# VICTORIA UNIVERSITY OF WELLINGTON
*Te Whare Wānanga o te Ūpoko o te Ika a Māui*

# School of Engineering and Computer Science
*Te Kura Mātai Pūkaha, Pūrorohiko*

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

## PitchHub - A Collaboration Platform for Innovators

Michael Winton

Supervisor: Dr. Kris Bubendorfer

Submitted in partial fulfilment of the requirements for
ENGR489 - Bachelor of Engineering.

### Abstract

The ability to connect innovative ideas to people and resources is an essential component of the innovation process. This project is concerned with empowering the innovation community with an online collaboration system that is simultaneously useful to all actors in the innovation ecosystem while ensuring that all sensitive IP shared is stored in a secure manner. TODO

# Contents

# Figures

# Chapter 1

# Introduction

## 1.1 Motivation

## 1.2 Project Objectives

## 1.3 Contributions

## 1.4 Outline

# Chapter 2

# Background

This chapter aims to provide background on the innovation ecosystem and contextualise where in this landscape PitchHub fits in. First, this chapter introduces the concept of an innovation community, and explores the roles that are present within this community. Second, this chapter discusses innovation online and what security issues are raised in this environment. Third, this chapter describes the current collaborative platforms being used in the innovation space and establishes where each stands within the role taxonomy. Fourth, this chapter concludes by discussing the importance of database security and provides background on Shamir's Secret Sharing scheme.

## 2.1 Innovation Community

In this world of constant communication the creation of ideas is an activity no longer isolated to inventors or researchers. It has evolved into what can instead be seen as a collaborative effort from a diverse community of actors. Von Hippel defined innovation communities as "nodes consisting of individuals or firms interconnected by information transfer links which may involve face-to-face, electronic, or other communication" [1]. The world of innovation today now includes actors from increasingly disparate domains who are able to contribute their unique capabilities to the innovation process [2]. The influx of unique skills being mixed into the innovation community has resulted in more unique opportunities for innovation being made possible. Therefore to encourage innovation is to also encourage the innovation community as well.

## 2.2 Common Roles in Innovation

The process of driving an idea from its conceptualisation to its realisation commonly requires a variety of actors. These actors as a team bring together the knowledge, skills and resources required to action the idea's fulfilment [3]. For example, the Apple ][ came to being with Steve Wozniak providing the technical knowledge and skills, Steve Jobs providing the project goals and marketing drive, and Mike Markulla providing the resources to finance the operation [4]. This is a recurring pattern, where subgroups of a team producing an innovative product or service have different responsibilities in regards to the end product or service's realisation. To formalise these common responsibilities Callaghan Innovation has identified four distinct roles that are embodied within the innovation process:

- Challenger
- Enabler

3

- Solver

- Facilitator

Challengers provide the idea or problem to be solved in order to realise a business opportunity. Enablers provide the resources required to action the innovation, this may be in terms of man-power, assets or financing. Solvers provide the answer to the idea or problem presented by the Challenger(s). Facilitators provide the connections to drive the innovation's execution, this may be in terms of connecting the idea to other people or helping the idea gain reputation. In most cases these roles are too large for one person to embody them all. To continue with the Apple ][ example, we may categorise Steve Jobs as a Challenger, asking why computers can't serve the consumer market, Steve Wozniak can be seen as an Enabler and Solver, as he both designed the Apple ][ and built them, and Mike Markulla, can be regarded as an Enabler and Facilitator, as he financed the production and also lent his reputation to the product. By recognising these roles and the interplay required between them to action an innovative idea, we can then also recognise that platforms which seek to encourage innovation must also provide functionality for the collaboration/networking between actors fulfilling these roles.

## 2.3 Innovation Online

One of the key technologies in the modern world is the internet. With the internet, communication and knowledge sharing has never been more accessible or easy. Naturally, the internet networking that is now an integral part of the innovation process has been enhanced by the internet with the reach it affords. This increased reach has allowed innovation communities to capitalise on the larger source of innovative potential and knowledge [5].

## 2.4 Security, Privacy, and Trust in Online Communities

The nature of bringing the innovation process online consequently involves bringing what could be commercially sensitive information online also. In recent times there has been a growing trend of online security attacks where user data has been compromised. Given this reality, there is a large amount of trust involved where users are relying on the platforms they are inputting their sensitive data into to take precautions to keep this data safe. Research within the domain of Economics has shown the importance of trust in a business context: "without trust, risk is paralyzing; transactions simply do not take place" [6], this notion is similarly applicable in the online innovation space where users are transacting in intellectual property and skills rather than money. This trust enables users to operate in an unsafe environment. It can be regarded as unsafe as users little power over how their data is stored and once it is stored they have no control over who can view. It is therefore important for platforms to make good on this trust, first by, implementing safeguards against security threats and second by, providing functionality that gives users control over the visibility of their data.

Unfortunately, the security of online communities does not solely depend on their technical security. As explored by Johnson et al. in their work regarding Facebook and privacy [7] social networks also face the problem of managing insider threat. Insider threat is where users inappropriately share content with members on the network. This problem is raised by the lack of or under use of privacy controls. In a platform where commercially sensitive information is the content at stake it is important that the platform enforce (or encourage) the use of these privacy controls. A study conducted by Shin explores the constructs of

security, privacy, and trust in social networks, his findings affirmed the above discussion, concluding that security and privacy play vital roles in developing trust from the users [8].

## 2.5   Related Work

Naturally, a platform that aims to facilitate collaboration for purposes of innovation should also be empowering the users embodying the roles in Section 2.2 to network and collaborate with each other. In this section we explore the current solutions being used to facilitate collaboration and discuss how each works in relation to supporting collaboration between these roles.

**IdeaForge** [9] is a collaborative innovation platform that supports the Challenger, Enabler and Solver roles. In it's own parlance IdeaForge is described as a three-sided marketplace where users can provide "ideas, time/skills or cash/resources". The main aim for this platform is to facilitate any-time/anywhere collaboration within the global innovation community. Additionally, IdeaForge provides some visibility settings for ideas such that they may be scoped as visible publicly or members only. IdeaForge does not provide explicit support for the Facilitator role, therefore ideas being hosted on IdeaForge require external facilitation. IdeaForge can be regarded as the most similar to PitchHub in spirit as it serves many of the roles identified and provides scoping functionality.

**Assembly** [10] is a collaborative platform that implicitly supports Challenger, Enabler, Solver, and Facilitator roles. The implicit collaboration support is facilitated through it's forum-like structure, where any of these roles may contribute and network. This is adequate however as established in Hautz et al.'s study of online innovation communities, they point out that innovation communities have a "very specific purpose and therefore requires a special kind of user participation and interaction" [5], this is why explicit support of collaboration between these roles is preferred over implicit support. Important to note is that Assembly is organised around groups rather than ideas, however these groups may be working on one or more ideas. Assembly's recommender system functionality, where users get recommended groups they may be interested in, illustrates how Assembly itself can be seen as carrying out the Facilitator's role. PitchHub and Assembly differ on focus, where PitchHub focuses on the idea Assembly focuses on the groups, this structure while applicable to the innovation space is less directed towards the immediate fulfilment of ideas and more for general collaboration.

**AngelList** [11] and **Enterprise Angels** [12] are examples of online platforms for investors (a subset of of Enablers) looking to fund businesses. Crowd funding and microequity platforms such as **Kickstarter** [13], **Indiegogo** [14] and **PledgeMe** [15] are becoming increasingly viable sources of funding for innovation. These platforms are primarily for Solvers looking to seed their solutions, and Enablers looking to get return on their investments. An interesting phenomenon of these platforms is the social "hype" that is sometimes garnered around many of the products/services launched on these platforms. While the solicitation of funds is not a primary goal of PitchHub the inherent socialness of these funding platforms is directly comparable.

Inevitably large social networks have also been used in the innovation space as platforms to help facilitate collaboration. Examples include **LinkedIn** [16] being used by New Zealand Healthcare Innovation [17], **Facebook** [18] being used in the Great New Zealand Science

Project [19], and **Google Groups** [20] being used in the National Science Challenges [21]. These platforms have the inherent benefit of convenience as many people in the innovation ecosystem are already members of these networks. Beyond the lack of explicit support for collaboration between the roles identified in Section 2.2 these platforms also suffer from lack of (used) privacy controls. This leads to what is not a conducive environment for users wishing to discuss commercially sensitive information. These re-purposed examples of social networks are in stark contrast to PitchHub's goal of facilitating collaborative innovation in a secure manner.

Overall, the proliferation of online networks has been a boon for communities, enabling unprecedented reach. The innovation community is no different and has benefited greatly from these networks, however as demonstrated in the above investigation these networks lack features which serve the directed collaboration between roles within the innovation community and also lack (used) privacy control functionality.
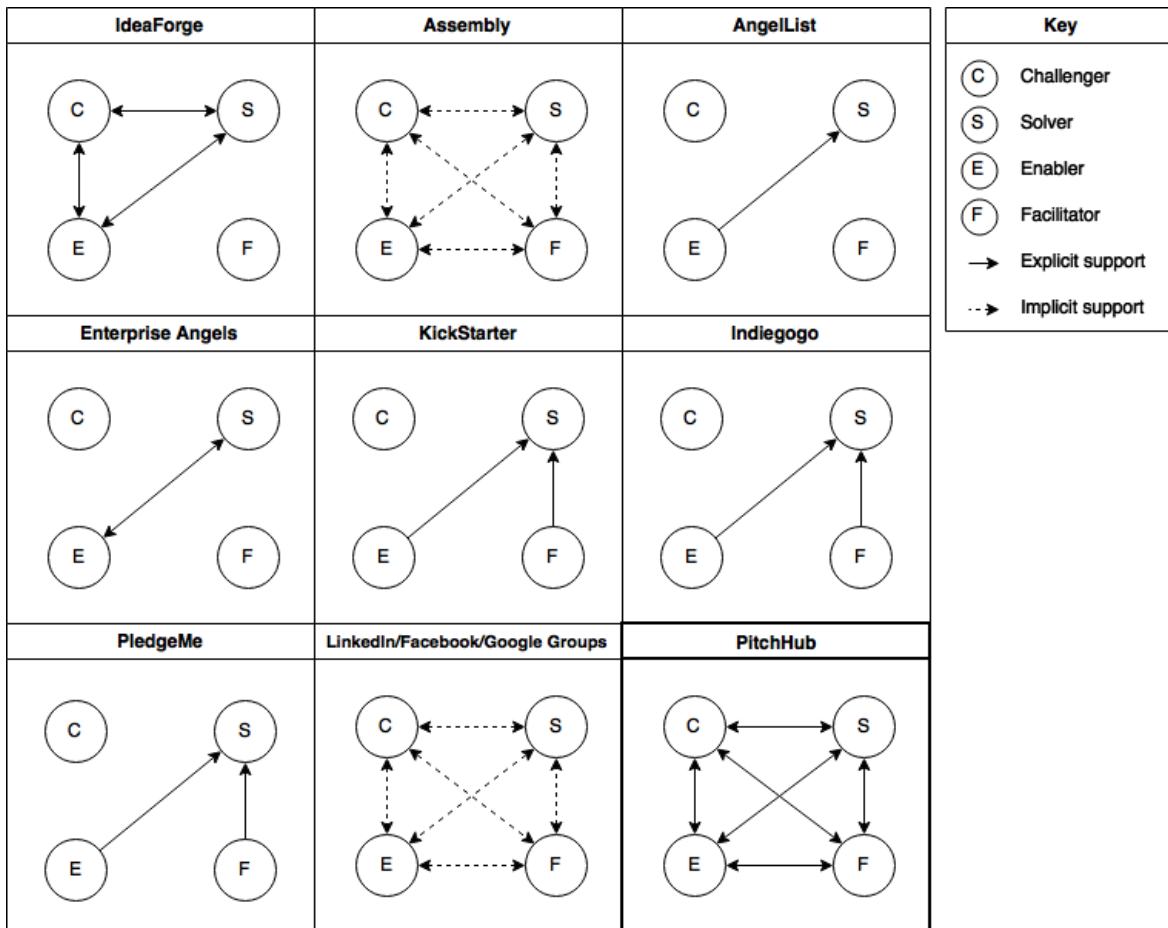


Figure 2.1: All collaborative platforms investigated do not provide explicit collaboration support between the all roles identified in Section 2.2. PitchHub aims to fix this by supporting networking between all roles.

As demonstrated in Figure 2.1 PitchHub seeks to fill the gap in the online innovation collaboration space by being a platform that supports explicit collaboration between all roles. Through this PitchHub aims to systematically build valuable business connections centred around the ideas. PitchHub has also been designed to incorporate features from the investigated platforms. From FaceBook, PitchHub extends its privacy control model with scope of

disclosure negotiation, this is discussed later in Section TODO.

## 2.6  Database Security

Securely storing data is a large and increasingly important research area to modern society. In recent years companies like Sony, Apple and Adobe have experienced data breaches resulting in compromised user data. In a system like PitchHub where commercially sensitive information is being handled extreme care must be taken to ensure its safety.

### 2.6.1  Threshold Security Schemes

Secret Sharing schemes are a type of Threshold Security Scheme where a piece of data is split into $n$ secret shares. To retrieve the original data a threshold of $k$ of $n$ shares ("$k, n$") must be met before the original piece of data can be decrypted. A fictional scenario that describes this concept is where the code for launching a nuclear missile is split between three officers, to launch the missile all three officers must be present to reconstruct the launch code. Each share in isolation cannot be used to launch the missile, nor can it be used to infer the original code. This "3, 3" example is somewhat contrived but it showcases the fundamentals of secret sharing in that: the secret can be recovered given the threshold is met and the secret is indeed secret given any combination of $t$ shares less than $k$.

### 2.6.2  Shamir's Secret Sharing Scheme

Shamir's secret sharing [22] is a threshold scheme based on polynomial interpolation.

$$q(x) = a_o + a_1 x + ... + a_{k-1} x^{k-1}$$

The fundamental idea is that given a "$k, n$" scheme a random polynomial of degree $k$-1 may be generated where the secret $D$ is hidden as term $a_o$. A set of $n$ points may then be constructed from this polynomial and shared to $n$ secret keepers. To recover the secret $D$ the process requires a subset of $k$ secret shares to calculate the coefficients of the polynomial using interpolation. With this a system is able to recover the secret $D$ at term $a_o$.

Shamir's Secret Sharing scheme effectively allows a system to share secrets to $n$ secret keepers while maintaining each secret's safety as a long as a threshold of $k$ malicious secret keepers is not met. Besides being secure, Shamir's Secret Sharing scheme holds a number of other useful properties such as being minimal, extensible and dynamic [22]. The scheme is minimal in that the the size of each share does not exceed the size of the original secret. The scheme is extensible in that $n$ may be increased by giving new secret keepers new points from the polynomial. The scheme is dynamic in that the shares may be changed by modifying the polynomial but retaining the term $a_o$.

As discussed in a number studies the simplicity of Shamir's Secret Sharing scheme does present limitations that have the potential for being exploited [23][24]. First, complete trust is given to the system who is dealing the secret shares, if an error occurs the secret may be irrecoverable therefore the system dealing the shares can be seen as a single point of failure. Second, Shamir's Secret Sharing scheme cannot detect secret keepers that cheat by supplying wrong shares, this brings about the problem where if $k$ members are compromised not only will the secrets be recoverable by the malicious party but the compromised members may be used to supply wrong secret shares to the system, effectively denying the first party system of recovering secrets. To combat these exploits Shamir's Secret Sharing may be extended with signature schemes [25][26] and verification schemes that do not assume the party dealing the shares is honest or infallible [27][28].

### 2.6.3   Threshold Scheme Related Work

Should I have this?

# Chapter 3

# Requirements

## 3.1 Previous Work

To contextualise the requirements identified this section first introduces Callaghan Innovation's previous work on PitchHub. Callaghan Innovation began work on the idea of PitchHub in 2013. Since this time Callaghan Innovation has discerned what functionality a collaborative innovation platform like PitchHub needs to fulfil it's aim of driving innovation by connecting the roles identified in Section 2.2.

### 3.1.1 Pitch Cards Conceptualisation

As discussed in Chapter 2 the innovation community has very specific purpose and therefore requires special kind of user participation/interaction [29]. The interaction which PitchHub facilitates is orientated around ideas. The notion of an idea is very general and ambiguous and to be able to convey it clearly requires precision. To facilitate this PitchHub structures ideas in the form of 'Pitch Cards'. Pitch Cards describe ideas in basically the same way CRC cards describe classes, by teasing out the fundamentals and leaving out the cruft. Callaghan Innovation designed Pitch Cards to explicitly support collaboration between the roles around a Pitch Card. To do this each Pitch Card is made up of a number of Pitch Points which relate to a role. Figure 3.1 displays Callaghan Innovation's original Pitch Card view.



Figure 3.1: Callaghan Innovation's design for a Pitch Card describes an idea with the following Pitch Points: Value Proposition (Any role, describing the idea's value), Business Opportunity (Challenger), Resources (Enabler), Solutions (Solver), Facilitation (Facilitator), Collaborative Decision (Any role, voting on the idea).

### 3.1.2 Collaboration Conceptualisation

Collaboration on PitchHub is actioned through users making suggestions and comments on these Pitch Points. This is ultimately how PitchHub offers the explicit support of collaboration between roles. Collaboration on PitchHub can be seen as a negotiation, where Pitch Card initiator's describe the idea, and suggestions from the community on the various Pitch Points are then accepted of rejected by the initiator. Accepted suggestions then update the Pitch Point, while rejected Pitch Points just serve as a record of the discussion. Beyond this negotiation of content, collaboration on PitchHub also features negotiation of visibility of content. Figure 3.2 displays an example of a PitchCard view from the initiator's view.



Figure 3.2: Callaghan Innovation's design for a Pitch Card's visibility scope as seen from a Pitch Card initiator's view.

## 3.2 Methodology

The software methodology adopted in this project has been an agile, iterative approach. Each iteration is approximately one week in length and consists of the following actions: requirements analysis, requirements validation, design, development, testing, and documentation. During the early stages of the project identifying all of the requirements in a waterfall-like approach was infeasible as this would have required large contiguous amounts of time, of which Callaghan Innovation would not have been able to provide. So in keeping with the agile approach requirements were gathered progressively during client meetings and through the already completed conceptual work.

## 3.3 Functional Requirements

- The prototype must fulfil the user flows specified by Callaghan Innovation (i.e. the actions that enable the innovation community to connect and co-create).

- The prototype must be store data securely to establish trust with users, by knowing that their commercially sensitive data is safe.

- The prototype must provide privacy controls over user content so that users can control the scope of disclosure.

## 3.4   Non-functional Requirements

- The prototype must be performant, enabling users to fluently use the platform without distraction.

- The prototype must support a distributed architecture to enable PitchHub the ability to scale and store data in a geographically redundant configuration.

# Chapter 4

# Web Application Design

The design of the web application focused on the use of standard architecture patterns and web development technologies. This approach was taken as web application development covers numerous domains and technologies. In Shklar and Rosen's tome "Web Application Architecture: Principles, Protocols and Practices" they describe the core areas of knowledge required: HTTP, HTML, SMTP, JavaScript, Databases, Graphics Design, and Server Technology [30]. While designing a unique foundation specifically optimised for PitchHub is enticing, as Donald Knuth is famously quoted "97% of the time premature optimization is the root of all evil" [31]. Furthermore, given the time constraints of this project, PitchHub leverages battle tested open source libraries to deliver more functionality while reserving the ability to make custom modifications and changes as necessary. This chapter explores these design choices and also the alternative designs that were considered throughout this project.

## 4.1 Architecture

The architecture of the web application was the first work accomplished on PitchHub. The architecture is the abstraction of the system and hence should be designed with the desired system qualities needed to to be achieved. With PitchHub the security and distributed requirements identified in Chapter 3 were captured in the three tier architecture designed, hence formalising the requirements specification early on.

The common software engineering practice of separation of concerns has a strong presence within this architecture as seen in Figure 4.1. Each tier's logic and responsibilities is encapsulated from the other tiers. Their only knowledge of the outside world is in the design-time defined interfaces of their neighbouring tiers. To partially fulfil the security requirement PitchHub communicates using the *Transport Layer Security* (TLS) protocol, that provides secure bidirectional communication between the client and server.

The data layer is designed in a distributed configuration where database nodes may be scaled horizontally. The data intensive nature of PitchHub means that it can improve it's performance through spreading the data and processing logic among nodes. This kind of approach is known as a shared nothing architecture [32], prevalent in NoSQL systems. The $n$ number of databases shown in Figure 4.1 act as the secret keepers for the database security security scheme discussed in Section 2.6, this is further elaborated in Chapter 6.

In Figure 4.1 within the Client and Server layers the Model-View-Controller architecture pattern has been employed continuing the design theme of separation of concerns. To unpack this term: models maintain state, usually communicating with a database, controllers coordinate interaction and are responsible for delegating tasks to models and views deter-

mine what data is rendered from the model. This separation of responsibilities enables complex sets of interactions to be standardised, conforming to conventions that are well defined and that can be easily be understood [33].
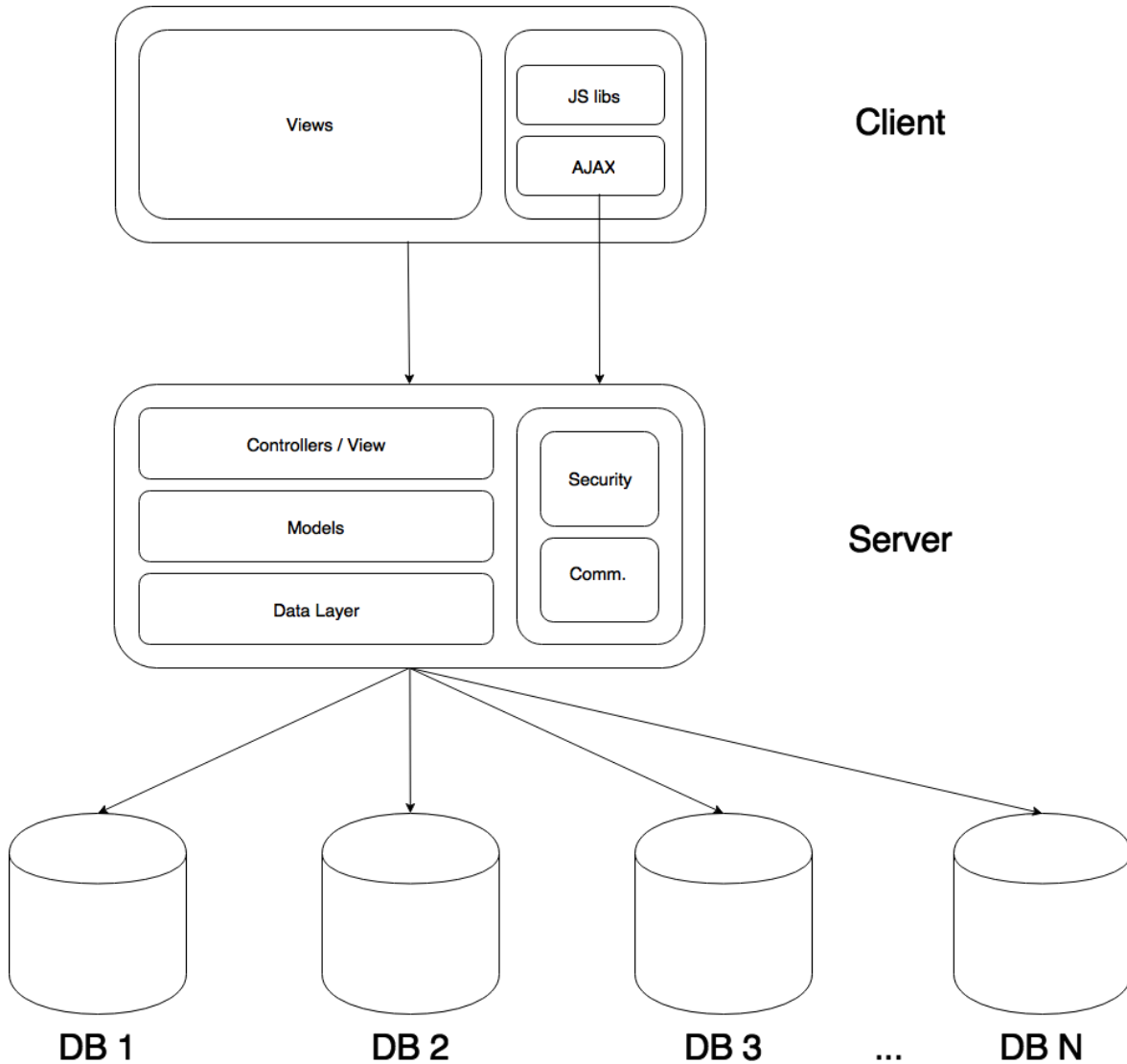


Figure 4.1: The 3-tier architecture used in PitchHub capturing the security and distributed requirements identified in Chapter 3 in the design.

## 4.2 System Model

The design of the system's model is another case of requirements being captured early in the design phase. Figure 4.2 illustrates the system's classes and their relationships as a class diagram. The requirement to implement privacy controls is captured in the Pitch Card and Comment classes' relationship with the DisclosureScope class. To illustrate this it is important to understand how the platform works: A user creates a Pitch Card, detailing the idea's attributes in the Pitch Card's Pitch Points, their aim is to get the community to collaborate on the Pitch Card to derive meaningful information or connections to action the Pitch Card,

collaborator's on a Pitch Card can make suggestions or comments on these Pitch Points to help drive the idea forward. By providing scoping on Pitch Cards initiator's set a base restriction level for the Pitch Card and it's related content. The negotiation aspect of PitchHub is introduced with the Comments and Suggestions users may contribute to PitchCards. As seen in Figure 4.2 these classes also have scoping, however in this case scoping can only be set to an equivalent or more restrictive level than that specified in the Pitch Card. An example of this is where an initiator set's the content scope to 'Members', so only members of the PitchHub can view the idea, now if a user were to contribute a suggestion on a Pitch Point this suggestion can only be scoped as 'Members' or any level which is more restrictive, they cannot however set it to 'Public'. This interaction/requirement as seen in Figure 4.2 is designed with the Strategy Pattern so that future scopes will have minimal change to the application.

## 4.3 Technology Choice

The quality of service attributes of a web application are deeply influenced by the fundamental technologies backing the solution. In this section both the framework and database selections are discussed in relation to quality of service attributes and more importantly the requirements identified in Chapter 3.

### 4.3.1 Framework Selection

Research was conducted on the web application frameworks available in effort to speed up the prototyping process. Ruby on Rails, Laravel, Django, MEAN and OpenSocial were identified as frameworks that could work in fulfilment of the requirements specified. Ultimately the choice of frameworks was between Ruby on Rails and OpenSocial as they are written in languages that I most understand (Ruby and Java, respectively). Ruby on Rails is an open source framework that embraces RESTful web service design and conforms to the MVC architecture. Of note, Ruby on Rails has a wealth of open-source secret sharing and functional testing libraries that are directly applicable to the requirements of this project. OpenSocial in contrast to Ruby on Rails is first and foremost a framework for creating social network, and while PitchHub is not specifically a social network it's primary objective is to facilitate social interaction. Using OpenSocial would offer user authentication as well as messaging and posting functionality out of the box.

Of these frameworks Ruby on Rails was selected because of its vast open source library and elegant handling of complex user interaction flows. This decision results in a trade off in performance. Even in the current versions of each language Java has a significant performance advantage over Ruby [34]. For a simple web application this generally would not be a concern, however the secret sharing component entails the use of encryption algorithms which are computationally expensive. This was concluded not to be a major issue as Ruby on Rails offers the ability to run JRuby which is Ruby executed atop the JVM. JRuby offers significantly improved performance and even allows native Java to be executed if necessary [29].

### 4.3.2 Database Selection

The choice of database has profound effects on the performance and scalability requirements identified in Chapter 3. The rise of NoSQL databases has been attributed to the increasing need for highly scalable and performant databases. Given this need in PitchHub, in addition to PostgreSQL, the NoSQL databases MongoDB and Cassandra were also investigated.

The nature of the Pitch Card data PitchHub is modelling is inherently hierarchical and heterogeneous. In PitchHub, each Pitch Card has a varying number of Pitch Points, and each Pitch Point value also contains a number of interaction attributes. This data model naturally lends itself to the document model offered by MongoDB. Pitch Cards may be modelled in a single document with Pitch Point relations expressed via embedding. This has the additional benefit of being able to efficiently query this Pitch Cards.
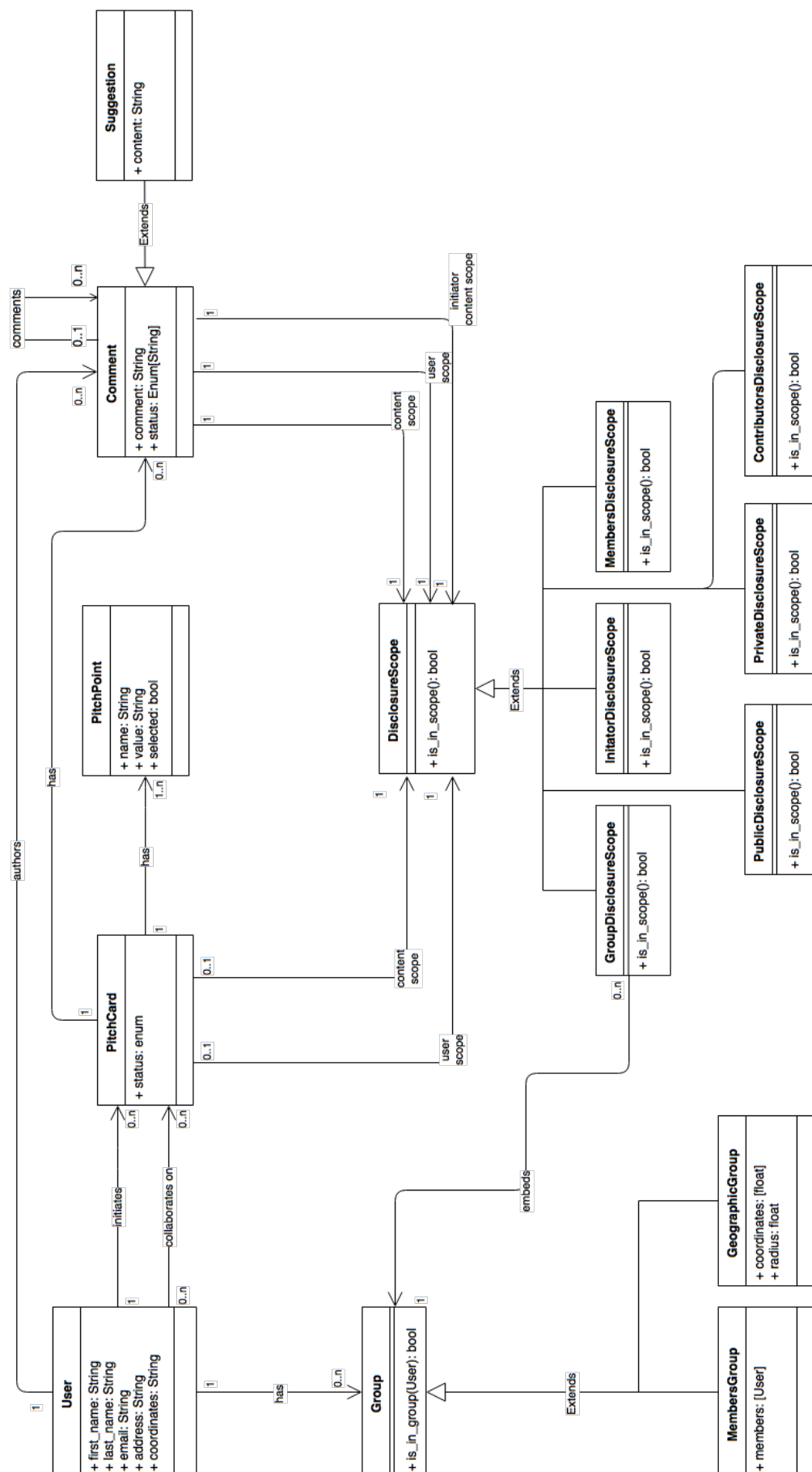
The case for using a relational database like PostgreSQL is also motivated by the inherent nature of PitchHub's data. For example, each user is associated to the Pitch Cards they have initiated and contributed to, as well as the suggestions and comments they have offered on Pitch Cards. Unlike the internal model of Pitch Cards these relations are not well suited to being hierarchical as these relations have associations which are N:M rather than 1:1 or 1:N. Modelling these objects separately in tables and querying them through joins in the relational model is the ideal way to represent and query these relations.

Ultimately MongoDB was selected as the database for PitchHub. MongoDB was chosen because the Pitch Card data is well suited to this data and the use case flows which require joins at most only need one join operation. It was concluded that using MongoDB and performing manual joins within the application is not a major issue because of this. Also MongoDB's ability to scale horizontally easily without the expensive migrations characteristic of relational databases provides an edge over PostgreSQL in meeting the scalability requirement.

## 4.4   Behaviour Driven Development

The development of the web application in keeping with the iterative approach involved progress on many aspects of the project each week: requirements analysis, requirements validation, design, development, testing, and documentation. Behaviour Driven Development (BDD) was incorporated within this process to ensure that the mutual understanding documented in the weekly reports was being directly (and accurately) translated into the prototype.

This kind of development involved requirements being distilled into specifications, with the behaviour of the code being specified ahead of the implementation. This meant that the implementation then had the onus to fulfil the specified behaviour. With this approach PitchHub's progress was able to be tracked in each iteration in regards to actual business value being delivered. A by-product of this approach has resulted in PitchHub's test-suite relying less on tests on classes and units but on specification. The BDD process has been described as producing what is essentially executable documentation [35].

Figure 4.2: PitchHub's system structure as represented in a class diagram. Of note is the Pitch Card and Comment classes and their relationship to the DisclosureScopes. This relationship describes the Pitch Card and Comment classes ability to scope the visibility of their content. (NB: Some attributes were left out for the sake of brevity e.g. Pitch Cards have an 'images' attribute)

# Chapter 5

# Web Application Implementation

## 5.1 Implementation Details

The standard web application functionality required within the prototype, such as authentication, request life-cycles and password resets, was straightforward as Ruby on Rails solves many of these problems, and offers a wealth of libraries that can assist. The Pitch Card functionality and scope of disclosure functionality described in the Sections 3 and 4 were implemented from the ground up. In this Section these two functionality item's implementation details are explored.

### 5.1.1 Pitch Card System

As exemplified in Section 3 Callaghan Innovation's specification for how the Pitch Card system works was quite mature and detailed. At it's core it required that users be able to initiate Pitch Cards and browse Pitch Cards, with the further ability to contribute suggestions or comments. To do this the web application separates the action's responsibilities. As discussed in Section 4.3.1 Ruby on Rails is architected on the MVC architecture pattern. Following Ruby on Rails convention the PitchHub prototype has models, views and controllers for each resource. The controllers adhere to the RESTful design principles, where resources are accessed using conventional HTTP resource methods and relationships are expressed via resource-nesting. The models, as designed in Figure 4.2, were implemented with the Mongoid [36] Object Document Mapper (ODM) for MongoDB. The Mongoid ODM subscribes to the "convention over configuration" philosophy that is held highly in the Ruby on Rails framework, offering a simple faade over the MongoDB query language. The views were implemented with HTML, SASS (CSS), and JavaScript. To enhance the user experience AJAX was implemented to speed up page load times by loading secondary or non-essential data asynchronously. AJAX was also heavily used in user interactions, such as contributing a suggestion/comment and setting disclosure scopes (as an initiator).

Figure 5.2 showcases the prototype's Pitch Card view from the initiator's perspective. The view can be deconstructed as follows: the sidebar contains the links to the main pages, the navigation bar contains the search box and user management drop-down, the main page space contains the Pitch Card and it's associated suggestions. Figure 5.3 contains the same sidebar and navigation bar however the main page space contains a grid of mini-pitch card views consisting of the Pitch Card's image (if any) as well as the *Value Proposition* pitch point.

### 5.1.2 Scope of Disclosure

As previously discussed in Section 4.2 and shown in Figure 4.2 the scope of disclosure functionality is implemented on both Pitch Card and Suggestion/Comment models. For Pitch Cards this scope of disclosure is achieved through a combined effort at the database level and at the application level. To illustrate this consider the 5.3. When the user loads up the dashboard the database is queried asking for Pitch Cards that the current user is able to see (if any). How this first step works is through the use of MongoDB's aggregation pipeline [37]. The aggregation pipeline in this instance deals with the fact that users assume different roles in the context of different Pitch Cards, and Pitch Cards themselves are heterogeneous in the scopes they are defined with. Unlike role-based Access Control Lists, where a user is checked if their role satisfies a particular permission level, the pipeline designed for Pitch-Hub checks the user against each role of the Pitch Card and then against the Pitch Card's visibility scope. This single pipeline aggregation query basically checks a matrix of constraints to check each context. This is visualised in Figure 5.1.
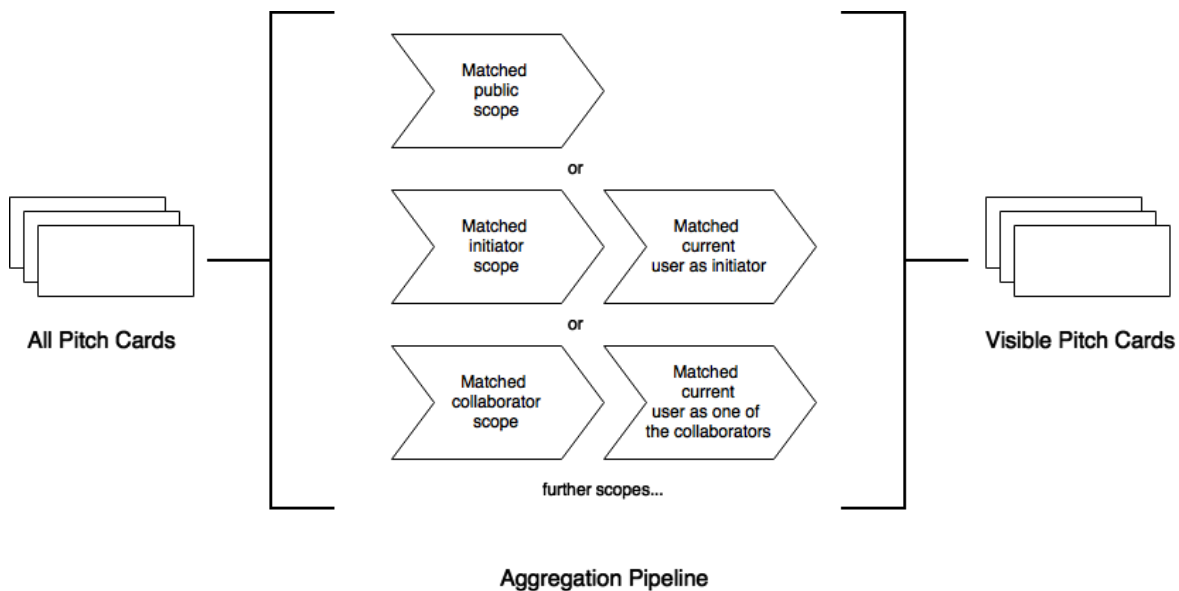


Figure 5.1: The pipeline aggregation query used to find all visible Pitch Cards checks for the scoping of the Pitch Card from the context of the current user.

Continuing the example above, once the viewable Pitch Cards for the Dashboard view have been retrieved from the database the application applies the identity scopes for each Pitch Card in the view. This piece of functionality decides whether the the initiator will appear by their name or by 'anonymous'. The way this is implemented is made simple through the ODM, where when the Pitch Card object is retrieved, the Pitch Card's nested scope objects are also retrieved. As seen in Figure 4.2 these scope objects implement the strategy pattern, so when they are retrieved the application is able to easily check whether the current user is authorised to see the Pitch Card's author agnostic of the actual scope being used. The use of the strategy pattern removes the need to do the less-pragmatic and less-extensible type-checking on scope objects to determine what scoping business logic should be used.

The marriage of database-level and application-level scoping caters to the context of the request's life-cycle stage to maximise efficiency. Certainly, it would have been possible to use the strategy pattern object scoping instead of what is essentially type-checking in the aggregation pipeline query to achieve this scoping. However, at this point in the request the application does not have Pitch Card objects (and their nested scope objects) to scope by,

therefore by expressing this logic in the database query the application is able to appropriately apply scope given the context, retrieving only the viewable Pitch Cards.

## 5.2 Implementation Challenges

### 5.2.1 Virtualised Environment

The time-constrained weekly/bi-weekly meetings with Callaghan Innovation elicited the need for their own locally hosted PitchHub instances. Their own PitchHub instances enabled them to check the progress of the prototype, answer UI/UX questions, and show the prototype to other Callaghan Innovation personnel.

Personally configuring the environment and setting up PitchHub was an option, however this would have still been a lengthy process detracting from actual meeting. Furthermore it would have been infeasible to repeat this for each stakeholder and their various machines. Requiring the stakeholders to do this themselves would have similarly been infeasible as configuring a locally hosted Ruby on Rails application is a non-trivial task [38], to exemplify this StackOverflow has excess of 50,000 questions in relation to Ruby on Rails installation/configuration [39][40][41][42].

To solve this problem the PitchHub environment and installation process was automated using a combination of Chef and Vagrant. Specifically, the Vagrant configuration sets up an Ubuntu virtual machine with routing configured for local access and Chef downloads and installs PitchHub's dependencies (Ruby, a JS runtime, and MongoDB). By having this infrastructure/configuration implemented via code it also serves as documentation for the PitchHub environment and also enables the use of version control within this aspect of the project as well. This process has the further advantage in that any future contributors to the project will have a very low barrier to entry.

### 5.2.2 Deployment

As discussed in Section 1.2 one of the objectives was to have a prototype deployed by August so that Callaghan Innovation may conduct an internal user study. Deploying PitchHub onto the world wide web turned out to be a fairly time-consuming challenge. Callaghan Innovation kindly provided the hardware, racked the machines in their server room, and also configured the firewall. Prior to this experience I had only limited experience in deploying web-applications and my knowledge was limited to that of using Heroku, a platform as a service provider [43]. In deploying PitchHub I learnt how to set up an nginx HTTP server [44] with the Phusion Passenger application server [45] and also configure SSL. The primary pain point in deploying PitchHub was with the actual code deployment. In deploying a Ruby on Rails application there are a few common options: SFTP into the server and transfer the files manually or using Capistrano [46] to automate the deployment. These approaches have different strengths and weaknesses. The manual SFTP option has a very low barrier to entry, as it essentially just a method that copies locally selected files/directories files onto the server, on the downside it is also a very brittle process, as it requires one to remember to pre-compile the assets and check that the local copy is the correct version of the application to be deployed. The Capistrano approach is similar to Vagrant in that the entire process is defined in code. Being automated is the greatest advantage to deploying with Capistrano as we can ensure each deployment fulfils each task correctly and in order, and should the process require more tasks these can be easily added. The problem with Capistrano is that it requires a deep knowledge of the framework. Ultimately, I decided to use Capistrano as automation is an attribute that appeals to me, as the initial investment is

negligible in comparison to the long term gains of robust and consistent deployments. Pitch-Hub is currently hosted at *pitchhub.net*, note that the current release is being used internally in Callaghan Innovation and requires an access code to sign up.

Figure 5.2: Fictional Ford Model T Pitch Card view from the initiator's perspective. The view is divided into two sections the Pitch Card and it's suggestions/comments. In the Pitch Card half users perform in-line editing on a Pitch Point to make a suggestion. In the suggestion/comments half the initiator may accept or decline the suggestion and set the suggestion/comment's scope.
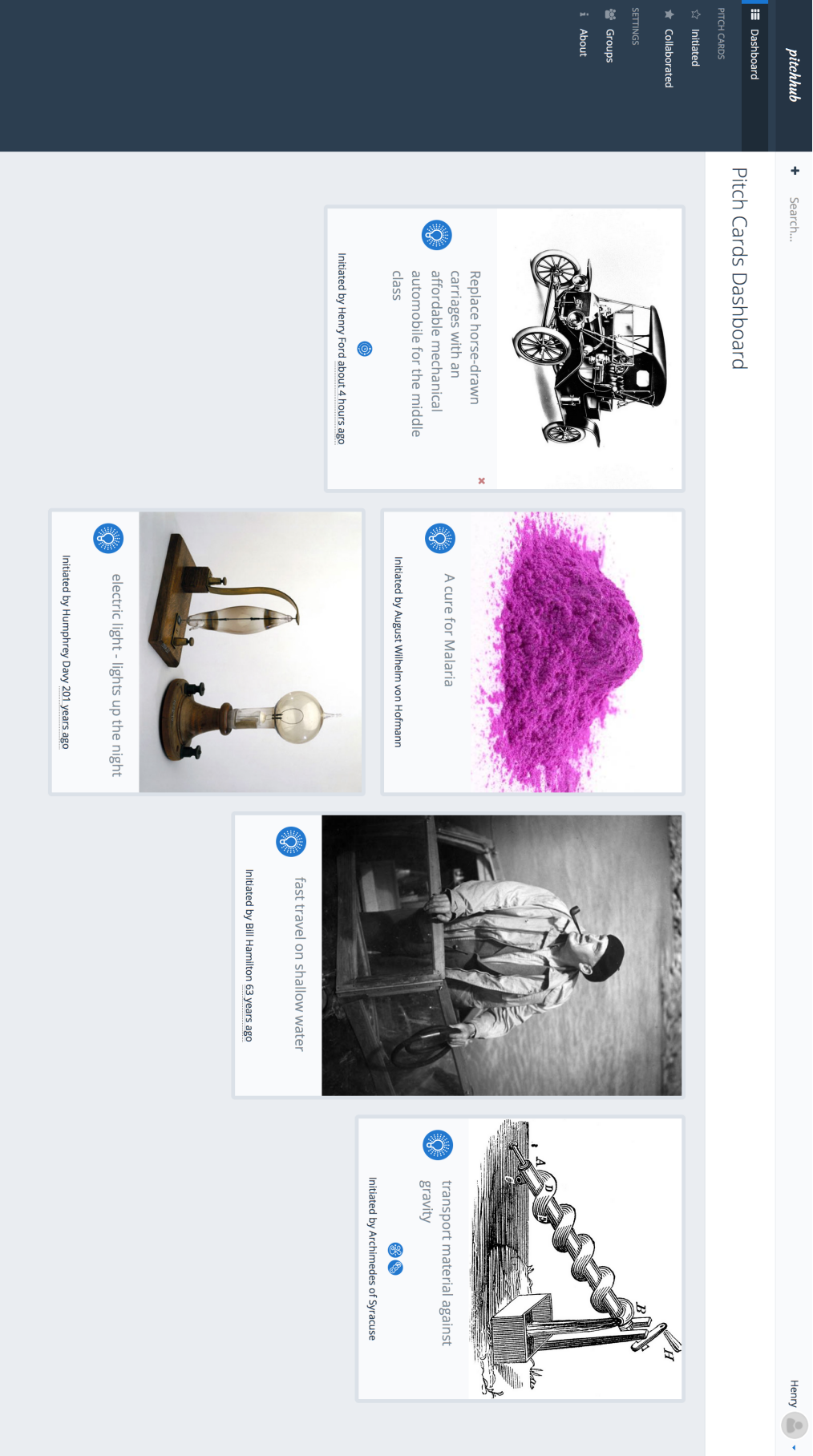
Figure 5.3: PitchHub's dashboard populated with fictional Pitch Cards. The grid layout displayed is responsive, so the Pitch Cards will reorganise and size to fit the user's device screen.

# Chapter 6

# Threshold Security Scheme

## 6.1 Design

As discussed in Section 2.6 Shamir's Secret Sharing Scheme is a threshold security scheme based on Lagrange interpolation. Recall, the scheme works by splitting the secret into $n$ shares, $k$ of which are required to then retrieve the original secret. In this section the design for the service enabling PitchHub to leverage the secret sharing scheme is discussed.

### 6.1.1 Shamir's Secret Sharing Scheme Service

First, in designing the service an open-source implementation of Shamir's Secret Sharing was selected. This was a key design decision. While the principles of Shamir's Secret Sharing are simple, personally implementing a cryptography library should be treated with care. To implement and establish trust in such a component would require rigorous testing to be proven as secure. Leveraging the collective strength of the open-source community ensures that the implementations have had many eyes go through them (with different backgrounds/expertises). The primary activity for PitchHub in regards to Secret Sharing was to design and implement a service that integrates the security scheme.

As discussed by prominent cryptographer and author Bruce Schneier "Building cryptography into products is hard... Flaws can appear anywhere. They can be in the trust model, the system design, the algorithms and protocols, the implementations, the source code, the human-computer interface, the procedures, the underlying computer system. Anywhere." [47]. PitchHub approached the integration of the security scheme library with emphasis on limiting the amount of coupling. By reducing coupling between the library and wider system we reduce the risk of system knowledge producing vulnerabilities within PitchHub and undermining the security scheme's integrity.

In designing this service the MVC architecture pattern was analysed in regards to which component is most suitable to handle this responsibility. The model could use a Ruby mixin to override both it's *save* and *find* methods such that the secret was split into $n$ shares on *save* for the $n$ databases. *Find* would work similarly, merely combining the queried shares from the $k..n$ databases and presenting the clear text secret. The view component is not supposed to deal with business logic and hence not an appropriate candidate for this responsibility. The controller in it's capacity of delegating tasks to models could also bear this responsibility. In the controller on *saves* it could split the secret among $n$ models and save each model to a distinct database. With *finds* the controller would need to connect to each database and retrieve the requested secrets to combine. The pros and cons of each design is apparent. Going strictly the model route would mean *monkey patching* [48] the behaviour of the ODM which could make it hard for other components that do not wish to use the Secret Sharing

implementation also this would require that models have knowledge outside of itself. Going strictly controller means that there is a fair amount of business logic within the controllers. However, both approaches consolidate the integration of the Secret Sharing library within it's own component. Ultimately, the final approach was a mixture of the above, controllers were modified such that they use explicit Secret Sharing *find* and *save* methods which encapsulate the knowledge of the *n* secret keeper databases, within these explicit Secret Sharing database methods the logic of the Secret Sharing library is isolated.

Despite both Models and Controllers being aware of the Secret Sharing functionality, they are only aware of as much to the capacity of their roles. The model handles the business logic, while the controller handles the delegation.

### 6.1.2 Overcoming Limitations of Threshold Security Schemes

Given share combinations up to $k$, Secret Sharing Schemes ensure that the secret is not revealed. However, if an attacker gains $k$ secret shares, the secret is easily reconstructed. This is simultaneously the advantage and weakness of Threshold Security Schemes. To explore further: up to $k$ compromised databases do not compromise the security of the secrets being held, but is reasonable to suggest that if an attacker can break into one database they are capable of breaking into all of them. Furthermore this scheme relies on the availability of $k$ secret keepers, should $k$ be unattainable the secrets are rendered irretrievable to both authorised and unauthorised users alike.

In the context of PitchHub this scenario requires an attacker to have infiltrated $k$ databases.

## 6.2 Implementation

### 6.2.1 Shamir's Secret Sharing Scheme Service

### 6.2.2 Secret Keeper Redundancy

# Chapter 7

# Experimental Methodology

## 7.1 Functional Testing Method

### 7.1.1 Testing Environment

talk about reproducible environment

### 7.1.2 Test Data

frequency analysis of data cleaned and given by CI's user trial
    seeded given frequency analysis results

### 7.1.3 Automated Testing

talk about selenium and user stories

### 7.1.4 Performance Considerations

talk about NN threshold

## 7.2 Security Testing Method

### 7.2.1 Security Testing Scope

Our threat model consists of resisting at least one shoulder surfing attack from an observer co-located at any position around the tabletop. Camera-based attacks are feasible with most knowledge-based authentication systems; but to defeat camera attacks was not our design goal. The pervasive na- ture of mobile devices instrumented with cameras is of par- ticular concern, but as with other manifestations of this same problem (e.g. at the ATM) we rely upon social conventions to deter active attempts to video record logins.

### 7.2.2 Threat Taxonomy

# Chapter 8

# Evaluation

## 8.1 Functionality

### 8.1.1 Comparison of Prototypes

## 8.2 Security

### 8.2.1 Threat Taxonomy

# Chapter 9

# Summary and Conclusions

## 9.1  A Summary of The Developed Prototypes

## 9.2  A Discussion of Online Innovation Collaboration and The Prototypes

## 9.3  Future Work

### 9.3.1  Recommendation Engine

### 9.3.2  Usability Evaluation/Improvement

## 9.4  Final Comments

# Bibliography

[1] E. A. Von Hippel, "Democratizing innovation," 2005.

[2] Y.-K. Che and I. Gale, "Optimal design of research contests," *The American Economic Review*, vol. 93, no. 3, pp. 646–671, 2003.

[3] J. F. Engelberger, "Robotics in practice: Future capabilities," *Electronic Servicing & Technology magazine*, 1982.

[4] J. Livingston, *Founders at work: stories of startups' early days*. Apress, 2007.

[5] J. Hautz, K. Hutter, J. Füller, K. Matzler, and M. Rieger, "How to establish an online innovation community? the role of users and their innovative content," in *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*. IEEE, 2010, pp. 1–11.

[6] J. Boyd, "In community we trust: Online security communication at ebay," *Journal of Computer-Mediated Communication*, vol. 7, no. 3, pp. 0–0, 2002.

[7] M. Johnson, S. Egelman, and S. M. Bellovin, "Facebook and privacy: it's complicated," in *Proceedings of the eighth symposium on usable privacy and security*. ACM, 2012, p. 9.

[8] D.-H. Shin, "The effects of trust, security and privacy in social networking: A security-based approach to understand the pattern of adoption," *Interacting with Computers*, vol. 22, no. 5, pp. 428–438, 2010.

[9] "ideaforge — the new engine of creation," http://ideaforge.io/, (Visited on 08/24/2015).

[10] "Assembly," https://assembly.com/, (Visited on 08/24/2015).

[11] "Angel list," https://angel.co, (Visited on 04/07/2015).

[12] "Enterprise angels —startup capital—entreprenuers—angel fund," http://www.enterpriseangels.co.nz/, (Visited on 08/26/2015).

[13] "Kickstarter," https://www.kickstarter.com/, (Visited on 08/26/2015).

[14] "Indiegogo: The largest global crowdfunding & fundraising site online," https://www.indiegogo.com/, (Visited on 08/26/2015).

[15] "Pledgeme," https://www.pledgeme.co.nz, (Visited on 04/07/2015).

[16] "Linkedin," https://www.linkedin.com, (Visited on 04/07/2015).

[17] "New zealand healthcare innovation — linkedin," https://www.linkedin.com/groups/New-Zealand-Healthcare-Innovation-2035021/about?report%2Esuccess=8QK6tymVv4FNpt6BxhCPZkDgzjfC-GH21u7zSpGeVjkNQ3hCQIv-c5Ggp8y0Zj2hqdM4ZJcgV2MPLXh (Visited on 08/26/2015).

[18] "Facebook - log in or sign up," https://www.facebook.com/, (Visited on 08/26/2015).

[19] "The great new zealand science project — facebook," https://www.facebook.com/nzscience?ref=ts&fref=ts, (Visited on 08/26/2015).

[20] Google, "Google groups," https://groups.google.com/forum/#!overview, (Visited on 04/07/2015).

[21] "Nz nsc 10 science for technological innovation - google groups," https://groups.google.com/forum/#!forum/nsc10, (Visited on 08/26/2015).

[22] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[23] A. Abdallah and M. Salleh, "Analysis and comparison the security and performance of secret sharing schemes," *Asian Journal of Information Technology*, vol. 14, no. 2, pp. 74–83, 2015.

[24] J. L. Dautrich and C. V. Ravishankar, "Security limitations of using secret sharing for data outsourcing," in *Data and Applications Security and Privacy XXVI*. Springer, 2012, pp. 145–160.

[25] V. Shoup, "Practical threshold signatures," in *Advances in CryptologyEUROCRYPT 2000*. Springer, 2000, pp. 207–220.

[26] M. Abdalla, S. Miner, and C. Namprempre, "Forward-secure threshold signature schemes," in *Topics in CryptologyCT-RSA 2001*. Springer, 2001, pp. 441–456.

[27] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *Advances in CryptologyCRYPT095*. Springer, 1995, pp. 339–352.

[28] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl, "Asynchronous verifiable secret sharing and proactive cryptosystems," in *Proceedings of the 9th ACM conference on Computer and communications security*. ACM, 2002, pp. 88–97.

[29] JRuby, "Calling java from jruby," https://github.com/jruby/jruby/wiki/CallingJavaFromJRubyl, (Visited on 04/07/2015).

[30] L. Shklar and R. Rosen, "Web application architecture," *Principles, Protocols and Practices, Editura Wiley*, 2009.

[31] D. E. Knuth, "Structured programming with go to statements," *ACM Computing Surveys (CSUR)*, vol. 6, no. 4, pp. 261–301, 1974.

[32] M. Stonebraker, "The case for shared nothing," *IEEE Database Eng. Bull.*, vol. 9, no. 1, pp. 4–9, 1986.

[33] A. Leff and J. T. Rayfield, "Web-application development using the model/view/controller design pattern," in *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*. IEEE, 2001, pp. 118–127.

[34] C. M. Mateo, "Performance of several languages," http://blog.carlesmateo.com/2014/10/13/performance-of-several-languages/, (Visited on 04/07/2015).

[35] D. Astels, "A new look at test-driven development," *http://blog. daveastels. com/files/BDD_Intro. pdf¿ Acessado em*, vol. 12, p. 2013, 2006.

[36] "Mongoid tutorial (5.0.0)," http://docs.mongodb.org/ecosystem/tutorial/ ruby-mongoid-tutorial/#ruby-mongoid-tutorial, (Visited on 09/02/2015).

[37] "Aggregation pipeline mongodb manual 3.0," http://docs.mongodb.org/manual/ core/aggregation-pipeline/, (Visited on 09/02/2015).

[38] D. R. Klapaukh, "Trials and tribulations of running ruby on rails," Personal Conversations, discussed on multiple occasions, mostly during the Victoria University Open Day.

[39] "Posts containing 'rails install' - stack overflow," http://stackoverflow.com/search?q= rails+install&s=c502979e-0897-4245-a1f8-e11be07f134d, (Visited on 09/03/2015).

[40] "Posts containing 'rails configuration' - stack overflow," http://stackoverflow.com/ search?q=rails+configuration, (Visited on 09/03/2015).

[41] "Posts containing 'rails gem' - stack overflow," http://stackoverflow.com/search?q= rails+gem, (Visited on 09/03/2015).

[42] "Posts containing 'rails environment' - stack overflow," http://stackoverflow.com/ search?q=rails+environment, (Visited on 09/03/2015).

[43] "Heroku — cloud application platform," https://www.heroku.com/, (Visited on 09/04/2015).

[44] "nginx news," http://nginx.org/, (Visited on 09/04/2015).

[45] "Fast web server & app server, ruby python node.js - phusion passenger," https:// www.phusionpassenger.com/, (Visited on 09/04/2015).

[46] "A remote server automation and deployment tool written in ruby." http:// capistranorb.com/, (Visited on 09/04/2015).

[47] Bruce Schneier, "Cryptography: The importance of not being different," *Computer*, vol. 32, no. 3, pp. 108–109, 1999.

[48] "Monkey patch - wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/ Monkey_patch, (Visited on 09/05/2015).