets you navigate through your project really fast using shortcuts and searching by file name, class or text. It auto completes function names, displays errors, shows types that are used with Scala programming language. Lets you refactor code automatically. Automatically generates things like Java constructors and default getters and setters for Java. Integrates with GIT, makes merging, committing, pushing, re basing a lot easier.

We use GIT as our version control system. GIT is pretty much the industry standard at this point in time.

And we use mantis bug tracker as our task management system.

Application

Code re-usability

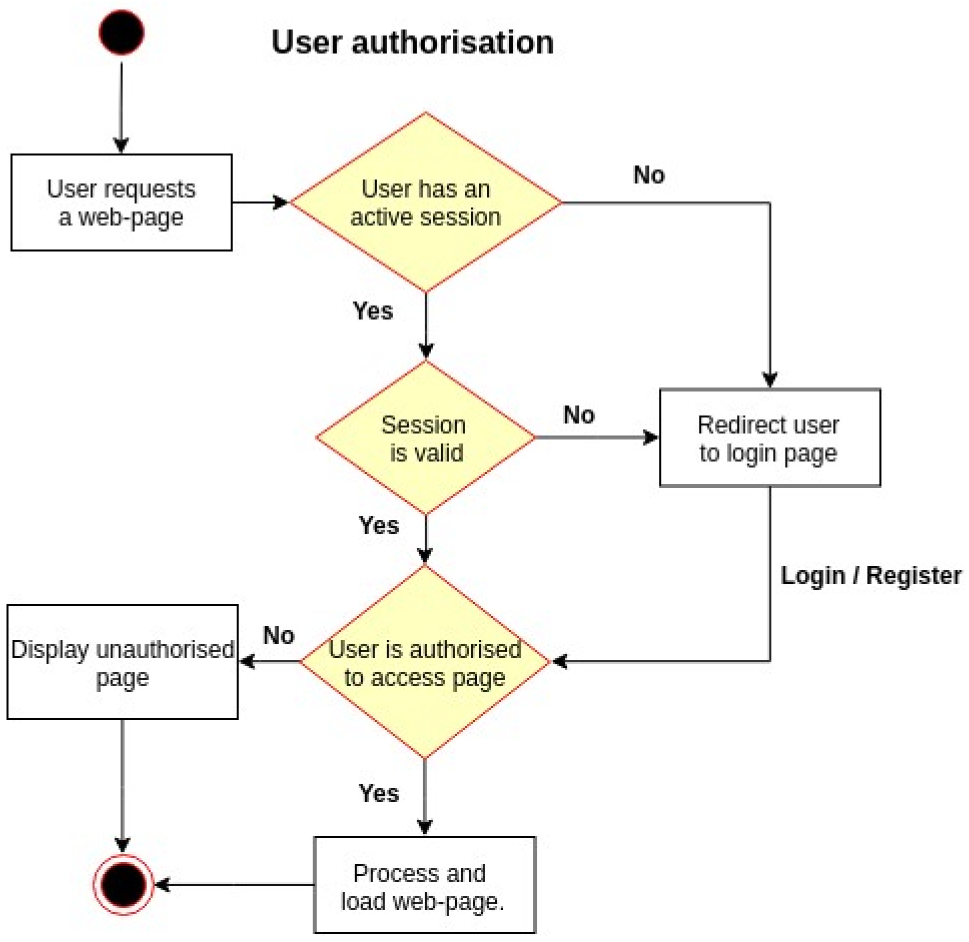
One of the biggest advantages of software engineering is work re-usability. Since all of our work is digital you can easily make a copy, reuse, re-purpose your code base.

When it comes to my work, I try to split things into small components that can be later reused for different things. This way you reduce the size of your code base, which in turn reduces maintenance costs.

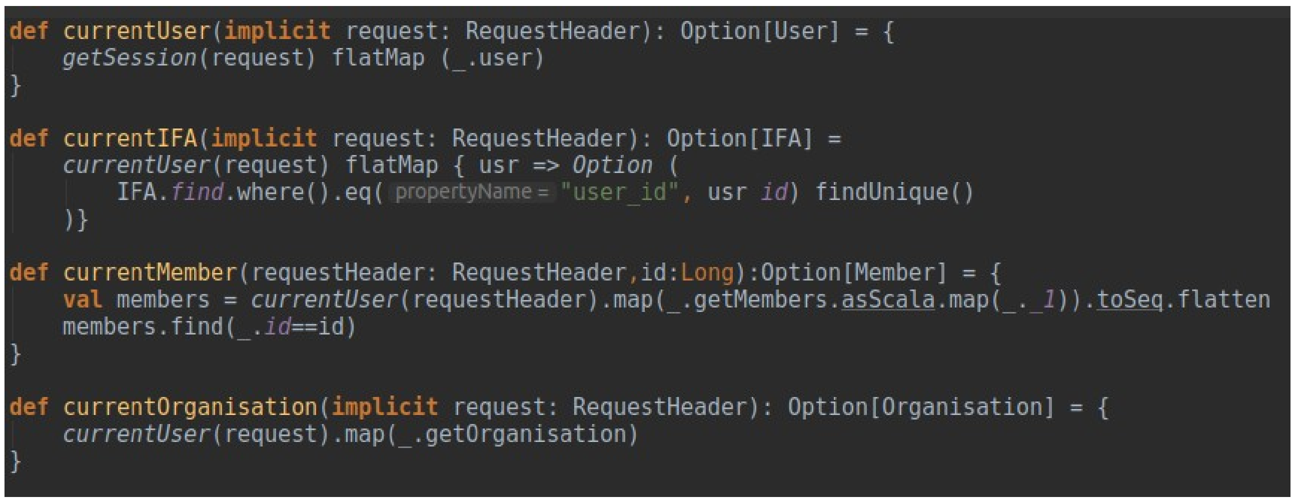
One of the main things where we reuse our code is the authentication system.

Authentication is one of those things that have to work very reliably or you risk a opening your software to potential security vulnerability. Therefore, all of the code that handles security has to be as compact as possible, less code means that fewer things can go wrong.

The basics behind our user authorisation concepts are outlined in the activity diagram below:


User authorisation
User requests a web-page
User has an active session
No
Yes
Session is valid
No
Redirect user to login page
Yes
Login / Register
Display unauthorised page
No
User is authorised to access page
Yes
Process and load web-page.

Here is an example of some of the authentication components that I have written for our application:

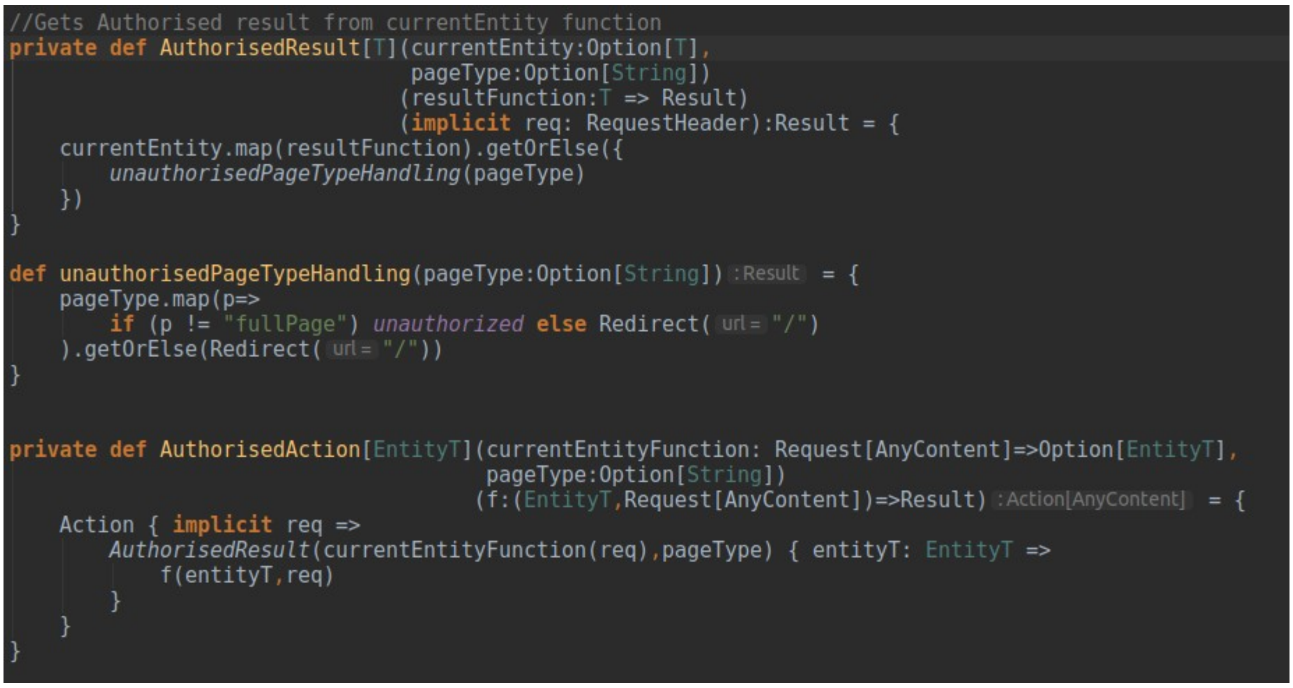

```
def currentUser(implicit request: RequestHeader): Option[User] = {
    getSession(request) flatMap (_.user)
}

def currentIFA(implicit request: RequestHeader): Option[IFA] =
    currentUser(request) flatMap { usr => Option (
        IFA.find.where().eq( propertyName = "user_id", usr id) findUnique()
    )}

def currentMember(requestHeader: RequestHeader,id:Long):Option[Member] = {
    val members = currentUser(requestHeader).map(_.getMembers.asScala.map(_._1)).toSeq.flatten
    members.find(_.id==id)
}

def currentOrganisation(implicit request: RequestHeader): Option[Organisation] = {
    currentUser(request).map(_.getOrganisation)
}
```

Here you can see how we get our main database entities which are (User, IFA, Member, Organisation).

CurrentUser function is the main authorisation function that validates the user based on the requestHeader which has our session cookie attached to it which in turn we use to validate that the user has sign up and is authorised to have access to a certain entity.

Each of those functions, except for the currentUser function, use currentUser function to get the proper entity. Access to every major Entity in our system written in 14 lines of code.

Here is another example of reusable code:



```scala
//Gets Authorised result from currentEntity function
private def AuthorisedResult[T](currentEntity:Option[T],
                                pageType:Option[String])
                               (resultFunction:T => Result)
                               (implicit req: RequestHeader):Result = {
    currentEntity.map(resultFunction).getOrElse({
        unauthorisedPageTypeHandling(pageType)
    })
}

def unauthorisedPageTypeHandling(pageType:Option[String]) :Result  = {
    pageType.map(p=>
        if (p != "fullPage") unauthorized else Redirect( url = "/")
    ).getOrElse(Redirect( url = "/"))
}


private def AuthorisedAction[EntityT](currentEntityFunction: Request[AnyContent]=>Option[EntityT],
                                      pageType:Option[String])
                                     (f:(EntityT,Request[AnyContent])=>Result) :Action[AnyContent]  = {
    Action { implicit req =>
        AuthorisedResult(currentEntityFunction(req),pageType) { entityT: EntityT =>
            f(entityT,req)
        }
    }
}
```

Here we see authorisation result handling. The unauthorisedPageTypeHandling declares how we should handle unauthorised access, in this case we either give back unauthorized response, which is process by our javascript libraries, or simply redirect to the main page.

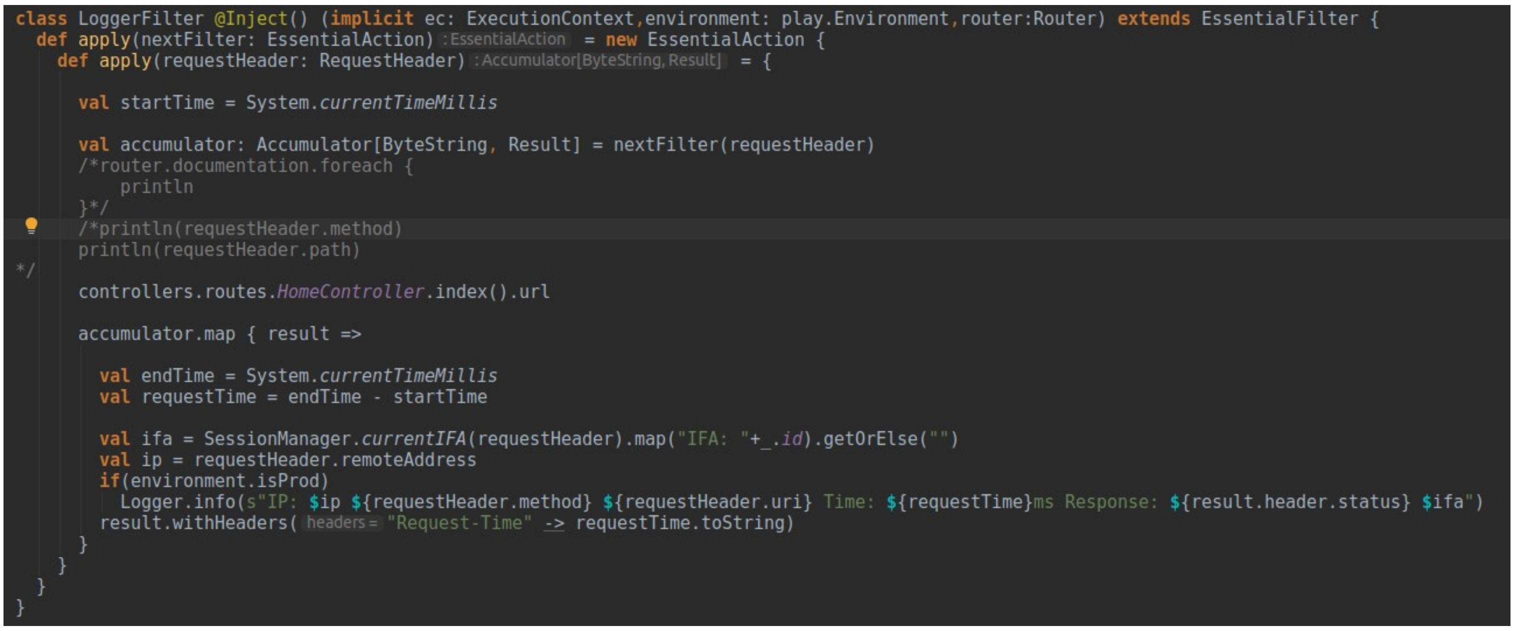
Deployment

One of my responsibilities at the company is deploying our development build for showcase purpose for potential future clients.

During development, our application runs in development mode, which is a non optimised version of our software. The files at that point are not compressed, we don’t log requests and many debugging features are enabled. This makes our software compile faster and generally easier to work with.

When we need to showcase our website we build a production version of our software. It takes a bit of time, because we have to compress all of our javascript, html, css files for client-side. Compile all of our codebase into jars for server-side. Run all the tests. The production build itself is mostly handled by Play Framework. We just run an sbt command “stage” and end up with an executable that we run from our server.

Here is an example of a logger filter that is only used during production that I wrote:



```
class LoggerFilter @Inject() (implicit ec: ExecutionContext,environment: play.Environment,router:Router) extends EssentialFilter {
  def apply(nextFilter: EssentialAction) :EssentialAction  = new EssentialAction {
    def apply(requestHeader: RequestHeader) :Accumulator[ByteString, Result]  = {

      val startTime = System.currentTimeMillis

      val accumulator: Accumulator[ByteString, Result] = nextFilter(requestHeader)
      /*router.documentation.foreach {
          println
      }*/
      /*println(requestHeader.method)
      println(requestHeader.path)
*/
      controllers.routes.HomeController.index().url

      accumulator.map { result =>

        val endTime = System.currentTimeMillis
        val requestTime = endTime - startTime

        val ifa = SessionManager.currentIFA(requestHeader).map("IFA: "+_.id).getOrElse("")
        val ip = requestHeader.remoteAddress
        if(environment.isProd)
          Logger.info(s"IP: $ip ${requestHeader.method} ${requestHeader.uri} Time: ${requestTime}ms Response: ${result.header.status} $ifa")
        result.withHeaders( headers= "Request-Time" -> requestTime.toString)
      }
    }
  }
}
```

This filter records all of the requests and stores them in a log file. All of this is done for audit, debugging and security purposes. We print out the IP of the request, request method, request url, request result, and the time it took to process the request.
This code is set to only work in production, since we don’t use this information during development.

After the build is compiled, I briefly test it by hand on my local machine to make sure that everything works. I push the latest build to our remote GIT repository and login to our remote server where we host our demo applications. I connect to the remote server using Secure Socket Shell, pull our up-to-date version of software from GIT repository and build the project from the remote server. From there on, I stop the old version of our website and execute a console command to start running the new version of software.

At the moment we don’t really do database migrations since our database setup is not yet finalised. I just delete all the data we had and upon start-up a script is executed that fills up our database with filler data for showcase purposes.

After the application was deployed I inform the project owner about the changes that were made via email, phone call or a meeting. Thoroughly explain how the new features work teach the project manager how to use new features.

Here is an example of an email I send after I deploy a new version of the application:

Hi [redacted],

A new version of our IFA tool-kit is now available at: [redacted]

Login details:
Username: [redacted]
Password: [redacted]

Affordability modelling is ready to go. At the moment only DC Pension assets are available for use with affordability modelling.

There are still a couple of non-work disruptive minor issues, that need to be resolved. These are going to be patched on Thursday morning.

Thursday's updates will not impact saved data, so you can start filling up data for the presentation.

All of the organisational features have been disabled for this release since they are not quite ready to be showcased.

Let me know if there are any issues with this release.

Will you be able to swing by our office this Thursday?

- Pavel Vjalicin

*Some of the text in this email has been censored.

User training

Ineffective use of software can make it less effective or useless, in order to avoid that you have to make sure that your users know how to use your software properly. Because we decided to use Agile method for this particular project, we keep things simple.

During our meeting with the project owner, I go through the changes that were implemented that week and showcase how they work. Because of constant changes to the project, we try to keep things brief and avoid writing excessive documentation. User training problems are usually solved by the software itself. Our software is designed to be self-explanatory.

Testing

Application Testing

Most developers would agree that testing is essential when it comes to software development.

Generally speaking, testing is used to manage the risks involved in software development. You write code that makes sure that your code is running properly, making it harder to make mistakes.

Some people use tests to help design and describe the software they are developing by using such development techniques like Test Driven Development. The general idea behind Test Driven Development is that you start with writing all of the tests necessary to achieve your goals and after that you actually work on implementing the features that your tests describe. This way the only thing you need to worry about is making sure that your test are correct and describe the objectives that you want to achieve with your software.

When it comes to our company, we try to minimise the amount of tests we write because of the following reasons:

The specifications of our current project change quite frequently. That means that for every iteration of our product we would have to rewrite tests, almost doubling the work required.

Instead of focusing on tests, we focus on writing our code in a compile-time safe way. This simplifies the development process because most of the issues we can encounter are caught by the compiler.

Here is one of the unit tests I have implemented in our project:

On the server-side of our web application we have REST API that is used to request information from our database and transferring that data to the browser.

One of the things we have to do while processing API requests is authenticating the user. Since we store client's financial data we have to make sure that our authentication process is secure and verified by tests.

In this case, we test if the current user can only access information that he has a permission to view or edit.

When writing tests you have to make sure to separate the function that is being tested from everything else and inject a mocks of all of the external functions. "Mocks" are used to imitate functionality of a needed system if that system cannot be initialised during test. In our case we will be mocking userAction that is used to authenticate user based on the session cookie from the browser. Because we don't use a browser for testing purposes we have to emulate the authentication process using a mock.

Here is the class that we are testing:

```
@Singleton
class ApiActions @Inject()(
                  userAction:UserAction,
                  ifaAction:IfaAction) extends Results
```

Here we are injecting userAction and ifaAction classes that are used to authenticate user. We are going to be overriding those function during our tests.

ApiActions function accesses our database and that means that for our tests to run successfully we will also need to mock our database. Our normal database uses a remote MYSQL server that we can't really use for testing purposes since we can potentially corrupt existing data during testing. All of the database settings are stored in the configuration file and because of that I wrote a script that would Inject a custom test configuration file that setups an in-memory database for testing purposes and wipes everything from memory after the tests are done.

Here is the test configuration file I have created for this purpose:

```
include "application.conf"

db {
 default {
  driver = org.h2.Driver
  url = "jdbc:h2:mem:~/h2db/ifatoolstest;MODE=MYSQL;"
  username = sa
  password = ""
 }
}
ebean.default = ["models.*"]
```

And here is a line of code that I have added to our build tool to replace the configurations file while we run tests:

```
  javaOptions in Test += "-Dconfig.file=conf/application.test.conf"
```

After that I created a function to populate our in-memory database with data that will be used for testing purposes:

```
def initTest = {
  OrganisationInitData(
    "TestOrg",
    AddressInitData("Test Street",
      Some("Test Line2"),
      "Test town",
      "Test county",
      "Test PostCode"),
    Seq({
      val fn = "Test UserWithoutMembers"
      UserInitData(
        fn,
        personInformation.genEmail(fn),
        gen.phoneNumber,
        "WithoutMember",
        "password",
        Seq(),
        Seq()
      )}, {
      val fn = "Test UserWithMembers"
      UserInitData(
        fn,
        personInformation.genEmail(fn),
        gen.phoneNumber,
        "WithMember",
        "password",
        Seq(roles.adviser),
        Seq(gen.memberInitData(spReader))
      )},{
      val fn = "Test UserWithMembers2"
      UserInitData(
        fn,
        personInformation.genEmail(fn),
        gen.phoneNumber,
        "WithMember2",
        "password",
        Seq(roles.adviser),
        Seq(gen.memberInitData(spReader))
      )})
  ).create()
}
```

In this case we are creating a couple of users with members and without members that should not have access to each others members, our purpose here is to test that users can only access their own data since in this case they don't have the necessary permissions to access other users data.
Now we can start writing our tests:

```scala
def findUserByUserName(userName:String) = {
  User.find.query().where().eq("userName",userName).findOne()
}



val users = new {
  val advisers = new {
    val withOwnedMember = findUserByUserName("WithMember")
    val withOwnedMember2 = findUserByUserName("WithMember2")
    val withOutMembers = findUserByUserName("WithoutMember")
  }
}


 "Test Database" must {
  "have organisation" in {
    val orgL  = Organisation.find.all().asScala.length
    orgL must be > 0
  }
  """have user "WithMember" with member""" in {
    Option(users.advisers.withOwnedMember) must not be empty
  }
  """have user "WithMember2" with member""" in {
    Option(users.advisers.withOwnedMember2) must not be empty
  }
  """have user "WithoutMember" without members""" in {
    Option(users.advisers.withOutMembers) must not be empty
  }
}
```

Here are the tests I wrote to ensure that our test database works properly and that the users were created.

```scala
"Member with members" must {
  val user = users.advisers.withOwnedMember
  "be Adviser" in {
    user.isIfa mustEqual true
  }
  "have members" in {
    user.getMembers.asScala.length must be > 0
  }
}
"Member without member" must {
  val user = users.advisers.withoutMembers
  "not have members" in {
    user.getMembers.asScala.length mustEqual 0
  }
}
```

Here we insure that our users have their members properly created.

And finally we are ready to test if the user has only the access to their own members here:

```
"GET members API" must {
          val noUser => Option[User] = (req)=>None
          val noIFA = (req)=>None
          val user = (req)=>Some(users.advisers.withOwnedMember)
          val ifa = (req)=>Some(users.advisers.withOwnedMember.getIfa)
          val noAccessAPI = new ApiActions(new UserAction(noUser), new IfaAction(noIFA))
   val ifaApi = new ApiActions(new UserAction(user),new IfaAction(ifa))
   "respond with Bad Request when no user in session" in {
      val result =                    noAccessAPI.get.members.apply(FakeRequest()).value.get.get
      result mustBe BadRequest
   }
   "show only members that belong to that adviser" in {
      val result =          ifaApi.get.members.apply(FakeRequest()).value.get.get
      val string = resultToString(result)
      val response = read[Seq[MemberData]](string)
      response.length mustBe 1
   }
}
```

Here we initialise our authentication mocks that return authenticated entity, in our case we bypass the authentication methods and inject entities directly:

```
val noUser = (req)=>None
val noIFA = (req)=>None
val user = (req)=>Some(users.advisers.withOwnedMember)
val ifa = (req)=>Some(users.advisers.withOwnedMember.getIfa)
```

Here we inject ApiActions classes with our mock authentication data:

```
val noAccessAPI = new ApiActions(new UserAction(noUser),new IfaAction(noIFA))

val ifaApi = new ApiActions(new UserAction(user), new IfaAction(ifa))
```

And finally we can start testing out API responses. Here we get a request result by mocking a request we expect it to respond with BadRequest:

```
"respond with Bad Request when no user in session" in {
  val result = noAccessAPI.get.members.apply(FakeRequest())
  result mustBe BadRequest
}
```

And here we again mock a request and after transforming the result data we expect to have a response of JSON with type of MemberData with only one object in it, proving that this user doesn’t have access to members of other user with members that we have initialised before:

```
"show only members that belong to that adviser" in {
  val result = ifaApi.get.members.apply(FakeRequest())
  val string = resultToString(result)
  val response = read[Seq[MemberData]](string)
  response.length mustBe 1
}
```

The last thing left to do for us is to actually run the tests.

I run tests by running a terminal command:
sbt test-only api.ApiTest (this command limits the scope of testing to api.ApiTest class)

And we get the following response:

```
[info] ApiTest:
[info] Test Database
[info] - must have organisation
[info] - must have user "WithMember" with member
[info] - must have user "WithMember2" with member
[info] - must have user "WithoutMember" without members
[info] Member with members
[info] - must be an Adviser
[info] - must have members
[info] Member without member
[info] - must not have members
[info] GET members API
[info] - must respond with Bad Request when no user in session
[info] - must show only members that belong to that adviser
[info] ScalaTest
[info] Run completed in 3 seconds, 943 milliseconds.
[info] Total number of tests run: 9
[info] Suites: completed 1, aborted 0
[info] Tests: succeeded 9, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[info] Passed: Total 9, Failed 0, Errors 0, Passed 9
[success] Total time: 6 s, completed 29-Jan-2019 20:45:10
```

All of our tests succeeded, ensuring us that our test setup is working properly and that GET members API request does not leak information and responds with BadRequest when the user is not authenticated.

Database Testing

Besides application testing I also need a way to test our database. The way I accomplish that is by connecting to the database directly and executing one-off queries.

Currently we use phpMyAdmin which is an administration tool for MySQL and MariaDB in a form of a web application.

The way I use phpMyAdmin is: I login with my username and password and navigate to the database I am working with. After that I user the SQL interface to execute one-off queries on the database. One-off queries are useful when debugging problems with the database or for creating one-off data for testing purposes.

Our application doesn't execute SQL queries directly, instead we use an object-relational mapping tool Java Ebean. Java Ebean lets us create Model classes and automatically generates appropriate SQL tables based on the Model class. The reason behind all of this is that you don't have to really think about MySQL while developing the application you just use Object-oriented code like in the rest of your program. An added bonus of using Java Ebean is that it also somewhat protects from MySQL injections because your SQL inputs are validated automatically.

The side-effect of using an object-relational mapping tool is: since you don't actually write any MySQL you don't know what the tool is doing precisely. Because of that sometimes errors emerge where a table is initialised not the way you wanted it to or something strange is can be happening while you pull or write data to database.

To help us debug our database operations we have a logging system in place that shows us all of the queries that Java Ebean sends to our database and if something goes wrong I can look up the specific query and manually test it.

Here is an example of one of the queries that Java Ebean generates:

```
[debug] io.ebean.SQL - txn[1021]
select distinct t0.id, t3.id, t3.first_name, t3.last_name, t1.id, t2.id, t2.first_name,
t2.last_name, t4.id, t5.id, t5.user_id, t5.member_id, t5.permission_level, t0.id
from member t0 j
oin person t3 on t3.id = t0.main_person_id
left join ifa t1 on t1.id = t0.ifa_id
left join user t2 on t2.id = t1.user_id
join organisation t4 on t4.id = t0.organisation_id
left join custom_member_permissions t5 on t5.member_id = t0.id
left join custom_member_permissions u1 on u1.member_id = t0.id
where t0.organisation_id = ?  and (u1.user_id = ?  or t1.user_id = ? )  order by t0.id;
--bind(1,3,3)
```

In this case I just would just copy the whole thing and input that into phpMyAdmin SQL console to see if the data that is returned from this query is what I need.

In other cases I can execute Create, read, update, delete or CRUD statements on the database to either verify that our database is working properly or input or change data for testing purposes. In this case I would input something like this into our phpMyAdmin MySQL interface:

Create:
```
INSERT INTO person (first_name, last_name)
VALUES (John, Smith)
```

Read:
```
SELECT first_name, last_name FROM members
WHERE first_name = “John”
ORDER BY last_name
```

Update:
```
UPDATE person
SET first_name = “Jane”
WHERE first_name=”John” AND last_name=”Smith”
```

Delete:
```
DELETE FROM person
WHERE first_name = “Jane” AND last_name = “Smith”
```

Graphical User Interface

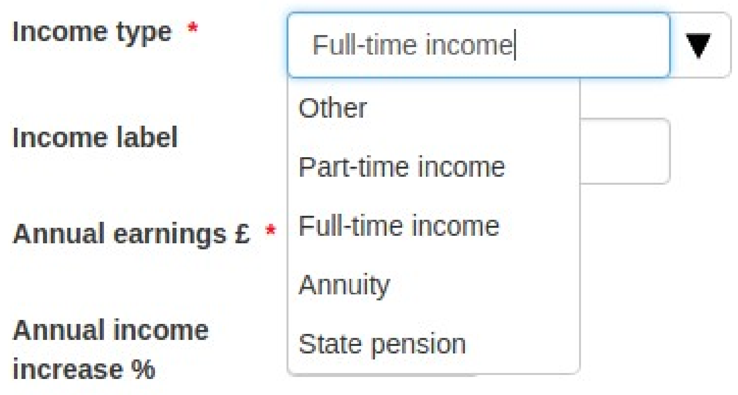
GUI is a complete necessity when it comes to modern applications. It dramatically enhances user experience of any application. Provides a way for people to work with complex programs without knowing how to code.

GUI in web applications is always a big challenge. Generally, you want to minimise the amount of client input data without impacting the flexibility of your web application.

The biggest challenge in web in creating a good GUI for a web application is browser compatibility. Web browsers are not created equally. You should start with establishing what kind of browsers you want to support and prepare accordingly. In my case we supported all modern browsers starting including Internet Explorer 9+ and all mobile device browsers (Always check if Safari supports your solution. In many cases, Safari browser features are inconsistent in comparison to other web browsers.).

There are three ways to implement any given GUI into your web application:
1. Using html/css browser features (Works if they go in-line with your browser compatibility specifications).
2. Using a poly-fill (simulates a behavior of a feature that is not supported by an older browser)
3. Implementing GUI from scratch (Preferred only when the first two options are not available. You don't want to reinvent the wheel).

During one of our meetings, we decided that we need an input field for income types that would give you options to choose from and provide a way to input your own income type name. I ended up implementing it from scratch.



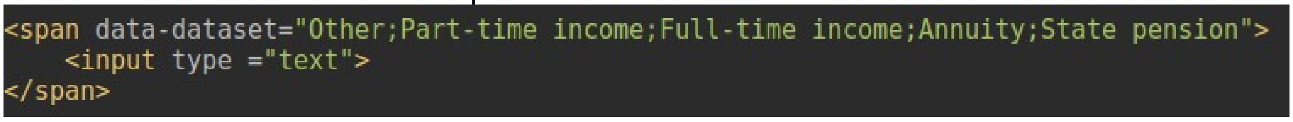
Finished input field

I started this assignment by looking for built in browser features. And good news I found a browser feature that does exactly this. HTML tag <datalist></datalist> does exactly what I was looking for. The problem is that it is not supported by “Safari” and “Internet explorer 9” browsers. That meant that I can’t use it since we have both “Safari” and “Internet Explorer 9” in our specifications.

The next step was looking for a poly-fill that works on both of those browsers. And I found a few polyfills that were decent. But the problem was, that they were not working quite as good as original data list. And as they say, the devil is in the details. Because of that, I decided to implement it myself using javaScript.

I used CoffeeScript to implement this feature. It is a functional programming like language that supports all modern JavaScript functionality and compiles to an old version of JavaScript that runs on all browsers. It works great with jQuery since it removes all of the unnecessary syntax that you end up with while using jQuery.

I made sure that this script is simple and easy to read. Here’s is the code that you need to include in the html file to add an input:



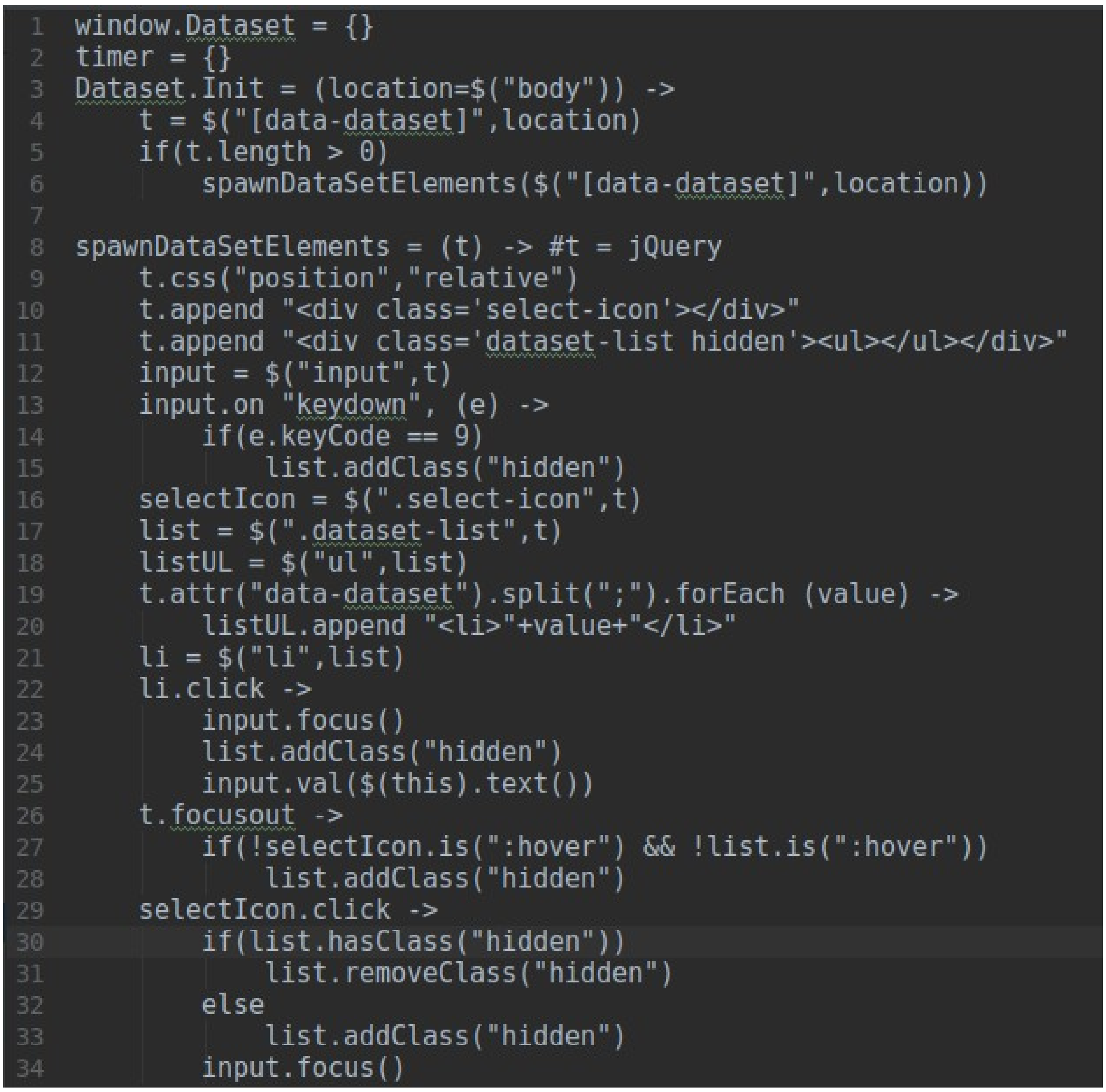
```html
<span data-dataset="Other;Part-time income;Full-time income;Annuity;State pension">
    <input type ="text">
</span>
```

This goes in your HTML. We use “data” parameter that works on all browsers to add all of the options for datalist. Everything else is done with JavaScript.

JavaScript code to initialise datalist when element is loaded:

Dataset.Init(jQueryElement)

And it changes the HTML so it would work exactly like the data list HTML tag.
CoffeeScript code:



```
window.Dataset = {}
timer = {}
Dataset.Init = (location=$("body")) ->
    t = $("[data-dataset]",location)
    if(t.length > 0)
        spawnDataSetElements($("[data-dataset]",location))

spawnDataSetElements = (t) -> #t = jQuery
    t.css("position","relative")
    t.append "<div class='select-icon'></div>"
    t.append "<div class='dataset-list hidden'><ul></ul></div>"
    input = $("input",t)
    input.on "keydown", (e) ->
        if(e.keyCode == 9)
            list.addClass("hidden")
    selectIcon = $(".select-icon",t)
    list = $(".dataset-list",t)
    listUL = $("ul",list)
    t.attr("data-dataset").split(";").forEach (value) ->
        listUL.append "<li>"+value+"</li>"
    li = $("li",list)
    li.click ->
        input.focus()
        list.addClass("hidden")
        input.val($(this).text())
    t.focusout ->
        if(!selectIcon.is(":hover") && !list.is(":hover"))
            list.addClass("hidden")
    selectIcon.click ->
        if(list.hasClass("hidden"))
            list.removeClass("hidden")
        else
            list.addClass("hidden")
        input.focus()
```

After I was done with the implementation I made sure to manually check whether the newly implemented input actually works.

I don’t interact with user interface testers and both the user interface implementation and testing is done by me.

In the end, I have implemented a custom input field for our web application that permits our users to choose an income type from options of different inputs or input their own income type based on their need. This reduces the amount of letters a user needs to type savings user’s time. This is especially relevant when it comes to mobile devices, because data input on mobile devices is generally slower in comparison to desktop devices. In our case, this is crucial, since our web application is designed to be used mostly with tablets.

Database management

Database setup

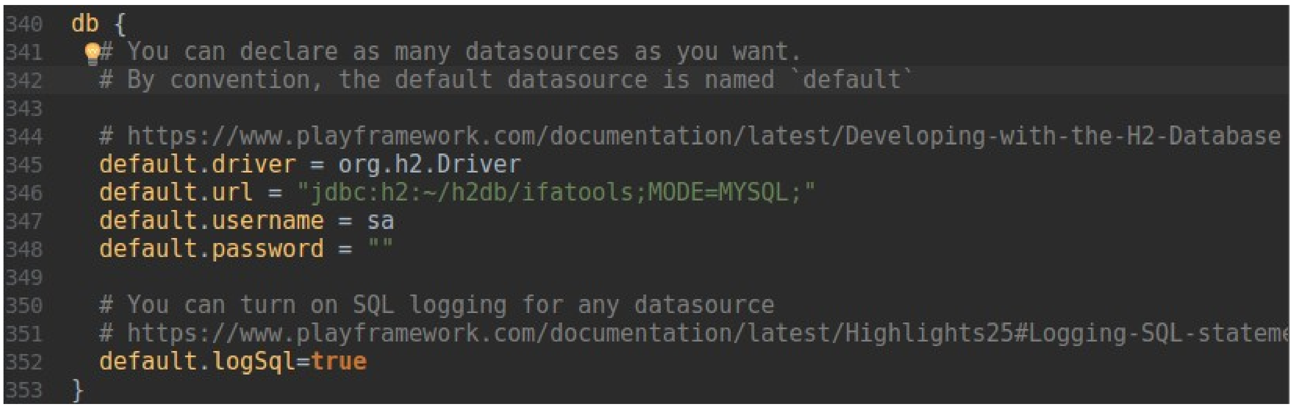
At the moment of writing this we use H2DB database for prototyping purposes. In the later stages of development we plan on using something different.

Play framework comes with support for H2DB integration and a plugin that you have to add to your SBT build configuration.

SBT - an open-source build tool for Scala projects.

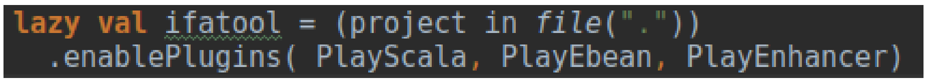
We connect to the database using Play framework’s Java Database Connectivity or JDBC API plugin. Play framework provides us with an easy way to do that through the configuration file of our app.

Here is the code that we include in our application.conf file:



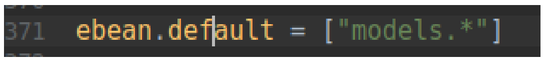
```
db {
  # You can declare as many datasources as you want.
  # By convention, the default datasource is named `default`

  # https://www.playframework.com/documentation/latest/Developing-with-the-H2-Database
  default.driver = org.h2.Driver
  default.url = "jdbc:h2:~/h2db/ifatools;MODE=MYSQL;"
  default.username = sa
  default.password = ""

  # You can turn on SQL logging for any datasource
  # https://www.playframework.com/documentation/latest/Highlights25#Logging-SQL-stateme
  default.logSql=true
}
```

We also add PlayEbean plugin that automatically setups Ebean for us:



```
lazy val ifatool = (project in file("."))
  .enablePlugins( PlayScala, PlayEbean, PlayEnhancer)
```

And lastly we include the direction to our database models:



```
ebean.default = ["models.*"]
```

All of this code is based on Play framework documentation.

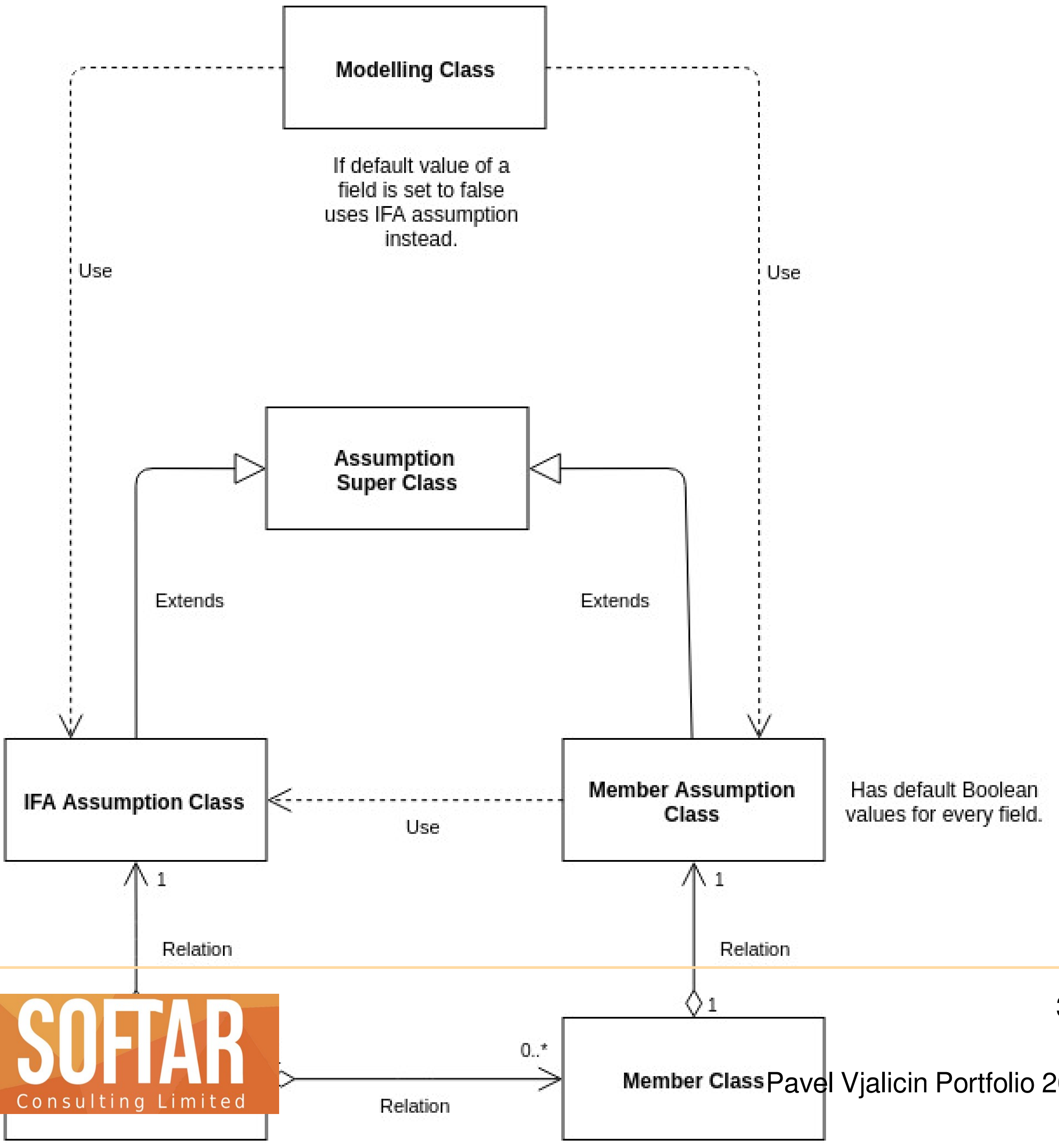
Database use-case

When it comes to database management we decided to use Ebean for Java, which is a object-relational mapping tool.

Object-relational mapping tools are used to convert data incompatible types using object-oriented programming languages. In this case it converts database data to Java objects that we can create, modify and delete using Java or Scala. This way our database code is less prone to human error.

During my work on this project I was tasked to create a system which lets IFA to assign default modelling assumptions as well as have the option to overwrite default assumptions with a set of custom assumptions. When dealing with such requests I usually start by gathering all the needed information from the specifications. If the specifications are not enough for me to implement the feature I contact the stakeholder and explain my plan: what do I need to do to implement this feature, how I will implement it and why I choose this particular method for implementation. Below I describe the way I have implemented this feature.

Here is a UML diagram to visualise an abstract version of our modelling assumptions structure:



I applied the above diagram to create an object-oriented solution. I showcase my implementation below.

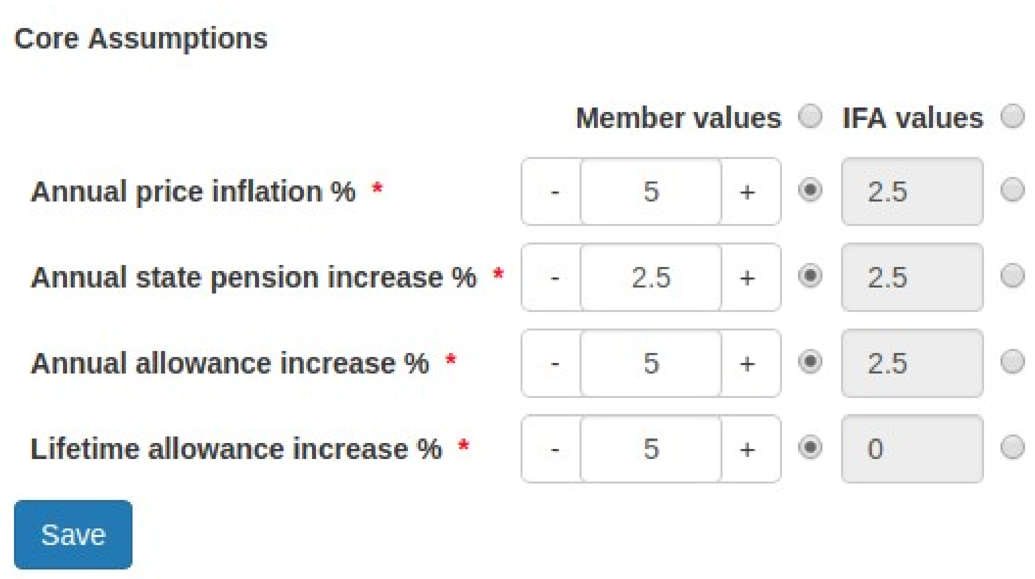
The web application I am currently working on is made to make retirement projections based on user data. In other to do that, we store all kinds of assumptions that will be used in order to make the projection.

As you can see from the diagram, in order to improve user experiece we have two types of assumptions. **I**ndependent **F**inancial **A**dviser assumptions and member assumptions.

The IFA assumptions are assumptions that are pre-set by an IFA. Those assumptions are used to create default member assumptions and are used for general circumstances.

For the unusual cases or unique retirement plans default IFA assumptions might not be accurate enough to use for the retirement modelling. Instead IFA might decide to overwrite those with Member Assumptions.

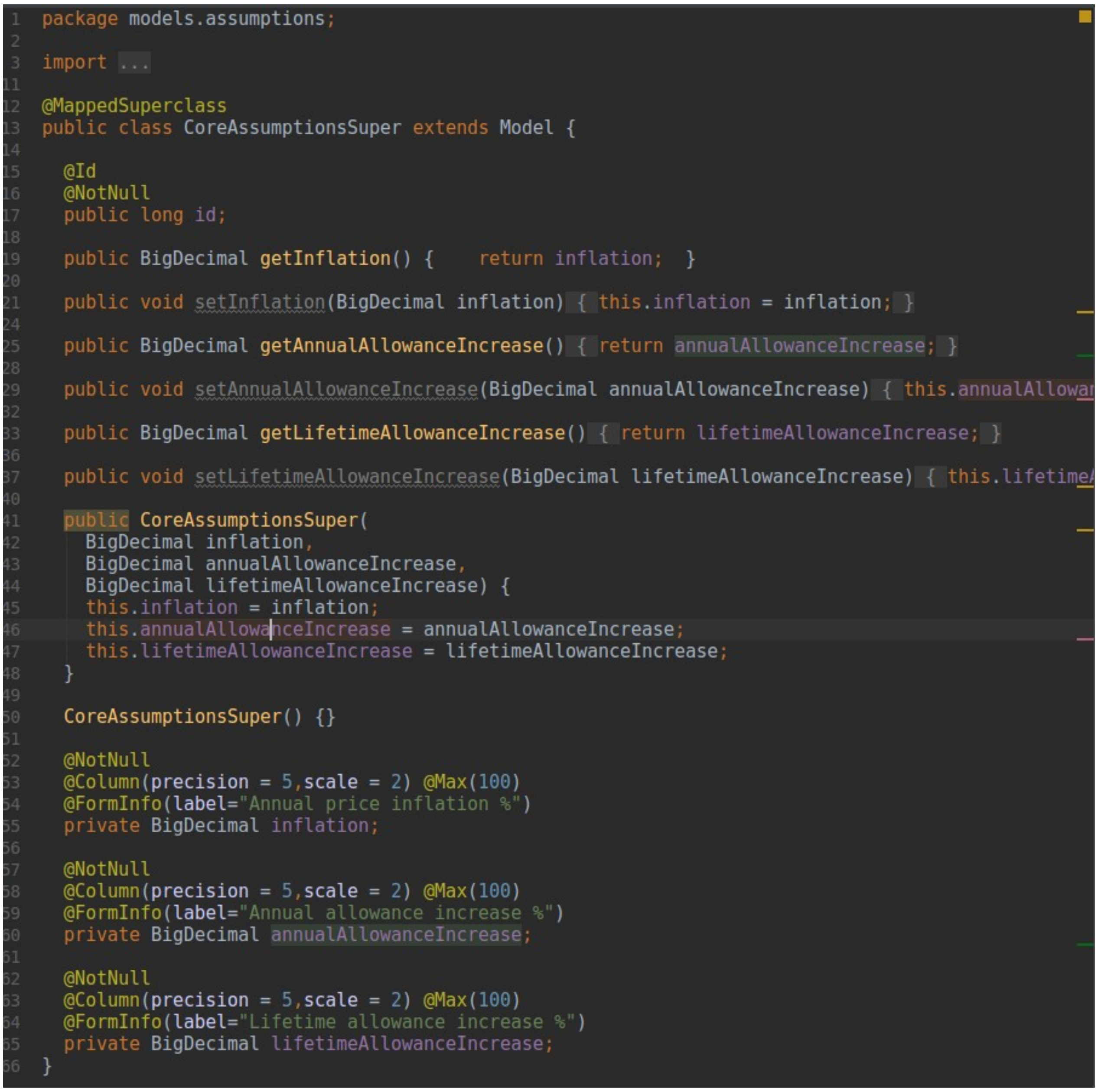
Here is an example of assumption user interface:



Core Assumptions

	Member values		IFA values	
Annual price inflation % *	- 5 +	●	2.5	○
Annual state pension increase % *	- 2.5 +	●	2.5	○
Annual allowance increase % *	- 5 +	●	2.5	○
Lifetime allowance increase % *	- 5 +	●	0	○

And here is how CoreAssumptionsSuper.java looks like:



```java
package models.assumptions;

import ...

@MappedSuperclass
public class CoreAssumptionsSuper extends Model {

  @Id
  @NotNull
  public long id;

  public BigDecimal getInflation() {    return inflation;  }

  public void setInflation(BigDecimal inflation) { this.inflation = inflation; }

  public BigDecimal getAnnualAllowanceIncrease() { return annualAllowanceIncrease; }

  public void setAnnualAllowanceIncrease(BigDecimal annualAllowanceIncrease) { this.annualAllowa

  public BigDecimal getLifetimeAllowanceIncrease() { return lifetimeAllowanceIncrease; }

  public void setLifetimeAllowanceIncrease(BigDecimal lifetimeAllowanceIncrease) { this.lifetime

  public CoreAssumptionsSuper(
    BigDecimal inflation,
    BigDecimal annualAllowanceIncrease,
    BigDecimal lifetimeAllowanceIncrease) {
    this.inflation = inflation;
    this.annualAllowanceIncrease = annualAllowanceIncrease;
    this.lifetimeAllowanceIncrease = lifetimeAllowanceIncrease;
  }

  CoreAssumptionsSuper() {}

  @NotNull
  @Column(precision = 5,scale = 2) @Max(100)
  @FormInfo(label="Annual price inflation %")
  private BigDecimal inflation;

  @NotNull
  @Column(precision = 5,scale = 2) @Max(100)
  @FormInfo(label="Annual allowance increase %")
  private BigDecimal annualAllowanceIncrease;

  @NotNull
  @Column(precision = 5,scale = 2) @Max(100)
  @FormInfo(label="Lifetime allowance increase %")
  private BigDecimal lifetimeAllowanceIncrease;
}
```

This is a class that is inherited from CoreAssumptions class and IfaCoreAssumptions Class.

Here we use the following annotations based on Ebean specification:

@MappedSuperclass – Annotation used for indicating that this is a class that we will use for inheritance process.

@Id – Annotation used for generating ID index that will be used to find needed object in CoreAssumption and IfaCoreAssumption tables.

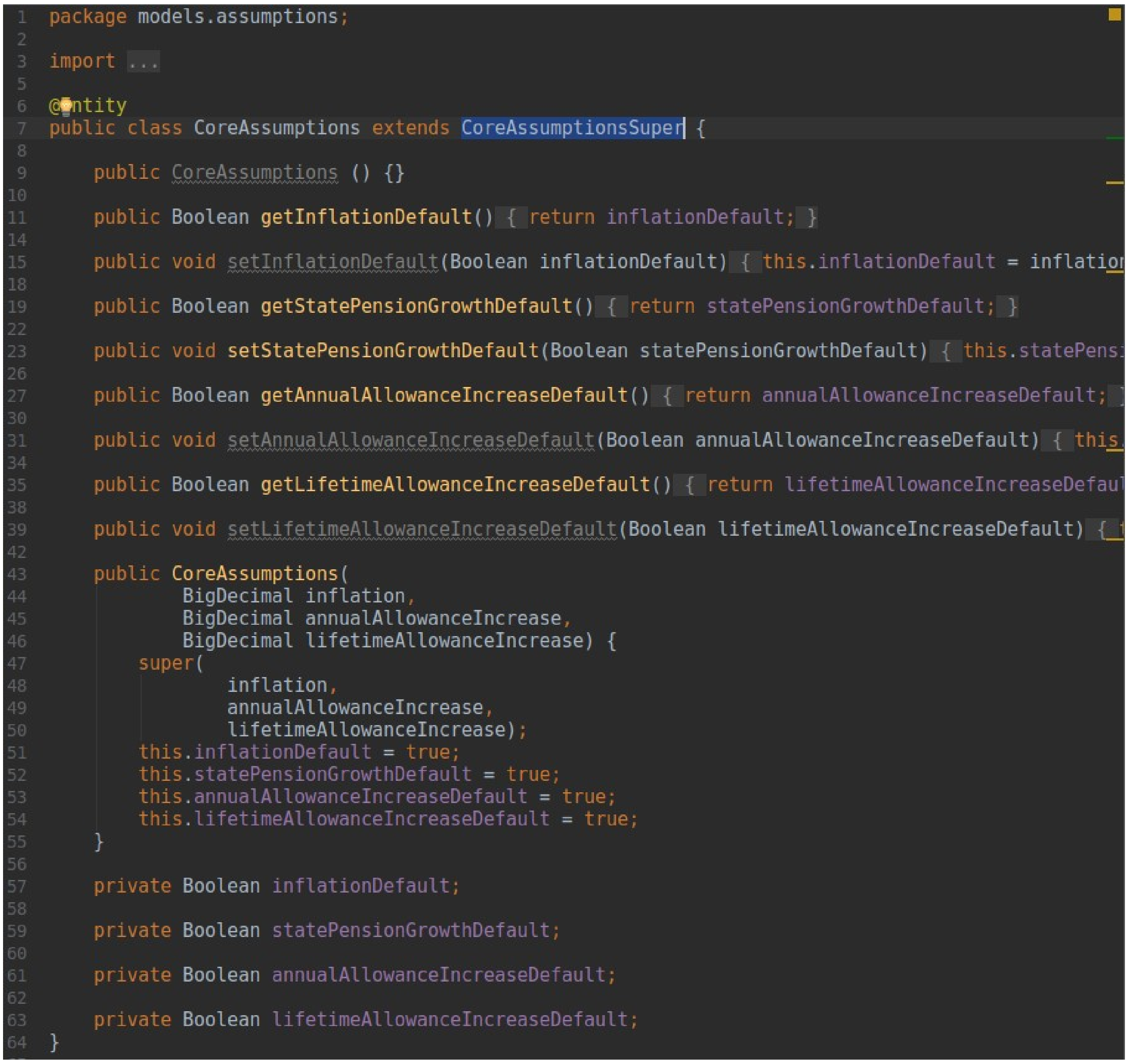
@NotNull – Annotation that doesn't allow the field to be empty. We also use this annotation to generate our Input fields. If @NotNull annotation is present we add required attribute to that input fields HTML code.

@Column – Annotation that specifies the size of data in the field. We also use this annotation combined with @Max annotation to determine the step, min value, max value of a field while generating HTML input field.

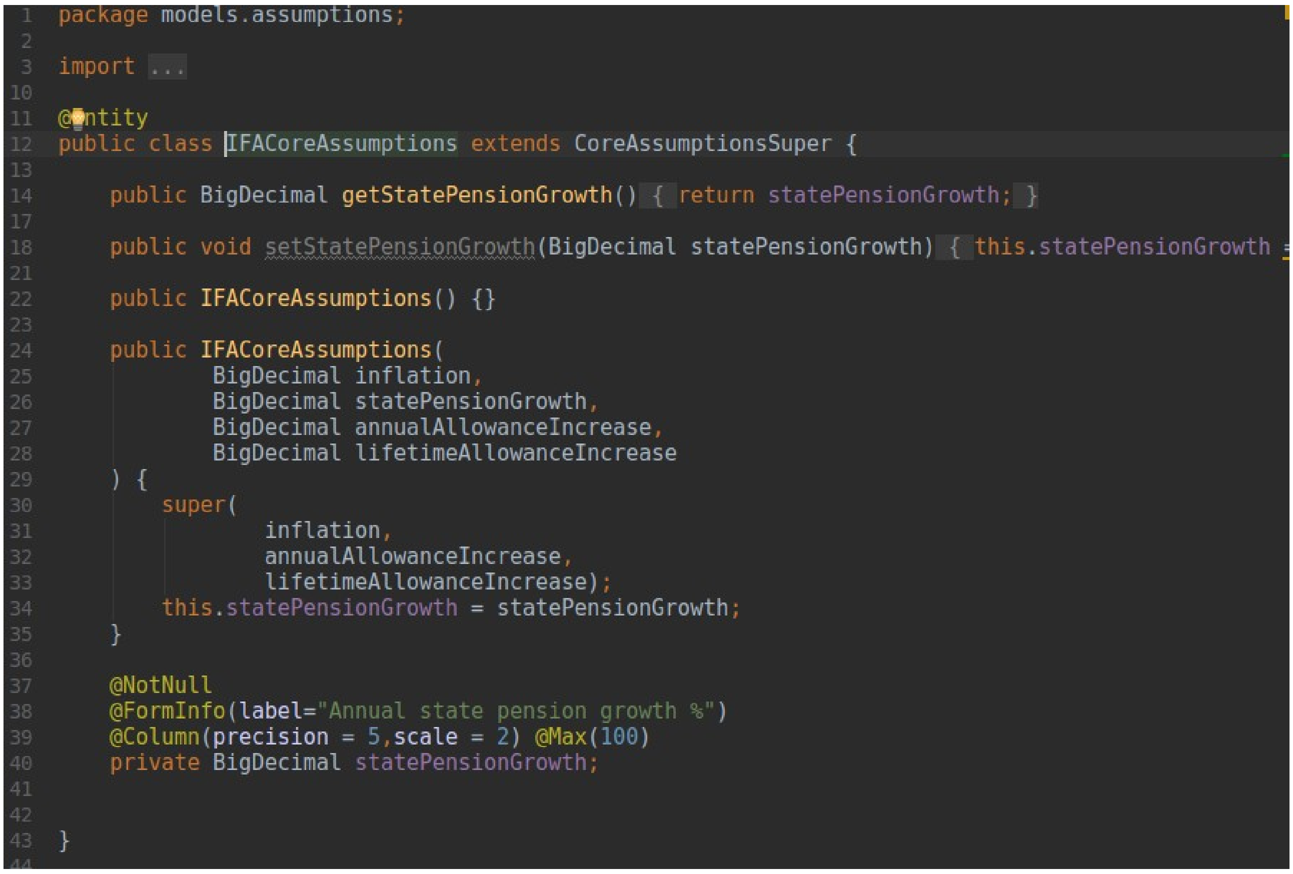
@Max – Annotation that is used for generating our HTML input field to declare a Max value for the specific field.

@FormInfo – Annotation that is used with a system that I designed and implemented. We use a combination of Java reflection and Scala macro to generate a proper field based on this and other annotations for our front-end HTML. The HTML is generated by looking up a field and based on the field's type and field's annotations it automatically generates a proper front-end HTML input field.

And here is screenshots of IFACoreAssumptions and CoreAssumptions:



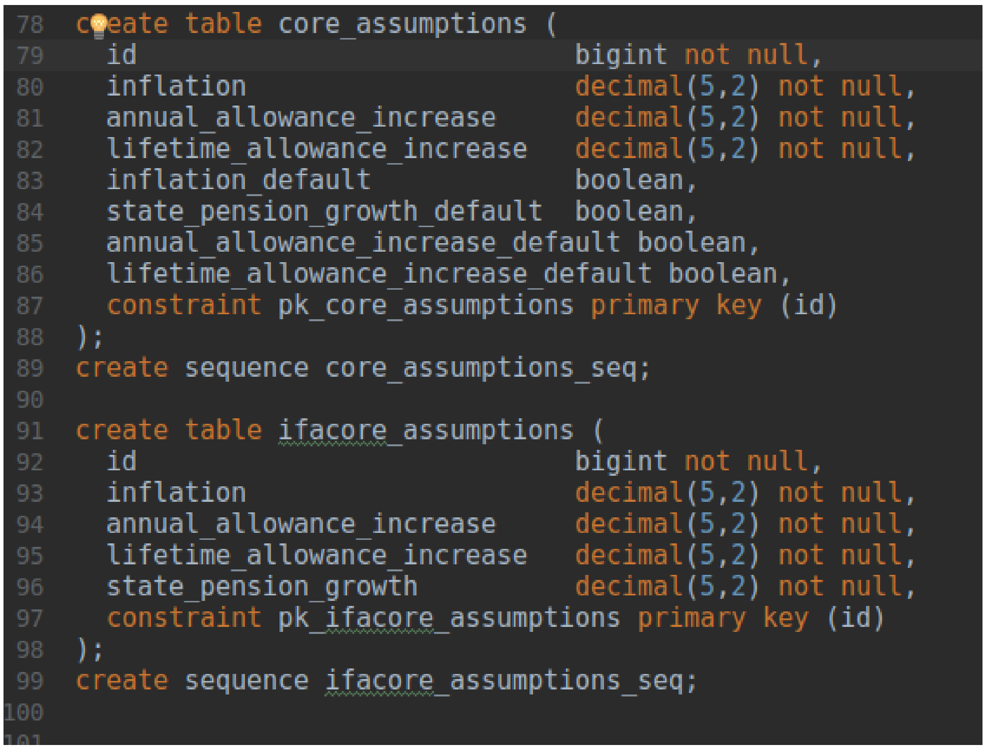
```
package models.assumptions;

import ...

@Entity
public class CoreAssumptions extends CoreAssumptionsSuper {

    public CoreAssumptions () {}

    public Boolean getInflationDefault() { return inflationDefault; }

    public void setInflationDefault(Boolean inflationDefault) { this.inflationDefault = inflatio

    public Boolean getStatePensionGrowthDefault() { return statePensionGrowthDefault; }

    public void setStatePensionGrowthDefault(Boolean statePensionGrowthDefault) { this.statePens

    public Boolean getAnnualAllowanceIncreaseDefault() { return annualAllowanceIncreaseDefault;

    public void setAnnualAllowanceIncreaseDefault(Boolean annualAllowanceIncreaseDefault) { this.

    public Boolean getLifetimeAllowanceIncreaseDefault() { return lifetimeAllowanceIncreaseDefau

    public void setLifetimeAllowanceIncreaseDefault(Boolean lifetimeAllowanceIncreaseDefault) {

    public CoreAssumptions(
            BigDecimal inflation,
            BigDecimal annualAllowanceIncrease,
            BigDecimal lifetimeAllowanceIncrease) {
        super(
                inflation,
                annualAllowanceIncrease,
                lifetimeAllowanceIncrease);
        this.inflationDefault = true;
        this.statePensionGrowthDefault = true;
        this.annualAllowanceIncreaseDefault = true;
        this.lifetimeAllowanceIncreaseDefault = true;
    }

    private Boolean inflationDefault;

    private Boolean statePensionGrowthDefault;

    private Boolean annualAllowanceIncreaseDefault;

    private Boolean lifetimeAllowanceIncreaseDefault;
}
```



```java
package models.assumptions;

import ...

@Entity
public class IFACoreAssumptions extends CoreAssumptionsSuper {

    public BigDecimal getStatePensionGrowth() { return statePensionGrowth; }

    public void setStatePensionGrowth(BigDecimal statePensionGrowth) { this.statePensionGrowth

    public IFACoreAssumptions() {}

    public IFACoreAssumptions(
            BigDecimal inflation,
            BigDecimal statePensionGrowth,
            BigDecimal annualAllowanceIncrease,
            BigDecimal lifetimeAllowanceIncrease
    ) {
        super(
                inflation,
                annualAllowanceIncrease,
                lifetimeAllowanceIncrease);
        this.statePensionGrowth = statePensionGrowth;
    }

    @NotNull
    @FormInfo(label="Annual state pension growth %")
    @Column(precision = 5,scale = 2) @Max(100)
    private BigDecimal statePensionGrowth;


}
```

@Entity – Annotations is used to mark this class as an Entity for Ebean. This way Ebean knows to generate an SQL query to create this class with specified fields.

Here is the automatically generated sql code to generate the tables:



```
create table core_assumptions (
  id                             bigint not null,
  inflation                      decimal(5,2) not null,
  annual_allowance_increase      decimal(5,2) not null,
  lifetime_allowance_increase    decimal(5,2) not null,
  inflation_default              boolean,
  state_pension_growth_default   boolean,
  annual_allowance_increase_default boolean,
  lifetime_allowance_increase_default boolean,
  constraint pk_core_assumptions primary key (id)
);
create sequence core_assumptions_seq;

create table ifacore_assumptions (
  id                             bigint not null,
  inflation                      decimal(5,2) not null,
  annual_allowance_increase      decimal(5,2) not null,
  lifetime_allowance_increase    decimal(5,2) not null,
  state_pension_growth           decimal(5,2) not null,
  constraint pk_ifacore_assumptions primary key (id)
);
create sequence ifacore_assumptions_seq;
```

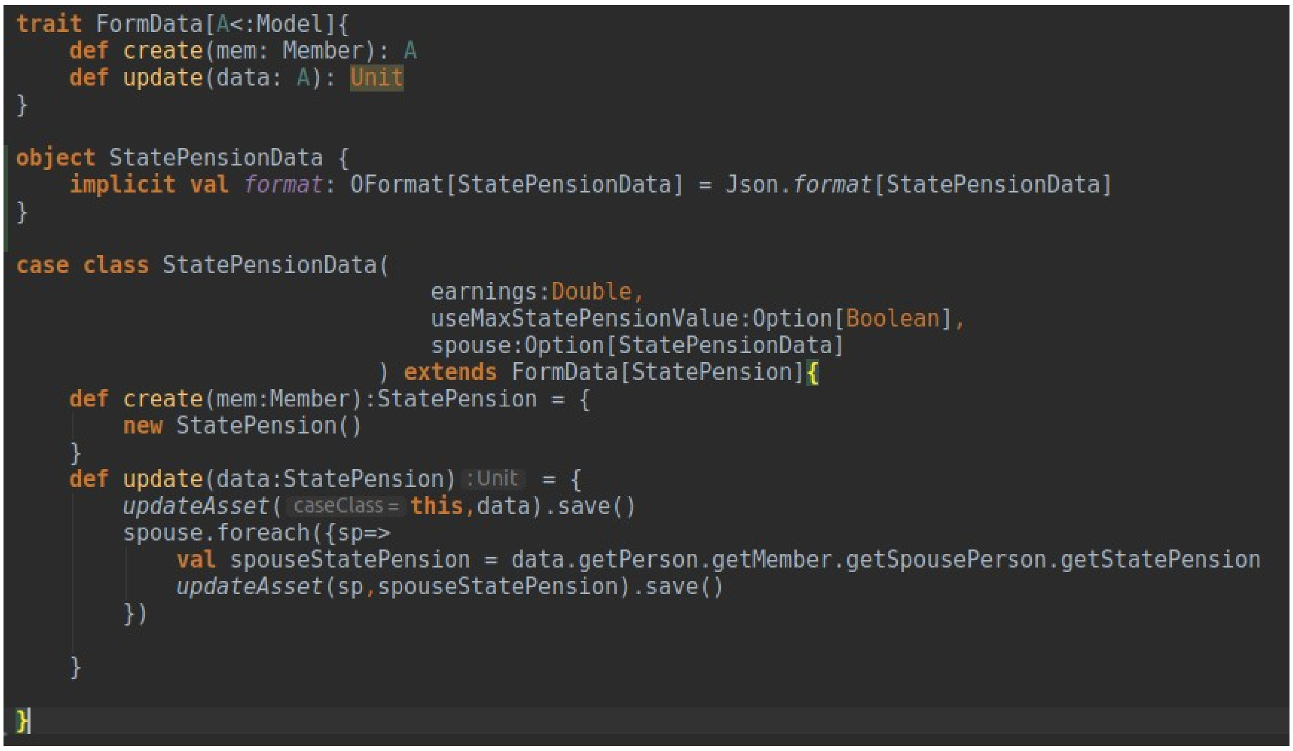
Note: This is not the whole code that is being generated by Ebean.

Data transformation

In most cases the data that you receive and the data that you need to store are in different data formats. Because of that you need to transform the data from one format to the other.

In our case, when it comes to client-server communication we receive the data as JSON, transform it to Scala class, save it to h2db model (similar to MYSQL).

Here is an example of this process:



```scala
trait FormData[A<:Model]{
    def create(mem: Member): A
    def update(data: A): Unit
}

object StatePensionData {
    implicit val format: OFormat[StatePensionData] = Json.format[StatePensionData]
}

case class StatePensionData(
                               earnings:Double,
                               useMaxStatePensionValue:Option[Boolean],
                               spouse:Option[StatePensionData]
                            ) extends FormData[StatePension]{
    def create(mem:Member):StatePension = {
        new StatePension()
    }
    def update(data:StatePension) :Unit  = {
        updateAsset( caseClass = this,data).save()
        spouse.foreach({sp=>
            val spouseStatePension = data.getPerson.getMember.getSpousePerson.getStatePension
            updateAsset(sp,spouseStatePension).save()
        })

    }

}
```

To do this we use play framework's json library.

First of all we have an Interface that takes Model type and has two definitions: create and update.

We call Json.format function with the type of our class, which is a macro (meta programming) function that converts the JSON that we received to Scala class.

After that we implement FormData interface and declare the logic behind Model creation and update.

In this case, in update definition of StatePensionData we use a function called updateAsset which I implemented. UpdateAsset function is a meta programming function that reads case class values, in this case: earnings, userMaxStatePensionValue, and automatically assigns the values to our model class.