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 a critical part of the innovation process. This project is concerned with empowering the innovation community with an online collaboration system that: engages the various roles present in the innovation ecosystem, functions on the notion of scoping/trust and ensures that all sensitive intellectual property shared is stored in a secure manner.

Contents

Figures

Chapter 1

Introduction

The aim of this project is to design, prototype and evaluate PitchHub, an on online collaboration system that empowers the innovation community. This is achieved by engaging the various roles present in the innovation ecosystem, functioning on the notion of scoping/trust and ensuring that all sensitive intellectual property shared is stored in a secure manner. This is an industry project in association with Callaghan Innovation.

1.1 Motivation

The deep integration of digital media within modern society has had a profound effect on how we communicate and collaborate with others. Historically innovation had a great deal to do with proximity. Knowledge of the problem and access to the resources required to solve the problem were the primary limiting factors. In the digital age, the move online has essentially obviated this need for proximity.

As a result the innovation community has been able to tackle problems regardless of geography. Despite this, Callaghan Innovation that the innovation community is to a large extent still fragmented. There are three primary causes for this. First, the current solutions used in the innovation space for collaboration either focus on networking around people rather than the ideas [1][2] or only facilitate collaboration between certain roles in the innovation community (e.g. investors, inventors, entrepreneurs) [3][4][5][6]. Second, contributing intellectual property online requires a significant amount of trust from the users. As once submitted the original poster has limited control over their intellectual property's dissemination. Third, the presence cyber threats is ever-growing [7]. Services facilitating innovation need to have security measures in place that protect stored intellectual property from potential malicious access.

1.2 Proposed Solution

A solution to the issues identified above is to use a system that is specifically designed for engaging the innovation community, that enables contributors the ability to control the scope of their intellectual property's dissemination, and that ensures security of this intellectual property. To engage the innovation community it must support all roles within the innovation ecosystem. In doing so PitchHub will be able to facilitate collaboration for a large percentage of the collaboration community. By providing functionality for explicit scoping of intellectual property users may protect themselves from accidental disclosure. Finally, by storing sensitive data via a Threshold scheme users can be assured of unconditional security with regard to malicious access (of up to k databases, discussed in Chapter 6). The aim of

this project has been to design, implement and evaluate this solution, henceforth referred to as PitchHub.

1.3 Contributions

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.

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" [8]. The world of innovation today now includes actors from increasingly disparate domains who are able to contribute their unique capabilities to the innovation process [9]. The influx of unique skills being mixed into the innovation community has resulted in more unique opportunities for innovation being made possible. To take advantage of these opportunities it is critical that there be a communication layer where these actors may collaborate.

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 [10]. 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 [11]. 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
- 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 innovation, this may be in terms of man-power, assets or financing. Solvers contribute answers 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 visibility. In most cases these roles are too large for one person to embody them all. To continue with the Apple][example, Steve Jobs may be categorised 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 and built the Apple][, and Mike Markulla, can be regarded as an Enabler and Facilitator, as he financed the production and also lent his reputation to the product. It is important to recognise that many innovations, such as the Apple][, require a collaboration between these roles to be successfully executed. Platforms seeking to encourage innovation must therefore provide functionality for the collaboration/networking between actors fulfilling these roles.

2.3 Innovation Online

Since the advent of the world wide web communication and knowledge sharing has never been more accessible or easy. Naturally, this has been a boon for the innovation community. The reach the world wide web affords enables innovation communities to capitalise on sources of innovative potential and knowledge at unprecedented scale [12].

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. 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. Boyd's research into online marketplaces has shown the importance of this trust in a business context: "without trust, risk is paralyzing; transactions simply do not take place" [13]. This notion is similarly applicable in the online innovation space, the only difference being that users are transacting in intellectual property and skills rather than money. Because users have very little power over how their data is stored and disseminated it is paramount for platforms to make good on this trust. This can be achieved by implementing safeguards against security threats and providing functionality that gives users *control* over the visibility/dissemination 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 online privacy [14], social networks also face the problem of managing insider threat. Insider threat is where users unintentionally 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 [15].

2.5 Related Work

This section explores the current solutions being used to facilitate collaboration in the innovation community. This is done through the use of the Innovation Role Taxonomy developed. The Innovation Role Taxonomy consists of the roles identified in Section 2.2 as well as their level of support for collaboration between roles: either implicit or explicit. Implicit support is represented by systems that have not established functionality for collaboration between the roles, and yet is still employed by its users to collaborate in carrying out these roles.

IdeaForge [16] is a collaborative innovation platform that explicitly supports collaboration between 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 [17] 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" [12], 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 [5] and **Enterprise Angels** [18] are examples of online platforms for investors (a subset of of Enablers) looking to fund businesses. Crowd funding and microequity platforms such as **Kickstarter** [19], **Indiegogo** [20] and **PledgeMe** [4] 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** [1] being used by New Zealand Healthcare Innovation [21], **Facebook** [22] being used in the Great New Zealand Science Project [23], and **Google Groups** [2] being used in the National Science Challenges [24].

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 directed and 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. This is most evident through the Innovation Role Taxonomy, Fig 2.1. It is clear that there is a need for PitchHub, or a similar service, where explicit support for collaboration between all roles is provided.

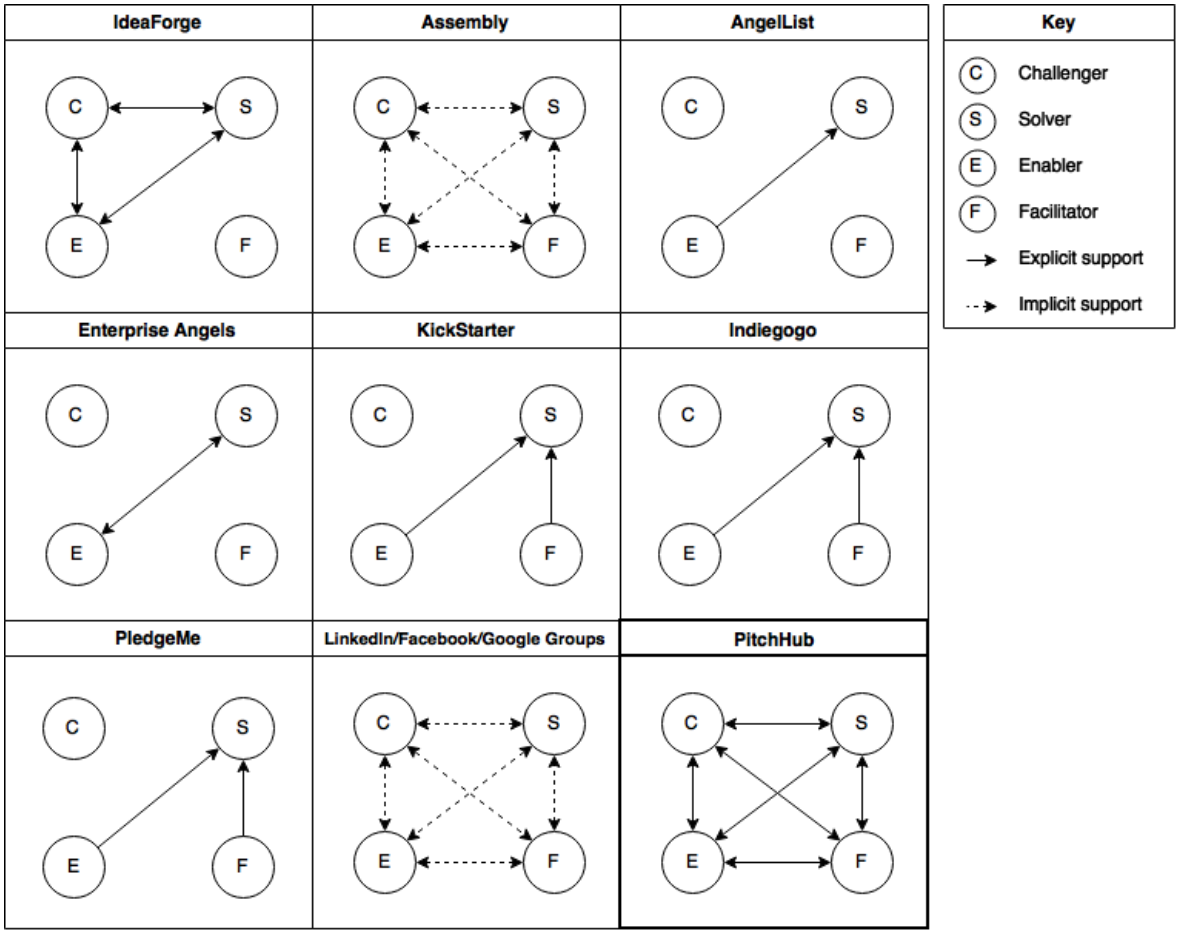

Figure 2.1: All collaborative platforms investigated do not provide explicit collaboration support between the all roles identified in Section 2.2. PitchHub aims to fix this by supporting networking between all roles.

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. From this point onward Callaghan Innovation's initial conceptualisation of PitchHub is referred to as as PitchHub alpha.

3.1.1 Pitch Cards Conceptualisation

As discussed in Chapter 2 the innovation community has a very specific purpose and therefore requires special kind of user participation/interaction [25]. 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 PitchHub alpha's Pitch Card concept.

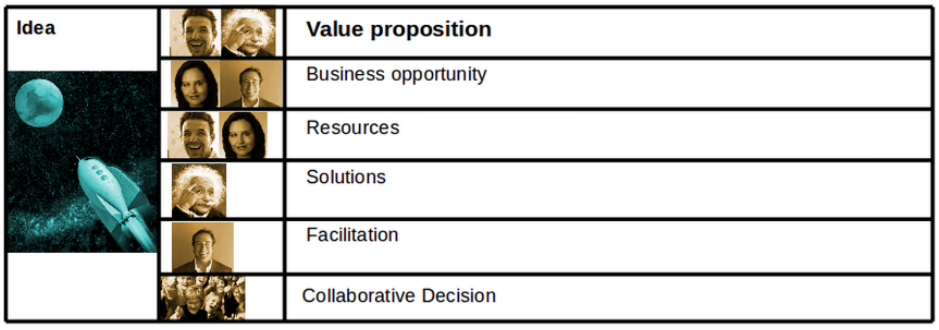

Figure 3.1: PitchHub alpha's design for a Pitch Card describes an idea with the following Pitch Points: Value Proposition (Any role, describing the idea's value), Business Opportunity (Challenger), Resources (Enabler), Solutions (Solver), Facilitation (Facilitator), Collaborative Decision (Any role,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

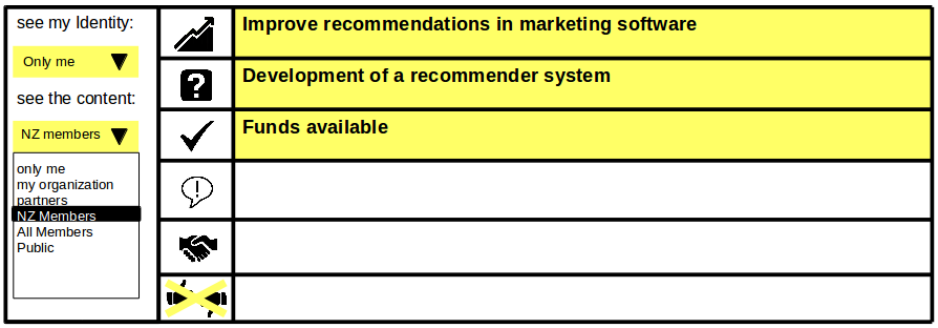

Figure 3.2: PitchHub alpha's design for a Pitch Card's visibility scope as seen from a Pitch Card initiator's view.

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 Design Requirements

The following functional requirements were identified as being key to the success of the project:

D1: The prototype must enable the innovation community to collaborate. The most significant requirement is to ensure that the user stories specified by Callaghan Innovation are fulfilled. The prototype must support these interactions in a sensible manner. Focusing on ideas, rather than people, users should be able to initiate and collaborate on the execution of innovative ideas.

D2.1: The prototype must enable users to scope the disclosure of their content/identity. An interaction behaviour must be developed such that users are able to negotiate the scope of their content to prevent unwanted attention. If users do not wish to disclose their identity

or wish to use a scope this behaviour must be supported. This protects users from accidental disclosure.

D2.2: The prototype must provide auditing functionality. In order for users to trust that the other users who have viewed their IP are not simply copying it, users must have the ability to audit who has seen their contributions. This enables users to track the movement of their IP and also discourage users from acting unfaithfully.

D2.3: The prototype must store sensitive data securely. In order for users to feel safe contributing what may be commercially sensitive communication all data of this nature must be stored in a secure manner. This protects users from malicious disclosure.

D3: The prototype must be portable and extensible. Given the varying degrees of technical skills of the stakeholders at Callaghan Innovation the prototype must be able to be run and extended with minimal effort. The prototype must therefore consider technologies that support ease-of-use over the entire SDLC.

D4: The prototype must be performant. In order to facilitate collaboration within the online innovation community the prototype must enable users to fluently use the platform without distraction. It is well-known that extended load/wait times degrade overall user satisfaction and also increase the likelihood of users abandoning operations [26][27][28].

D5: The prototype must support a distributed architecture. The ability to easily/efficiently scale to meet user demand is a highly desirable property for web systems today. The prototype should use a distributed architecture to achieve this scalability. Consideration must be given as to how redundancy and security operate within the system's distributed context.

Chapter 4

Web Application Design

TODO

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 [29]. 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" [30]. With this in mind, 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 User Stories

The principal function of PitchHub is to facilitate collaboration in the innovation community, this is reflected in requirement D1. As such the design of PitchHub began with specifying these fundamentals in user stories. To do this each user story captured a desired behaviour by identifying the behaviour's corresponding 'who', 'what', and 'why' attributes. An example user story capturing *Post Pitch Card* behaviour is as follows:

```
As a user, I want to create a Pitch Card so that the community and I may
collaborate to find a solution
```

To unpack on this example, the 'who' is a user, the 'what' is the ability post Pitch Cards, and the 'why' is the desire of finding a solution through collaboration. So from these user stories not only is the behaviour captured but also relevant expectations of state. To continue using the above example, it can be discerned that once the user has posted a Pitch Card there is an expectation that other users will be able to see it and also interact with it 'to find a solution'. Specifying these user stories was done in collaboration with Callaghan Innovation to ensure the behaviour documented was accurate.

To have a complete understanding of the functionality required by PitchHub user stories were used to describe all desired behaviour, ranging from sign-up to content scoping (as per requirement D2.1). These user stories as well as their feature definitions (the executable/testable representation of these user stories) are found in Appendix

4.2 UI Design

TODO

4.3 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 [31], 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 6.1, 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 determine 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 [32].

4.4 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,

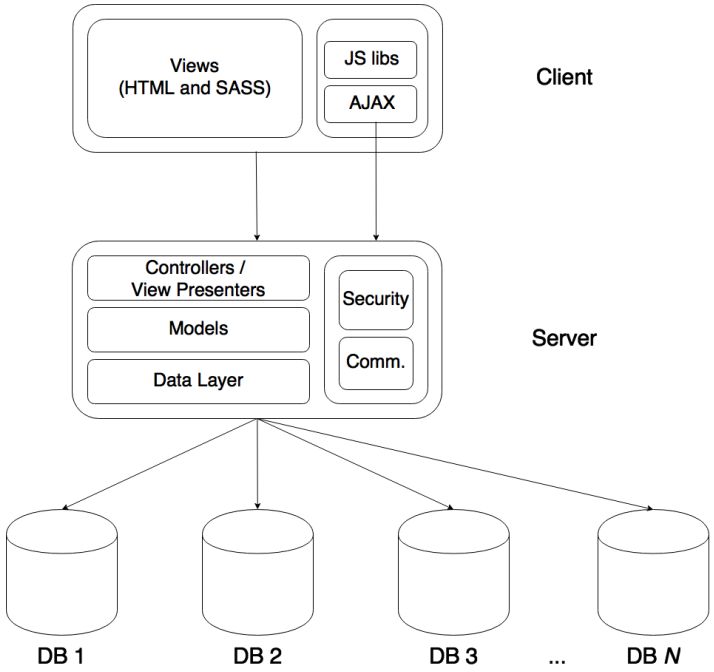

Figure 4.1: The 3-tier architecture used in PitchHub capturing the security and distributed requirements identified in Chapter 3 in the design.

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.

4.5 Technology

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.

4.5.1 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 these were identified as being the most accessible. 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 networks, 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 [33]. 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 [25].

4.5.2 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 (Postgres), 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 Postgres 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 both databases were selected for PitchHub. MongoDB has been configured as the default database in the prototype. While Postgres has been used to diversify the threshold scheme's secret keepers, this is discussed in Section 6.3.2. MongoDB was chosen as the default because the Pitch Card data is well suited to MongoDB's data model 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 Postgres in meeting the scalability requirement.

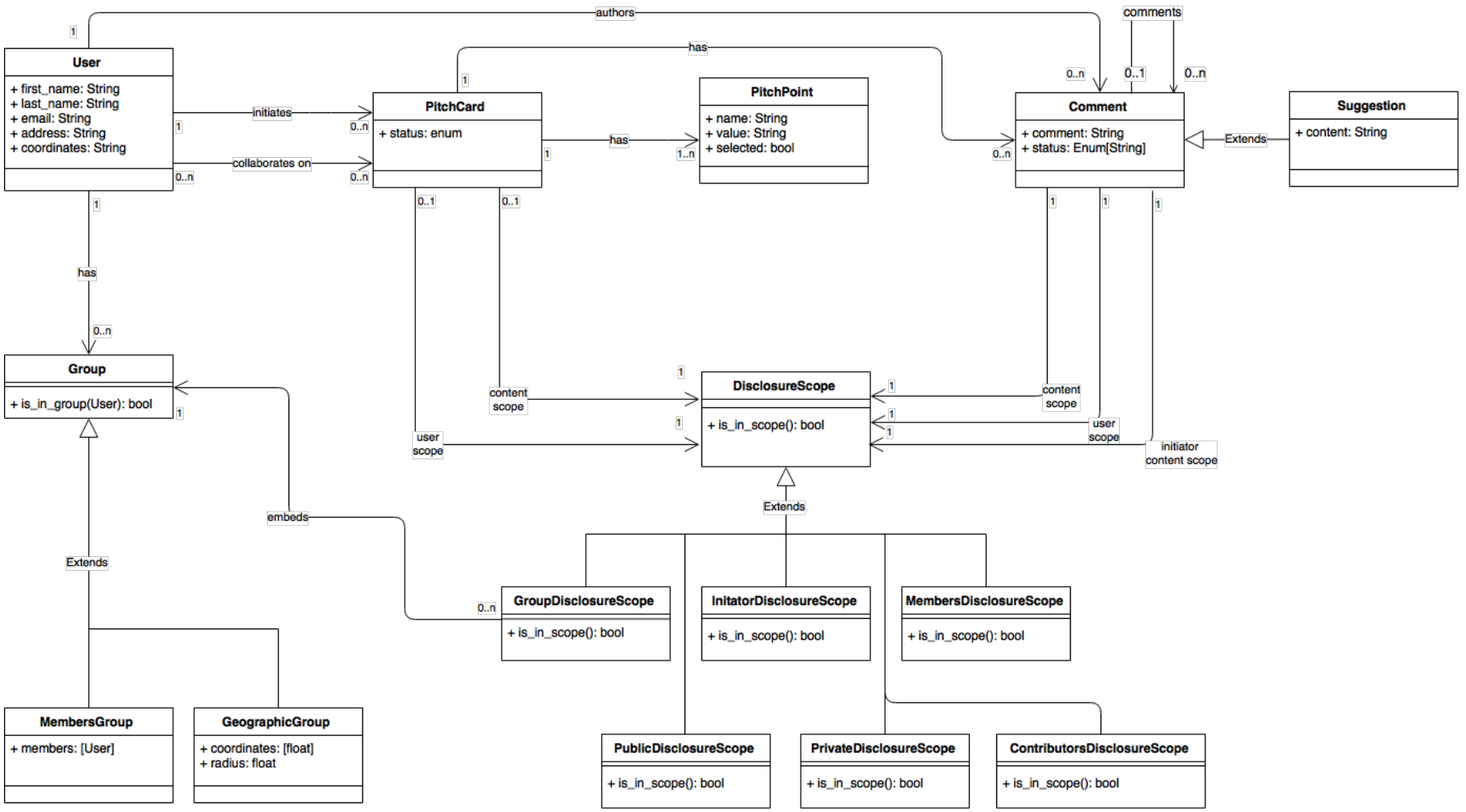

Figure 4.2: PitchHub's system structure as represented in a class diagram. Of note is the Pitch Card and Comment classes and their relationship to the DisclosureScopes. This relationship describes the Pitch Card and Comment classes ability to scope the visibility of their content. (NB: Some attributes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

Chapter 5

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 4.5.1 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 [34] 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 façade 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 4.4 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 [35]. 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 PitchHub 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.

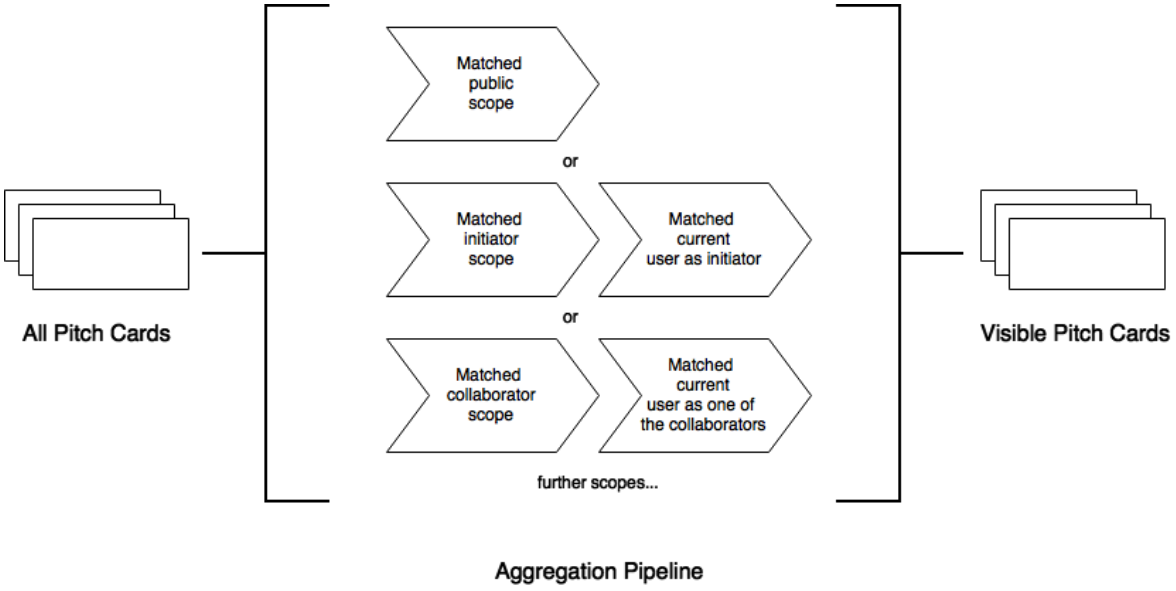

Figure 5.1: The pipeline aggregation query used to find all visible Pitch Cards checks for the scoping of the Pitch Card from the context of the current user.

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 [36], to exemplify this StackOverflow has excess of 50,000 questions in relation to Ruby on Rails installation/configuration [37, 38, 39, 40].

To solve this problem the PitchHub environment and installation process was automated using a combination of Chef and Vagrant scripts. Vagrant was employed to automate the Ubuntu virtual machine setup and Chef was leveraged to manage PitchHub's various environment 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 enables the use of version control within this aspect of the project. This process has the further advantage in that any future contributors to the project will have a very low barrier to entry in terms of setting up their own locally hosted PitchHub instance.

5.2.2 Deployment

As discussed in Section 1.2 one of the objectives was to have a prototype deployed for internal use by Callaghan Innovation. The deployment process of PitchHub consisted of two steps: first, setting up the infrastructure for the deployment and second, deploying the Ruby on Rails prototype onto the infrastructure. To support the first step, Callaghan Innovation provided a machine, racked it in their server room, and also configured the machine's domain name. I then completed the infrastructure setup by configuring the nginx HTTP server [41], the Phusion Passenger application server [42] and SSL. In regards to step 2, there exist a few prominent methods of deploying Ruby on Rails applications:

- A manual process using a combination of SFTP and SSH, to transfer files and update environment variables.
- An automated process facilitated by Capistrano, a Ruby on Rails deployment library.

The manual process, while easier for the initial deployment, was decided against as the process is brittle in that it relies on the deployer to fulfil each deployment step correctly and in order. For a Ruby on Rails application this process at the very minimum consists of: transferring the updated application code, pre-compiling the assets, downloading any newly added libraries and updating the binaries. Should this be done incorrectly or in the wrong environment this manual process has no 'undo' action, therefore the process of reverting the application must also be done manually. Capistrano alleviates these problems but at the cost of a steep-learning curve. Capistrano is similar to Vagrant in that the process

is automated by scripts which automate the entire process. Ultimately, the initial investment in defining the scripts is negligible in comparison to the long term gains of robust and consistent deployments. PitchHub 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.

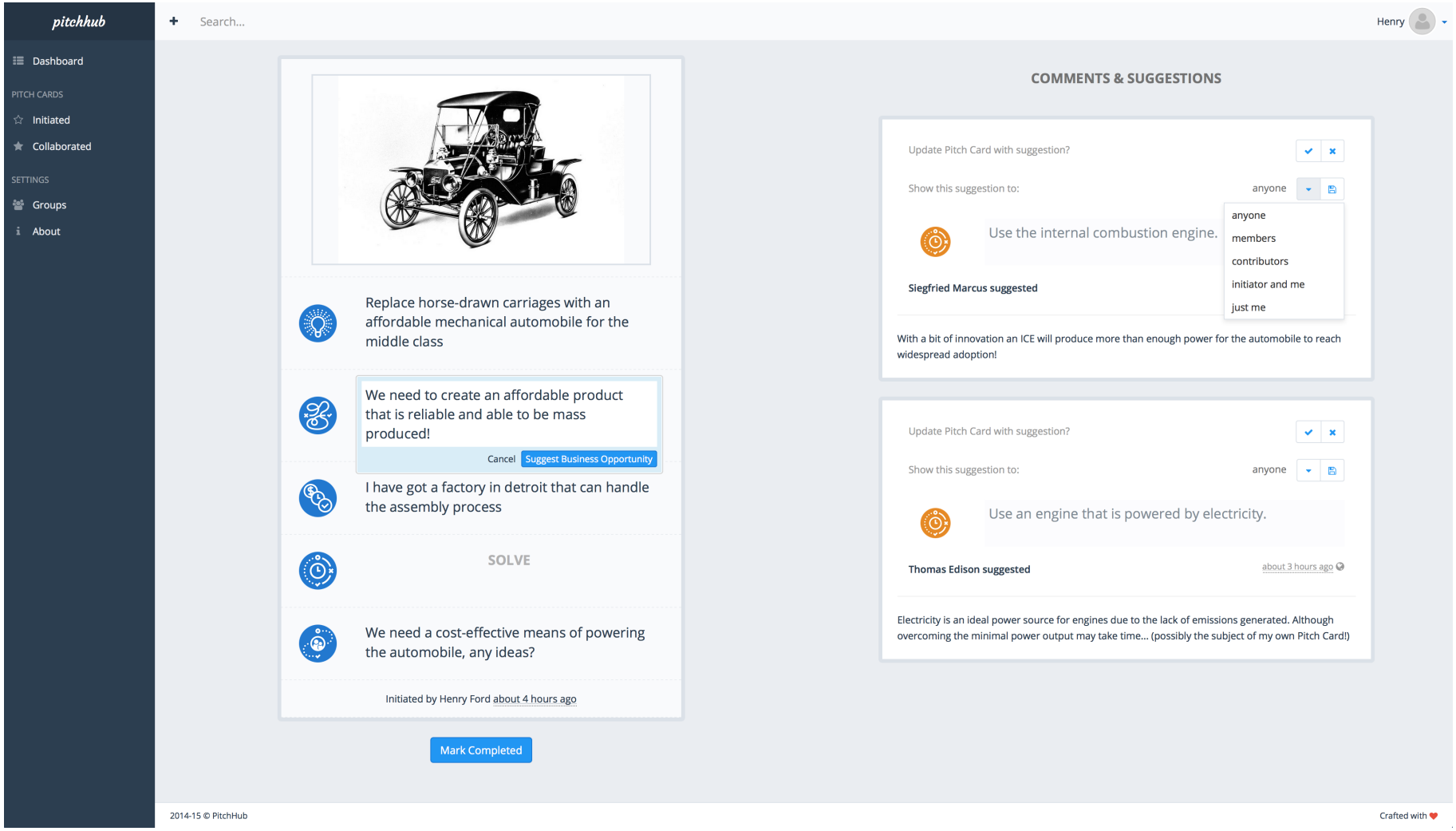

Figure 5.2: Fictional Ford Model T Pitch Card view from the initiator's perspective. The view is divided into two sections the Pitch Card and it's suggestions/comments. In the Pitch Card half users perform in-line editing on a Pitch Point to make a suggestion. In the suggestion/comments half the initiator may accept or decline the suggestion and set the suggestion/comment's scope.

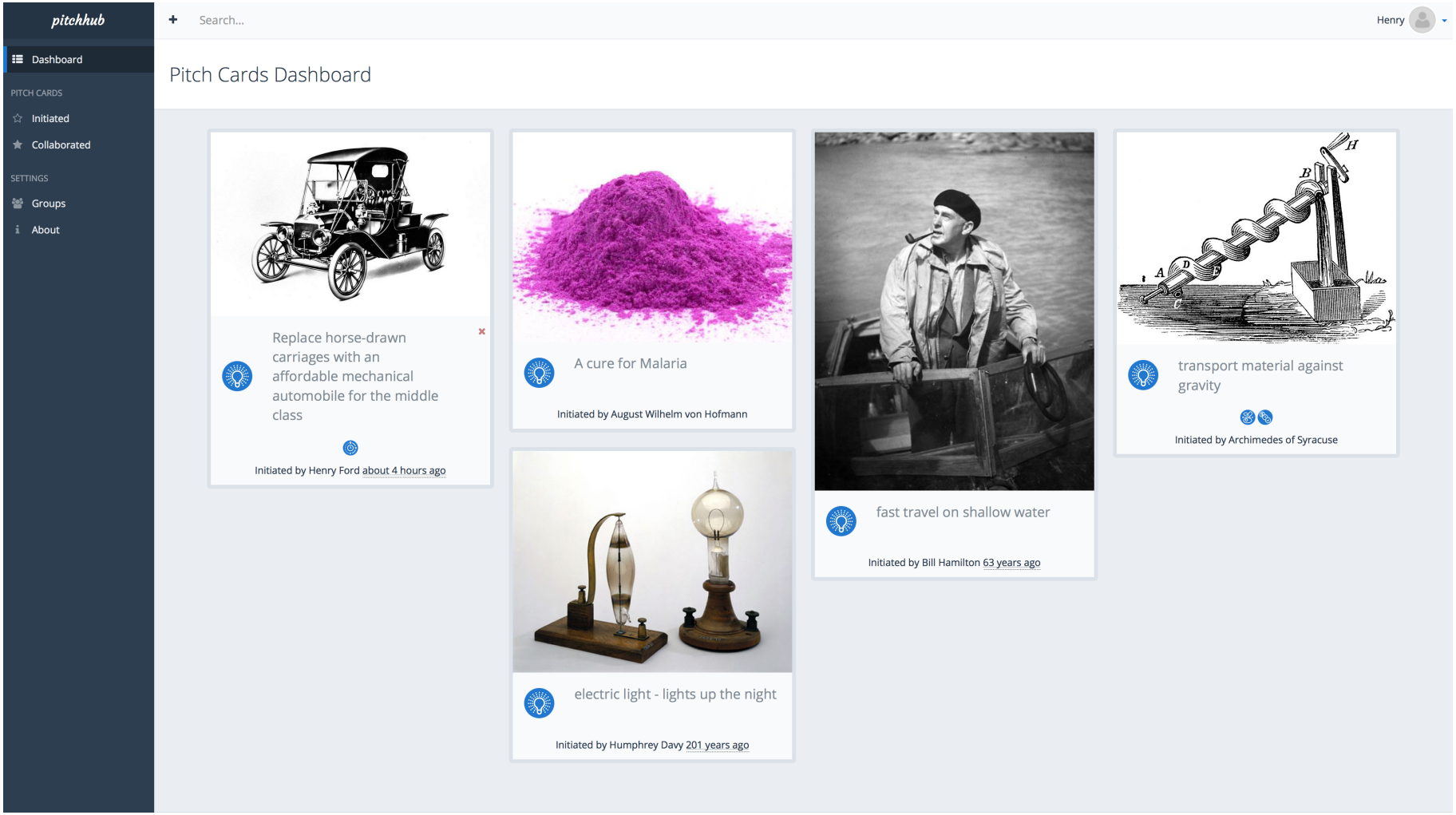

Figure 5.3: PitchHub's dashboard populated with fictional Pitch Cards. The grid layout displayed is responsive, so the Pitch Cards will reorganise and size to fit the user's device screen.

Chapter 6

Threshold Security Scheme

This chapters discusses the importance of database security, introduces threshold security schemes and describes how Shamir's Secret Sharing scheme has been integrated with PitchHub for the purposes of fulfilling design requirement D2.3: The prototype must store sensitive data securely.

6.1 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.

6.1.1 Background into 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.

6.1.2 Shamir's Secret Sharing Scheme

Shamir's secret sharing [43] 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 [43]. 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 [44][45]. 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 [46][47] and verification schemes that do not assume the party dealing the shares is honest or infallible [48][49].

6.2 Security Design

In this section discussion is presented on how Shamir's Secret Sharing scheme has been integrated into the PitchHub prototype.

6.2.1 Shamir's Secret Sharing Scheme Service

The first action in designing the service was choosing an open-source implementation of Shamir's Secret Sharing. Deciding to use an open-source implementation over writing a custom implementation 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 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." [50]. PitchHub approached the integration of the security scheme library with emphasis on limiting the amount of coupling. By reducing the coupling between the library and wider system the risk of system knowledge producing vulnerabilities within PitchHub and undermining the security scheme's integrity is also reduced.

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* [51] 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.

6.2.2 Overcoming Limitations of Threshold Security Schemes

Given share combinations up to k, Secret Sharing Schemes ensure that the secret's safety is maintained. 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 in the context of PitchHub: up to k compromised databases do not compromise the security of the secrets being held, but if an attacker is able to break into one database it is reasonable to infer that 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.

The first issue can be alleviated by introducing diversity to the secret keepers. Instead of all secret keepers being MongoDB instances including alternative data stores in the secret keeper configuration means that the vulnerabilities of the data stores themselves do not manifest as a vulnerability for the system as a whole. However, to do this effectively it was necessary to have a configuration where there is not k secret keepers of the same data store. If there were, the whole exercise of including alternative data stores would be rendered moot as an attacker with the ability to breach the particular data store would be able to fulfil the threshold k. This issue of diversity and security is explored by Littlewood and Strigini who conclude that depending on the context diversity can be one of the most effective means of neutralising 'systemic' attacks by merely avoiding the danger.

There is a fundamental tension in the relationship between availability and security in threshold schemes. Representations of the classic Threshold Security Schemes where each secret keeper contains one and only share of the secret favour security over availability. This is great when secret keepers do not have any membership changes or only member changes that still preserve k secret keepers. In this example should k or more secret keepers go down the entire service becomes unavailable, and in cases where k cannot ever be re-established then the secret data is irretrievable. There are a number of approaches that can be used to increase availability but each suffer from an increase in complexity and a decrease in security. For example, concepts used in distributed databases are directly applicable to secret sharing schemes. Redundancy and hinted hand-off are methods used by distributed database systems are used to ensure availability by node's sharing their data such that if they go down the data stored is still accessible through the nodes who hold the shared data. In the context of Threshold Schemes this means that some nodes may have more than one secret share for the same secret, this demonstrably reduces the significance of the threshold. For a system like PitchHub I believe classic redundancy, where each member contains a whole

other keeper's secret shares, is a step too far. Instead I suggest raising the availability of the secret members themselves by representing secret members as high availability (HA) replica sets rather than individual database instances, this architecture configuration is featured in Fig 6.1.

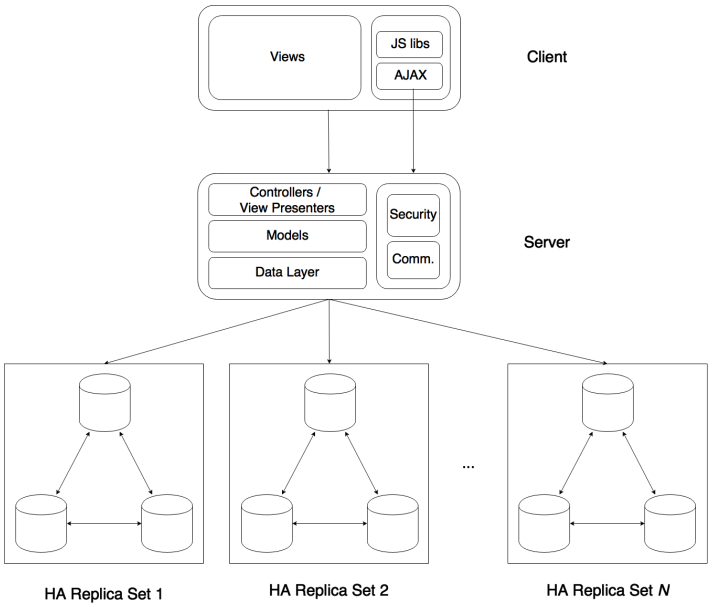

Figure 6.1: The architecture extended with high-availability (HA) replica sets increases the Threshold Scheme's robustness without compromising security.

6.3 Implementation

6.3.1 Shamir's Secret Sharing Scheme Service

Beyond the configuration of the service which is further explored in Subsection 6.3.2 an initial concern was in determining at what granularity the Pitch Cards/Contribution objects should be split. The options considered were as follows: split a binary representation of the object, split each field present in the object, or split only the sensitive fields in the object. The first option results in the entire object being encrypted with little ability to reason about the contents. This approach is considered too restrictive as data such as *user id*, *status*, and *timestamps* would have to be exposed to facilitate meaningful database queries. The second approach is better with regard to the fact that the structure of the data is reflected in the database however it would still require the same exposing of attributes as the first option. Ultimately, the approach for splitting objects was done on the field level, where each object's sensitive values are split and then placed into a new share that is persisted. Fig 6.2 visualises how this done on a comment object.

6.3.2 Diverse Secret Keepers

The secret sharing configuration developed for the prototype used a (*3*,*4*)-threshold scheme, where at least 3 of the 4 shares representing a secret must be combined to reconstruct the secret. The technologies used for the Secret Keepers are as follows: 2x MongoDB instances, and 2x PostgreSQL instances. The motivation behind using PostgreSQL instances to diversify the Secret Keepers is twofold. First, as discussed in Subsection 4.5.2 the relational

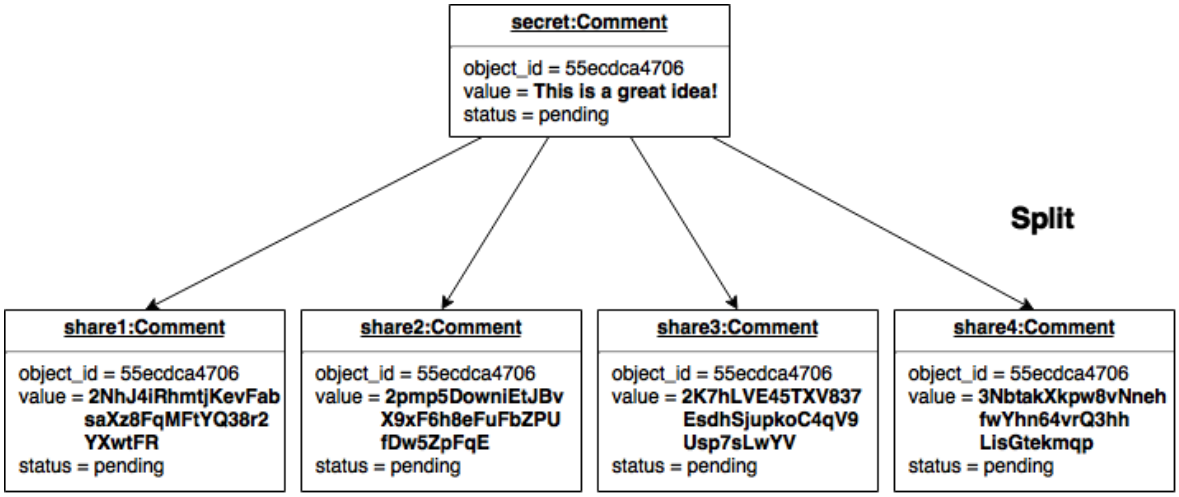

Figure 6.2: A comment object shown before and after the split operation. Each secret share contains an indecipherable portion of the original secret.

model is the ideal way to model N:M relationships users have with Pitch Cards and Suggestions/Comments. Second, the relational model is supported as the primary object-mapping system by Ruby on Rails. Given these reasons there are an abundance of advantages: modelling the data in a natural way that fits it's structure, better support throughout the development of the component, ability to prototype at a high velocity. Consideration was given to other document-model solutions such as CouchDB [53] and Couchbase [54] however these proved to have an underwhelming level of documentation for their ODM libraries.

An initial concern was that the upfront schema set-up and overall schema-strictness required by relational databases would impact PitchHub's portability and extensibility, important attributes captured in requirement *F4*. Thankfully, Ruby on Rails has out-of-the-box support for *migrations* which enable database schemas to be defined/altered incrementally. This is another case of PitchHub configuring infrastructure in code and opening the door for automation.

The most manual portion of setting up PitchHub at this stage is that the migrations required being manually executed on each database instance (as well as maintaining separate configuration files for each instance). To consolidate and automate this process PitchHub leverages the open-source community's *Octopus* library [55]. Octopus enables sharding and migration management for distributed database configurations. Integrating Octopus restores near 'plug-and-play' functionality. Still required is the actual setting up of the PostgreSQL instances. This can be done manually or via Vagrant but even this is considered to violate requirement *F4*, which identifies non-technical users being able to easily set up a local instance of PitchHub. To resolve this issue, the (development branch of) PitchHub makes use of SQLite [56], a self-contained SQL database. This configuration while not appropriate for production makes local instances of PitchHub with the secret sharing configuration easily accessible by non-technical users.

Chapter 7

Experiment Design and Test Bed

Following the completion and deployment of the prototype, an analysis of the artefacts produced was carried out. The evaluation was designed to verify the successfulness of the project against the requirements identified in Chapter 3. To build confidence in the results derived this chapter discusses the scope, assumptions, and limitations as well as the design of the experiments themselves.

7.1 Evaluation Scope

In designing the evaluation the requirements identified in Chapter 3 were deconstructed to clearly define, where possible, the conditions upon which each requirement can be considered fulfilled. The high-level results of this requirements deconstruction is displayed in the following table:

ID	Requirement	Fulfilment Criteria
D1	Enable the innovation community to collaborate	1. User stories describing collaboration in the innovation community are specified 2. These user stories specified are supported
D2.1	Enable users to scope the disclosure of their content/identity	1. User stories describing content/identity scoping are specified 2. These user stories specified are supported
D2.2	Provide auditing functionality	1. User stories describing auditing are specified 2. These user stories specified are supported
D2.3	Store sensitive data securely	1. A security expert verified the system
D3	Be portable and extensible	1. Discern from requirements
D4	Be performant	1. Measure(s) of performance are defined 2. These measures are met
D5	Support a distributed architecture	1. A distributed system is designed and implemented

Requirements D1, D2.1 and D2.2 have very similar fulfilment criteria and hence have been grouped together for the evaluation. Through this evaluation it is established that PitchHub has the required functionality to support collaboration within the innovation community. The experiment design and test bed implemented for these functional requirements are described in Section 7.3.1.

Requirement D4 relies on a simulation based evaluation, where synthetic data is seeded to imitate the load the entire innovation community in New Zealand could have on PitchHub. Requirement D5 is implicitly satisfied through this experiment as the simulation is deployed on distributed infrastructure. Through this experiment it is established that the

PitchHub application has the ability to support the innovation community. The experiment design and test bed implemented for these non-functional requirements are described in Section 7.3.2.

The breakdown's fulfilment criteria denotes the steps required to satisfactorily mark the requirement as completed, and as seen in requirement D2.3 and D3 some of these steps require external review. The implications of this are now discussed.

Achieving a conclusive degree of verification for requirement D2.3 would require a level of scrutiny and knowledge possessed only by a security expert. An evaluation of this requirement would require consideration of the following points: first, the cryptographic modules in the Secret Sharing Service must be verified and second, the security of the storage functionality cannot be considered in isolation. Should any combination of access control, communications, or session management components be compromised it is possible that the data protection component may also be compromised by association. Therefore evaluating whether "data is stored securely" must be done within the context of the wider system rather than specifically on the Secret Sharing Service. Verifying the Secret Sharing service's cryptographic implementation would take us further down the rabbit hole. OWASP's application security verification standard [57] details the minimum steps: all cryptographic modules fail securely, random number generators apply the appropriate standards, all cryptographic modules are validated against FIPS 140-2 or equivalent, all cryptographic modules operate in their approved mode according to their published security policies. Given the large burden to validate this requirement and the project's limited resources I regard the evaluation of requirement D2.3 out of scope. Cursory steps I have taken to build confidence in the prototype's security are as follows: Shamir Secret Sharing service, all requests enforce ssl, CSRF tokens to prevent CSRF, escaped user input to prevent XSS, white-list firewall set-up, *fstab* modification to prevent shared memory attacks, *sysctl* and host file modification to prevent IP Spoofing, and DenyHosts [58] integrated to prevent SSH attacks.

Assessing whether a system is portable and extensible is an inherently subjective endeavour. The fulfilment criteria for requirement D3 reflects this. A system can be regarded as portable if it works in all environments required of it, and likewise a system can be regarded as extensible given it is able to adapt to all it's future requirements. However, determining all the future environments and requirements of the system is a difficult task. PitchHub has therefore sought to achieve general portability and extensibility. The definition of portability adopted is: "the ease with which a system or component can be transferred from one hardware or software environment to another" [59]. As discussed in previous sections PitchHub has employed virtualisation and automation to make such tasks easy as possible. The definition of extensibility adopted is: "the ability of the system to tolerate additional features or functionality will little or no required rework of previously developed features or functions" [60]. PitchHub has been developed with this in mind, the layered architecture and MVC pattern followed mean that both hardware and software components are divided in terms of responsibility. This separation of concerns means that future change request modifications will be isolated to components whose responsibility fits. The inability to strictly quantify or qualify whether requirement D3 is met has led me to regard this requirement as out of scope for the purposes of this evaluation. However I believe the spirit of this requirement has been reflected in the system's design.

7.2 Assumptions and Limitations

@KB, SHOULD THIS BE AFTER THE EXPERIMENTS? -

The experiments undertaken have operated under a number of assumptions and limitations. These assumptions and limitations are explored in this section. Reflection on these assumptions and limitations are provided with discussion on their possible ramifications.

A limitation of the experiment described in Section 7.3.2 is that it seeded with synthetic data rather than user data. This is because no open-source user data currently exists within the domain of collaborative innovation networks[1]. To achieve a representative synthetic dataset I required: the number of people in the innovation community, to determine user base size, and also an idea of their platform engagement, to determine a distribution of the PitchCards contributed. Statistics New Zealand's (SNZ) data on innovation activities in New Zealand gives a picture of the innovation communities size from 2003 to 2013 [61]. SNZ's data is based on the population of businesses in New Zealand who categorise themselves as participating in "innovation activity". An important feature to note is that the data does not feature sole traders or businesses with less than 6 employees. SNZ's data categorises New Zealand businesses by size (e.g. 6 - 9 people, ... 50 - 99 people) and maps how many companies fit each range. This data presents three concerns: first, the small businesses not included may represent an appreciable portion of PitchHub's target audience. Second, the large ranges provided lowers the fidelity of the results derived. Third, by using the total business size the SNZ data includes people who does not directly have a stake in the innovation ecosystem (i.e. not resembling one of the roles identified in Section 2). These limitations have been deemed acceptable for the purposes of this evaluation because the goal is merely to build confidence that the prototype could work in production. Ultimately, this data gives a ballpark, albeit overly high, estimate of the size of the innovation community in New Zealand. The results from this analysis is presented in Table 7.1 and Fig 7.1.

Estimated total number of employees of businesses with innovation activities from the years 2005 to 2013		
Year	Expected Size	Worst Case Size
2005	453,619.5	613,929
2007	454,365	615,068
2009	431,442	578,196
2011	416,065.5	559,959
2013	431,524.5	579,549

Table 7.1: Estimates of total number of employees in businesses with innovation activities derived from Statistics NZ data.

The results shown identify the expected estimated size of the innovation community as well as the worst case estimated size (calculated using the upper range). The figures presented are believed to be quite high given that the current total workforce population of NZ is estimated to be 2.3 million people [62]. These figures being high however further build confidence that the prototype can support the NZ innovation community.

An informed estimation of the amount of contributions each user may have is inherently important to a performance evaluation where database queries make up the majority of processing time. To make the seed data more realistic a Pareto distribution, Fig 7.2, was chosen with Callaghan Innovation. The reason for such a distribution is that we conjecture that the vast majority of users will likely make only a few Pitch Cards and Suggestions, while a long tail of more avid users will be contributing a disproportionate amount of Pitch Cards and Suggestions relative to the entire user base. This conjecture is supported by recent work

[1] As of 29/9/15.

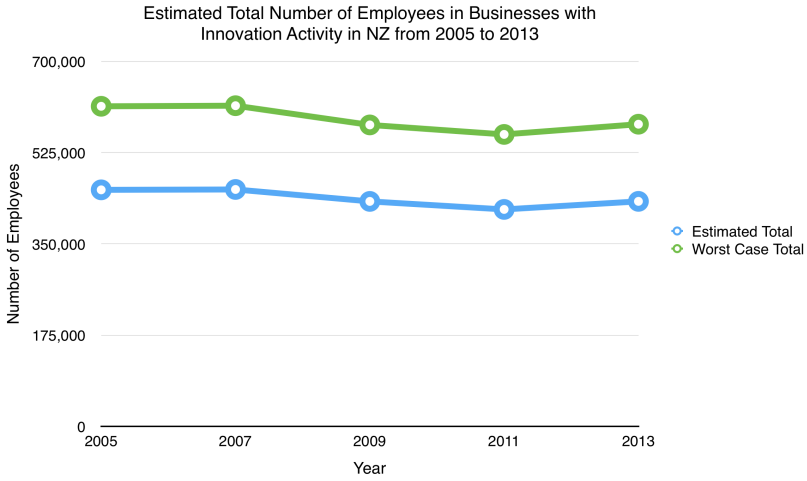

Figure 7.1: The estimated total number of employees of businesses with innovation activities from the years 2005 to 2013. The disparity between the expected sizes vs. worst case sizes is resultant from the wide range presented in the data.

analysing user engagement in online support groups [63][64]. The authors of both papers find that similar power distributions are able to characterise user engagement.

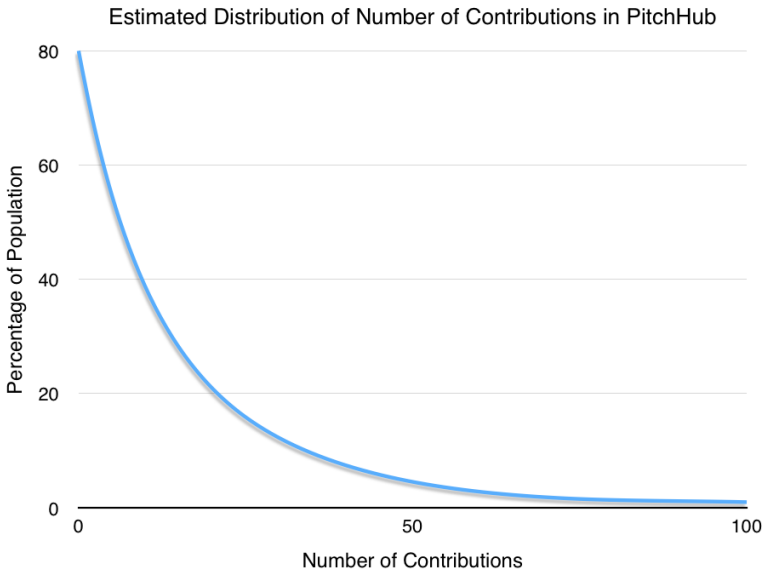

Figure 7.2: The assumed Pareto distribution describing user engagement with regard to Pitch Cards and Suggestions shared.

One of the key assumptions underpinning the PitchHub system is that the functionality specified in the user stories will actually facilitate collaboration within the innovation ecosystem. There has been no prior user study nor previous works which indicate that the collaborative approaches embraced (idea-deconstruction/scope of disclosure) by the prototype will either help or hinder the collaboration process. Based on Callaghan Innovation's experience in the innovation community and the current trend of platforms being released targeting innovators (e.g. IdeaForge and Assembly identified in Section 2) we believe that there is an inherent need for online collaboration within this community. Ultimately,

discerning whether the prototype produced has the ability to satisfy this need will be determined through experience. Currently the prototype has been bought-into by the wider Callaghan Innovation networks group. This is an ideal *proving ground* as it means that resources will be made available to continue PitchHub. Furthermore, the acceptance by a separate Callaghan Innovation group is a positive indication that the functionality supported is desired.

A limitation of the experiment design for the functional testing is that Selenium uses Firefox. Online sources agree that Google Chrome holds the dominating market share at around 60% as opposed to Firefox's share at around 20% [65]. Ideally, simulations would be run on all major browsers but as it stands the Firefox driver for Selenium is the most robust and complete. However, this limitation is considered to be acceptable given that browser vendor conformance to W3C standards is high [66].

7.3 Test Design

This section explores the three test beds designed for the evaluation. Each test bed was designed with different responsibilities in mind. The first, *TB1*, evaluates the collaboration functionality supported for requirements D1, D2.1 and D2.2 using executable user stories. The second, *TB2*, evaluates the the prototypes performance in various distribution configurations for requirement D4 and D5. The third, *TB3*, evaluates the collaboration functionality supported for requirements D1, D2.1 and D2.2 using a live deployment and real users. The design and aim of each test bed is discussed with reflection on how automation has also been used to improve their flexibility.

7.3.1 Test Bed 1 - Collaboration Functionality

The principal aim in testing the fulfilment of requirements D1, D2.1 and D2.2 is to ensure that users have the tools/functionality available to successfully collaborate online. Specifically what successful online collaboration is has been captured in the user stories developed by Callaghan Innovation and myself. The following is an example (concatenated for brevity) of the general granularity achieved in each user story:

```
Feature: Annotate Pitch Card
  As a user, I want to suggest changes on viewable pitches so that I can offer insight

  Background:
    Given following users exist:
      | username    | email             |
      | Foo         | foo@foo.foo       |
      | Bar         | bar@bar.bar       |
    And a user with email "foo@foo.foo" posts a well formed pitch card

  Scenario: suggest a change
    When I sign in as "bar@bar.bar"
    And I am on the first pitch card contributed by "foo@foo.foo"
    And I click "suggest" on the pitch point "solve"
    And I fill in the following:
      | change              | my suggested change                  |
      | comment             | reasoning for my change suggestion   |
    And I set the content scope of disclosure to "public"
```

```
And I set my identity scope of disclosure to "collaborators"
And I press "Submit"
Then I should see "my suggested change" within "discourses-content"
And I should see "reasoning for my change suggestion" within "discourses-content"
And I should see "less than a minute ago" within "discourses-content"
```

The test bed PitchHub uses for evaluating the functional requirements, *TB1*, 'executes' these user stories. It does so in exactly the same way users interact with the system through a browser. This has been achieved by using Cucumber, a Ruby behaviour driven development library, and Selenium, a popular web driver. To verify that these 'executable' user stories sufficiently simulate user behaviour let us explore the above example. The feature *Annotate Pitch Card* contains three main sections: a brief description of the test's aim, the background which specifies the beginning state of each test, and scenarios which are essentially the feature's tests. The content of the background and scenario blocks are interpreted via step definitions which use Selenium to then drive browser interaction (clicks, text input, etc.). For example, the line "When I sign in as bar@bar.bar" is interpreted through the step definitions into the following browser interactions: navigate to page `localhost:3000/users/sign_in`, fill email field, fill password field, click login button. A key advantage in having functionality testing automated in this way is that the state of the application is far simpler to reason about. At the start of each test the application is guaranteed to always be in the state specified in the background and throughout each scenario all statements following the 'then' keyword act as assert statements making in-test guarantees possible.

Ultimately, the aim of ensuring functionality is supported is well-suited to this type of testing for the following reasons: first, user interaction is simulated naturally, second, automation ensures that no lingering tests and their leftover states affect current tests, and third, the natural definition of these tests mean that non-technical users can specify and test behaviour.

7.3.2 Test Bed 2 - Performance

In evaluating performance there are a number of metrics that may be analysed e.g. latency, throughput, capacity. Ultimately, the wall-time metric was chosen because it reflects the performance of the system as experienced by the user. As PitchHub is a user-facing applications this was deemed to be the most important measure. To determine the prototype's fulfilment of requirements D4 and D5 the average wall-time to process a request in various configurations is evaluated. Wall-time is a key metric on which to evaluate performance as it represents the request's end-to-end processing time. This means that the time taken for server-side processing, database queries, and client-side processing are included in the results. This test bed, *TB2*, is used for several experiments: comparing the performance under different user base sizes, testing the overhead added through diverse secret keepers, and testing the overhead added of the Secret Sharing Scheme. Common amongst the tests is that they rely on the authenticity of the configuration/data used to derive meaningful results. To create an authentic test bed and environment several steps were taken:

- The application is hosted on a Passenger application server with production settings (e.g. caching).
- The tests simulate user input via the REST API.
- The test data size is modelled off of the Statistics New Zealand innovation reports.

- The infrastructure used reflects a highly available production configuration, see Fig 7.3.

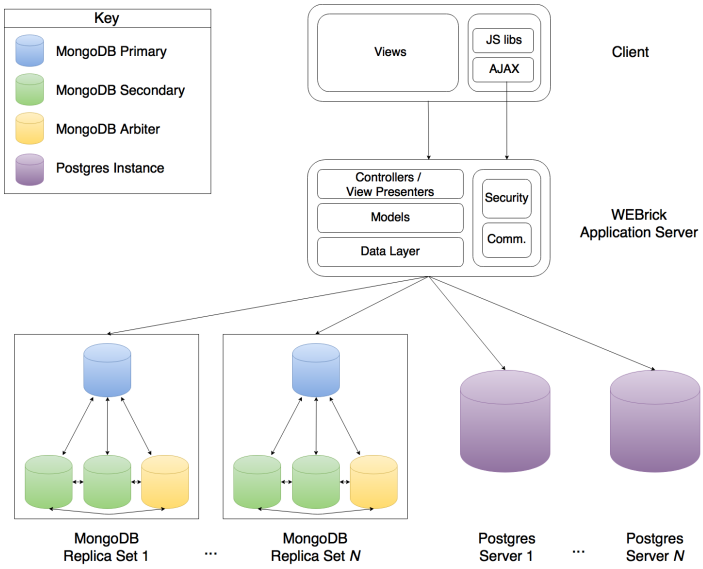

Figure 7.3: The architecture used for *TB2*. The MongoDB secret keepers are replica sets, the redundancy gained through this ensures high availability as discussed in Section 6.2.1.

The database configuration in Fig 7.3 reflects the general architecture utilised in the experiments. The n amount of MongoDB clusters and Postgres instances vary between tests to reflect the test's goal. For example, the test gauging the overhead of diversity in secret keepers compares a 2x MongoDB HA Replica Set, 2x Postgres instance configuration against a 4x MongoDB HA Replica Set configuration. In regards to the network topology, the MongoDB replica sets are hosted in AWS's Oregon region, the Postgres instances are hosted in AWS's Virginia region, and the application server is hosted on a DigitalOcean VPS in New York with 2GB of RAM and a 40GB SSD. Each AWS instance was hosted on Amazon's Elastic Compute Cloud (EC2) service with 2GB of RAM and a 3.3GHz CPU.

The collection of performance data is facilitated through the Rails-Perf-Test library [67]. The suite of integration-tests defined against this library are special in that they are specifically designed for benchmarking and profiling. For the purposes of the evaluation, emphasis has been placed on testing the more logically complex flows in PitchHub. Such flows include Pitch Card creation, Pitch Card display and the display of the Pitch Card dashboard. These flows were chosen as they require different actions in relation to the Secret Sharing scheme. Pitch Card creation involves the splitting of the secret into the n shares and saving these shares to n secret keeper databases. Pitch Card display, i.e. showing a single Pitch Card, involves at least k database requests followed by the reconstruction of the secret using at least k shares across each Pitch Point. Display Dashboard, i.e. showing a collection of Pitch Cards, involves at least k database requests followed by reconstruction of the secret using at least k shares across each Pitch Card's *value proposition* Pitch Point.

To increase the precision of the results the tests for each flow has been repeated 1000 times. In doing so the variability introduced by network latency is normalised. It should be noted that because caching is used in accordance to the prototype's production settings the first test run of each test takes significantly longer because the assets have not yet been cached. This test run's wall-time has been kept in the data set that is averaged for the final result because it is a valid run. This does present itself as an extreme value in each test's

data. The actual warm up times for each test have been excluded.

7.3.3 Test Bed 3 - Deployed Prototype

@KB, this section feels light, thoughts?

The goal of *TB3* is to analyse the usage of PitchHub by real people in the innovation community. To do this *TB3* consists of a deployed version of the PitchHub prototype which has been instrumented to capture all API requests. These requests are logged as user events containing the action with it's properties and metadata:


```
{
  "id":"8d4614a0-f9f2-47ae-983e-73c6ec4014b6",
  "name":"Processed dashboard#index",
  "properties":{
    "controller":"dashboard",
    "action":"index"
  },
  "visit_id":"7e9cc8d4-2a5d-4e9a-9697-1f3b14b610b5",
  "visitor_id":"4a41071c-e709-45ab-913c-eb0d890d10d6",
  "user_id":{"$oid":"560496e970697428b4000000"},
  "time":"2015-09-29T21:00:16.814Z"
}
```


Because of PitchHub's very specific target user group, meaningful data can only be derived from use by an actual innovation community. This is what motivated *TB3* to be released to the wider Callaghan Innovation organisation. Callaghan Innovation is regarded to be an ideal population to test the prototype against as all roles identified from Section 2 (challenger, solver, enabler, facilitator) are present within the organisation.

7.4 Response Time Thresholds

@KB, where would you advise this section go?

Evaluating the performance of the PitchHub prototype is done under various configurations using *TB2*. The measure upon which performance quality is based on is Nielsen and Miller's response time thresholds [68] for web application performance. The response time thresholds are as follows:

0.1 second is about the limit for having the user feel that the system is reacting instantaneously.

1.0 second is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay.

10 seconds is about the limit for keeping the user's attention focused on the dialogue.

Chapter 8

Evaluation

Following the completed implementation and infrastructure setup an evaluation of the project was carried out on prototype. The evaluation was designed primarily to verify whether or not the prototype developed was successful in fulfilling the requirements identified as in scope in Section 7.1.

8.1 Test I - Collaboration Functionality Fulfilment

8.1.1 Motivation

PitchHub's primary goal is to facilitate collaboration within the innovation community. To do this PitchHub seeks to engage all roles in the innovation community by focusing deconstructing ideas to appeal their expertise - in terms of value proposition, business opportunity, challenges, and solution. Test I seeks to verify that the prototype supports the behaviour necessary to facilitate this collaboration. As discussed in Section 3 this behaviour is distilled in the requirements D1, D2.1 and D2.2.

8.1.2 Results

The user stories which affirm the fulfilment of requirements D1, D2.1 and D2.2 in *TB1* confirm support of the following behaviour:

Requirement D1's user stories specify the ability to: post Pitch Cards, make suggestions on Pitch Points, accept/reject suggestions, comment on Pitch Points, comment on suggestions and mark Pitch Cards as completed/active.

Requirement D2.1's user stories specify the ability to: scope entities such as Pitch Cards, suggestions, and one's identity such that unintended viewers are avoided. To ensure the completeness of these user stories, each scope/viewer level combination is tested. These combinations are displayed in Fig 8.1.

Requirement D1's user stories specify that: users who have viewed a Pitch Card are able to be seen as a "viewer" by the PitchCard's initiator.

8.1.3 Discussion

The design of this experiment was guided by two main questions: first, "what behaviours are needed to facilitate collaboration in the innovation community?" And second, "how can

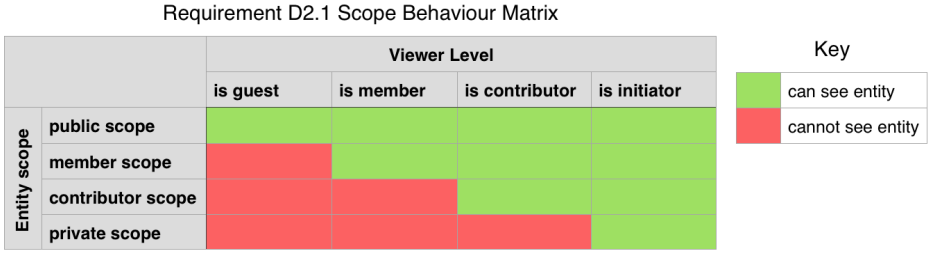

Figure 8.1: The scope matrix which illustrates the relationship between viewer level and entity scope in regards to viewing the entity.

these behaviours be verified as functional from a user's point of view?". In regards to the first question, determining what behaviours were required was to an extent already completed by Callaghan Innovation through their conceptualisation of PitchHub. To formalise these behaviours Callaghan Innovation and I collaborated on the specification of the user stories. In regards to the second question, realistically simulating user interaction was identified as key. Manually testing the various behaviours and scope combinations would be time-consuming and tedious. *TB1* automates this simulation with the added benefit of assertions, making it easy to verify that expected behaviour is indeed satisfied. Concluding on the fulfilment of requirements D1, D2.1 and D2.2 largely relies on whether the behaviours specified in relation to the first question accurately reflects the behaviour required by the innovation community. Generalising what a large population needs is a difficult task but it is my belief that because the user stories tested were created in collaboration with Callaghan Innovation, who is a large participant in the innovation community, they are therefore meet the level of credibility required for the purposes of this test. As such, it is concluded that requirements D1, D2.1 and D2.2 have been fulfilled.

8.2 Test II - Deployed Prototype Analysis

At the time of writing, the prototype, *TB3*, has been successfully been released to Callaghan Innovation and is ready for use. Unfortunately use of the prototype by Callaghan Innovation has not yet fully commenced. Without substantial activity to analyse this test was unable to be completed. It is expected that the access code will be distributed by the Callaghan Innovation stakeholders to the wider organisation in the coming weeks.

8.3 Test III - Community Size Support

8.3.1 Motivation

Supporting collaboration within the innovation community firstly requires that sheer size of the community is able to be supported. Test III seeks to test whether the PitchHub prototype could support the entire NZ innovation community. The sizes tested against are derived from the Statistics NZ data, where the expected size is estimated to be a population of 400,000 and the worst case size is estimated to be a population of 600,000.

Test III uses the prototype on *TB2* in a diverse "3, 4" secret sharing scheme configuration (2x MongoDB replica sets and 2x Postgres instances).

8.3.2 Results

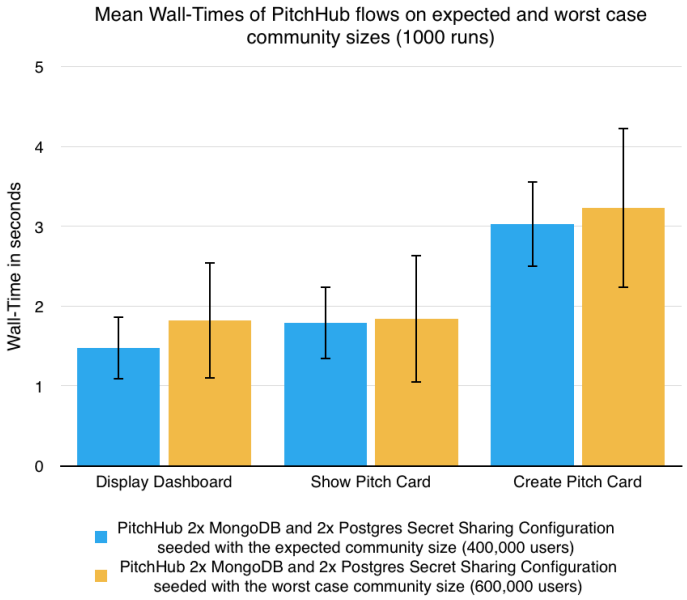

Figure 8.2: The mean response times with standard deviation for PitchHub flows: display dashboard, show Pitch Card, and create Pitch Card, under both expected and worst case community sizes. There is a very negligible performance difference between the two community sizes. Each size elicits mean response times that fall under Nielsen's third threshold.

8.3.3 Discussion

Fig 8.2 illustrates that the PitchHub prototype in a 2x MongoDB and 2x Postgres Secret Sharing configuration can support the innovation community at both the expected and worst case sizes (derived from the Statistics NZ data in Table 7.1). In terms of Nielsen's response thresholds, each flow falls under Nielsen's third threshold (1 =<10 seconds): where a users attention is still maintained but the wait-time interrupts flow of thought. Given that each flow in both community sizes meets the acceptable response threshold it can be concluded that the prototype satisfies requirement D4.

The difference in community sizes shows only a negligible performance difference in mean wall-time for each flow. This is because, even at the worst-case size, the database is only medium in size - where the entire database is storing between 10^5 and 10^7 records and can still fit on a single server [69]. At this size, performance is not affected by amount of data stored.

8.4 Test IV - Overhead of Threshold Scheme Security

8.4.1 Motivation

The secret sharing service is an important part of keeping sensitive user data secure. A consequence of this service is a loss of performance - primarily a consequence of the latency introduced by the queries to the n secret keepers and added processing on the application server in reconstructing the secrets. Test IV seeks to measure the overhead of this service and discuss whether the security afforded is worth the performance loss.

8.4.2 Results

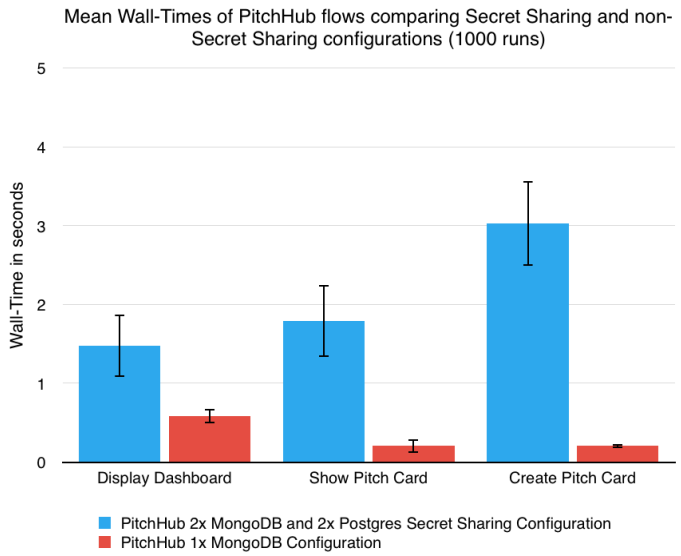

Figure 8.3: Comparison of the mean response times with standard deviation for PitchHub flows: display dashboard, show Pitch Card, and create Pitch Card, in both Secret Sharing and non-Secret Sharing configurations. There is a very significant performance difference between the two configurations. The non-Secret Sharing configuration falls under Nielsen's second threshold whereas the Secret Sharing configuration falls under the third.

8.4.3 Discussion

Fig 8.3 illustrates the difference in mean wall-times between Secret Sharing and non-Secret Sharing configurations. It is clear that the Secret Sharing service incurs a significant loss in performance. The most prominent case is the Create Pitch Card flow, where the Secret Sharing configuration is 14x slower. There are a number of important differences between the two configurations that contribute to this loss. The most important of which is operations involved, the Create Pitch Card flow in the non-Secret Sharing configuration is essentially an insert statement followed by a redirect to the newly saved Pitch Card, whereas in the Secret Sharing configuration the Pitch Card object is split into n shares, k-1 of these are adapted to Postgres' ORM format, each share is saved to their respective secret keepers, and on completion of the n saves the action is concluded with a redirect to the newly saved Pitch Card. The Secret Sharing configuration is simply doing more and is therefore taking longer. Determining whether this overhead is acceptable is an inherently subjective task. Strictly in light of Nielsen's thresholds it is not worth it. As the non-Secret Sharing configuration meets the second threshold while the Secret Sharing configuration slips to meet the third. But ultimately, the question that must be asked is: "Is the noticed delay and interrupted flow of thought worth the security afforded by the Secret Sharing scheme?". Based on the nature of the data being served I believe that this overhead is indeed worth it. To view the problem from another perspective, it seems reasonable to trade 2 seconds in delay for assured security in the event that secret keepers be breached.

8.5 Test V - Overhead of Secret Keeper Diversity

8.5.1 Motivation

To strengthen the security afforded by the secret sharing service diversity was added to the secret keepers. A consequence of this is that the service is now only as fast as it's slowest secret keeper. Test V seeks to measure the overhead of this added diversity and discuss whether the extra security afforded is worth the performance loss.

Test V uses the prototype on *TB2* in a both a diverse "3, 4" secret sharing scheme configuration (2x MongoDB replica sets and 2x Postgres instances) and a homogeneous "3, 4" secret sharing scheme configuration (4x MongoDB replica sets).

8.5.2 Results

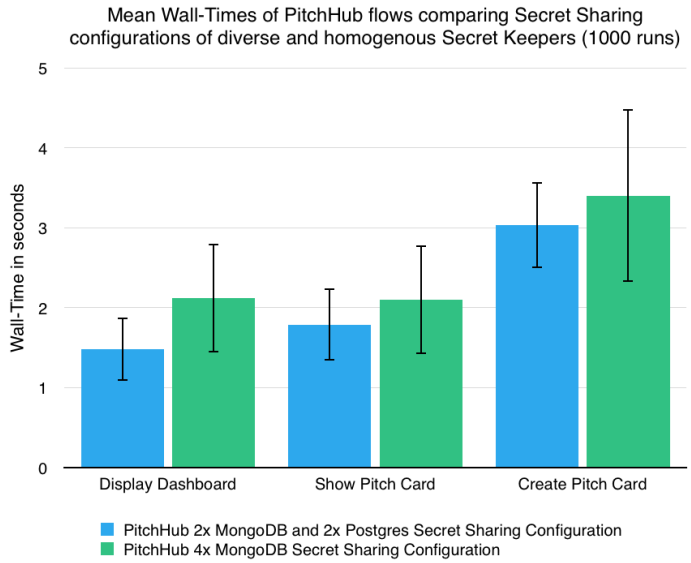

Figure 8.4: Comparison of the mean response times with standard deviation for PitchHub flows: display dashboard, show Pitch Card, and create Pitch Card, in both diverse and homogeneous secret keeper configurations. There is a noticeable performance difference between the two configurations where the diverse configuration performs faster. Both configurations fall under Nielsen's third threshold limit.

8.5.3 Discussion

Fig 8.4 illustrates the difference in mean wall-times between Secret Sharing configurations with diverse and homogeneous secret keepers. The diverse configuration is consistently faster than the homogeneous configuration across all flows. The 10% increase is likely because of the choice of secret keepers used - since MongoDB is less performant than Postgres in both read and write operations [70]. A disadvantage with the Postgres instances is that they do not provide the same assurance of availability as the MongoDB replica set secret keepers. This may present itself as a problem should the Postgres instance become unavailable resulting in it's shares becoming unavailable and hence harder to meet the required k shares needed for reconstruction. Overall, these results indicate that this particular diverse configuration is worth the extra implementation effort as it not only solves the Secret Sharing scheme's redundancy issue but also performs better.

8.6 Test VI - Concurrent Load Support

8.6.1 Motivation

Beyond being able to satisfy individual requests within an acceptable threshold it is important the PitchHub is able to do this while under the load of the innovation community. Test VI aims to assess the extent of the load that can be sustained by *TB2* in a 2x MongoDB and 2x Postgres Secret Sharing configuration. This test specifically looks at the load that can be supported by the Show Pitch Card flow as this is assumed to be the most frequently requested flow.

8.6.2 Results

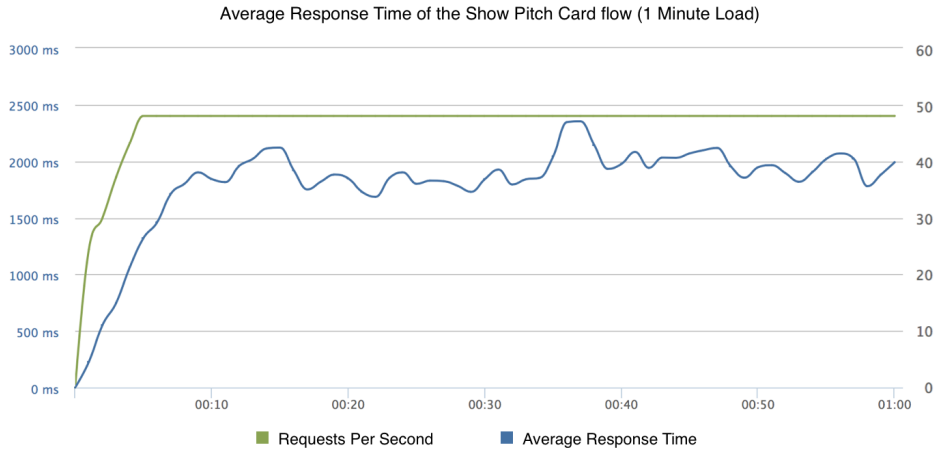

Figure 8.5: Load test of the Show Pitch Card flow in a 2x MongoDB and 2x Postgres Secret Sharing configuration. At 50 requests per seconds the prototype is able to maintain a 2 second response time average.

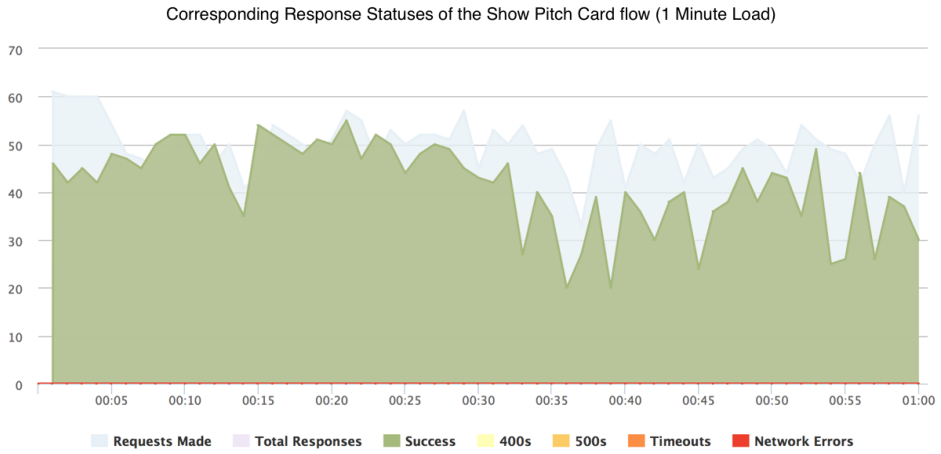

Figure 8.6: Corresponding response statuses of the Show Pitch Card flow load test conducted in Fig 8.5. This graph demonstrates that the 2x MongoDB and 2x Postgres Secret Sharing configuration on *TB2* can reliably withstand a load of 50 requests per second.

8.6.3 Discussion

From the data in Fig 8.5 and Fig 8.6, it is apparent that *TB2* can reliably sustain a load of 50 requests per second while maintaining an average response time of 2 seconds. This is significant as it reveals that the individual response times found in Fig 8.2 can be sustained at load. Beyond 50 requests per second *TB2* begins to return error code 500s, meaning that the server knows it has had a problem fulfilling the request. Determining exactly how many users 50 requests per second may support is difficult to estimate. Frequency analysis on Fig 8.2's results would have been helpful in identifying the community's usage patterns. Nevertheless, without this data, the following assumptions have been made: the load is distributed evenly, each user uses PitchHub once a day, and each user session is 10 requests deep (page requests). Calculating the necessary requests per second for the expected and worst case community sizes using these assumptions results in the following:

Size	Required Requests Per Second (rps)
Expected	45 rps
Worst case	70 rps

Based on these assumptions *TB2* can therefore satisfy the load of the expected community size but not the worst case. To increase the load supported *TB2* could be either scaled vertically or horizontally, such that the application server size is increased or a load balancer and other servers are added. One major drawback of the assumptions is that the load is unlikely to be even. Peaks in usage will occur throughout the day and it is this maximum peak load that will need to be satisfied. Once *TB3* has gathered sufficient data, these peaks may be modelled, and subsequently more accurate load estimates can be made.

Chapter 9

Conclusions and Future Work

This chapter reviews the project's contributions in light of the design requirements, discusses potential future work and finally, surmises the project.

9.1 Review

The project's principal contributions are as follows:

C1: Innovation Role Taxonomy. TODO

C2: UI design. TODO

C3: Architecture. TODO

C4: User stories describing collaboration in the innovation community. Existing user stories of collaboration in the innovation community do not exist. As such new user stories were specified with an emphasis on networking/collaborating around ideas. These stories specify the following: engagement of all roles within the innovation community, the ability to scope content's disclosure, and audit who has seen ideas that you have initiated.

C5.1: A PitchHub prototype. Built on the popular Ruby on Rails framework, the prototype was developed to support the user stories specified. Having followed good coding practices this artefact is open to future extensions.

C5.2: A PitchHub prototype deployed to Callaghan Innovation. For internal use within Callaghan Innovation a prototype was deployed following a production configuration: Passenger application server, caching, SSL, and various security measures suggested by OWASP (e.g. ssh hardening, protection against IP spoofing).

C5.3: A PitchHub prototype with Shamir's Secret Sharing scheme integrated. Use of the Shamir's Secret Sharing scheme, most commonly used in key services, was integrated into the PitchHub prototype. Sensitive information such as Pitch Points, suggestions, and comments is encrypted using this service providing unconditional security.

C5.4: A PitchHub prototype with Diverse Secret Keepers. In strengthening the security provided by the Secret Sharing scheme, diverse secret keepers were introduced. PitchHub supports the following databases: MongoDB, Postgres, MySQL and SQLite. To maintain *plug-and-play* functionality afforded by MongoBD the configuration of SQL secret keepers has been automated.

C6: Virtualised development environment. To make installation and setup of PitchHub as easy as possible the development environment has been virtualised using Vagrant and Chef. This was primarily for the benefit of non-technical stakeholders, but will be equally useful for future maintainers.

C7: Automated deployment process. To make host configuration and code deployment as easy as possible the deployment process was automated using Capistrano. Deployments can be triggered effortlessly and use GitHub to ensure production reflects the Master branch. Beyond this deployments can be rolled-back easily, this will be useful in the future should any deployments prove to be defective.

TODO review in light of design requirements

9.2 Future Work

This section discusses potential extensions to the PitchHub prototype.

9.2.1 Recommendation Engine

A potentially useful feature would be if Pitch Cards could be recommended to users based on their ontological profile. This would encourage collaboration through showing users ideas they are more likely to have an interest in. Offline learning is certainly viable, but an interesting challenge would be integrating the recommender to work with the Secret Sharing service in an online learning approach.

9.2.2 Usability Extension/Evaluation

TODO

9.2.3 RealMe Integration

TODO

9.3 Summary

TODO

Bibliography

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

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

[3] 100open, "100% open," http://www.toolkit.100open.com, (Visited on 04/07/2015).

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

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

[6] "Quirky," https://www.quirky.com, (Visited on 04/07/2015).

[7] C. Thompson, "Businesses facing increasing cyber threats: Security experts," http://www.cnbc.com/id/100421313, January 2013, (Visited on 04/07/2015).

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

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

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

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

[12] 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.

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

[14] 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.

[15] 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.

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

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

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

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

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

[21] "New zealand healthcare innovation — linkedin," https://www.linkedin.com/groups/New-Zealand-Healthcare-Innovation-2035021/about?report, (Visited on 08/26/2015).

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

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

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

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

[26] "Site speed: case studies, tips and tools for improving your conversion rate — econsultancy," https://econsultancy.com/blog/10936-site-speed-case-studies-tips-and-tools-for-improving-your-conversion-rate, (Visited on 09/09/2015).

[27] "The average web site load time is a sluggish six seconds - infographic - the american genius," http://theamericangenius.com/social-media/the-average-web-site-load-time-is-a-sluggish-six-seconds-infographic/, (Visited on 09/09/2015).

[28] "How loading time affects your bottom line," https://blog.kissmetrics.com/loading-time/, (Visited on 09/09/2015).

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

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

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

[32] 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.

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

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

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

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

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

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

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

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

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

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

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

[44] 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.

[45] 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.

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

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

[48] 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.

[49] 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.

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

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

[52] B. Littlewood and L. Strigini, "Redundancy and diversity in security," in *Computer Security–ESORICS 2004*. Springer, 2004, pp. 423–438.

[53] "Apache couchdb," http://couchdb.apache.org/, (Visited on 09/13/2015).

[54] "Accelerate web, mobile, and iot applications with the world's fastest nosql database." http://www.couchbase.com/, (Visited on 09/13/2015).

[55] "tchandy/octopus - github," https://github.com/tchandy/octopus, (Visited on 09/13/2015).

[56] "Sqlite home page," https://www.sqlite.org/, (Visited on 09/13/2015).

[57] "https://www.owasp.org/images/5/58/owasp_asvs_version_2.pdf," https://www.owasp.org/images/5/58/OWASP_ASVS_Version_2.pdf, (Visited on 09/27/2015).

[58] "Denyhosts - wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/DenyHosts, (Visited on 09/27/2015).

[59] M. Mattsson, H. Grahn, and F. Mårtensson, "Software architecture evaluation methods for performance, maintainability, testability, and portability," in *Second International Conference on the Quality of Software Architectures*. Citeseer, 2006.

[60] "What are good ways to measure extensibility? - quora," https://www.quora.com/What-are-good-ways-to-measure-extensibility, (Visited on 09/27/2015).

[61] "Innovation in new zealand: 2005," http://www.stats.govt.nz/browse_for_stats/businesses/business_growth_and_innovation/, (Visited on 09/29/2015).

[62] S. NZ, "Key labour force measures by qualification, age and sex," http://nzdotstat.stats.govt.nz/wbos/Index.aspx?DataSetCode=TABLECODE7080, (Visited on 10/06/2015).

[63] B. Carron-Arthur, J. A. Cunningham, and K. M. Griffiths, "Describing the distribution of engagement in an internet support group by post frequency: A comparison of the 90-9-1 principle and zipf's law," *Internet Interventions*, vol. 1, no. 4, pp. 165–168, 2014.

[64] T. van Mierlo, D. Hyatt, and A. T. Ching, "Mapping power law distributions in digital health social networks: Methods, interpretations, and practical implications," *Journal of medical Internet research*, vol. 17, no. 6, 2015.

[65] "Browser statistics," http://www.w3schools.com/browsers/browsers_stats.asp, (Visited on 10/05/2015).

[66] "Web browser standards support," http://www.webdevout.net/browser-support, (Visited on 10/05/2015).

[67] "rails/rails-perftest github," https://github.com/rails/rails-perftest, (Visited on 10/08/2015).

[68] J. Neilsen, "Response times: The 3 important limits," http://www.nngroup.com/articles/response-times-3-important-limits/, January 1993, (Visited on 04/07/2015).

[69] "Large database — scaledb extending mysql for volume and velocity," http://www.scaledb.com/large-database.php, (Visited on 10/12/2015).

[70] J. S. Van der Veen, B. Van der Waaij, and R. J. Meijer, "Sensor data storage performance: Sql or nosql, physical or virtual," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 431–438.