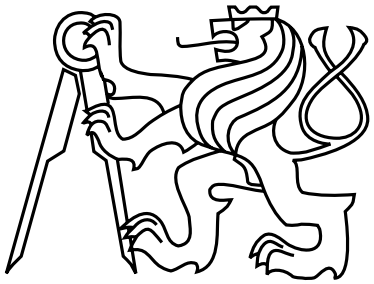
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering



Master's Thesis

Semantic Data Analysis and Visualization of User Interactions

Bc. Michal Švácha

Supervisor: Ing. Ivo Malý Ph.D.

Study Programme: Open Informatics

Field of Study: Software Engineering

May 26, 2016

Aknowledgements

I would like to thank dearly to my cat, dog and goldfish I never had. You were a true inspiration to me. With you, I would have never made this document happen. Also, my PlayStation 3 proved to be an amazing tool to keep me sane while writing this thesis. Thank you SONY.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.
I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Prague on May 27, 2016 .....

Abstract

The current status quo in data collection is to collect everything that is available. This approach has given rise to a trend in the last couple years - "Big Data". Data that is inconveniently large for processing, interpretation and inference. When a company decides to leverage big data, it usually ends up having too much information and no real business value. Each department uses different domain vocabularies and there is therefore no synchronization or understanding. This master's thesis is trying to take the opportunity of this disorder and connect the endpoints together while leveraging big data to provide an end-to-end solution.

The focal part of this thesis is the focus on user interactions - the collection of data from mobile applications. Before such collection can even happen, the interactions must be defined - the "what", "when" and "why". Starting with management, over to architecture and engineering to interpreting results as the destination - uniting all steps to form a bigger picture.

Abstrakt

Momentální status quo ve sběru dat je sbírání a ukládání všeho, co je k dispozici. Tento přístup dal vzniknout trendu posledních let - "Velká data". Tedy data, která jsou nepohodlně objemná pro zpracování, výklad, a odvozování závěrů. Když se společnost rozhodne, že chce využít velká data, dopadne to většinou tak, že má příliš mnoho informací bez žádné reálné hodnoty. Každé oddělení používá vlastní doménové názvosloví a tudíž chybí synchronizace a porozumění. Tato diplomová práce se snaží využít této příležitosti neuspořádanosti pro spojení všech konců dohromady a vytvořit tak komplexní řešení.

Těžištěm práce je zaměření se na uživatelské interakce - sběr dat z mobilních aplikací. Než nějaký sběr vůbec nastane, je třeba mít interakce definované - tedy "co", "kdy" a "proč". Počínaje projektovým managementem, přes architekturu a softwarové inženýrství až k interpretaci výsledků jako konečným bodem - spojení všech kroků k vytvoření uceleného náhledu.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In lean/agile product development, it is necessary to have formalized user feedback loops in place, to measure product performance against various (quantitative) metrics. Such feedback loops obtain the information and statistics regarding user engagement and interaction with the developed product. Are the users using it the way the creator imagined or did they find any other means of utilizing it? What stops the users from doing the task they intended to do? Do they get everything they need, and at the same time, does the creator get what was expected? In other words, are the dominant means of usage incentive compatible for the users?

Not only do the current solutions for gathering such user data for both quantitative and qualitative metrics work out-of-the-box with low-level semantics only (everything is a general activity on a general resource), but they also tend to run on somebody else's servers. What if the product developer is in a highly regulated market, such as pharmaceuticals, and has to own all their users' data? How can the current solutions' space be utilized and tweaked in order to fit such a schema?

Data Disconnectivity

The most problematic issue in large corporations is the semantic disconnectivity of the data. This occurs when data is collected ad hoc without any set direction or goal to be achieved. It may physically be all there, however nobody knows what or how it should be connected in order for it to make sense and drive value. Do I have good data or do I just have petabytes of useless log trace? Am I gathering information on what the application was intended to do or am I only filling the database with irrelevant garbage information. Most importantly though, am I gathering the information I need in a consistent fashion that corresponds to the domain of the shareholders?

Let's draw an analogy here and let me illustrate the problem with a real-world example.

Example

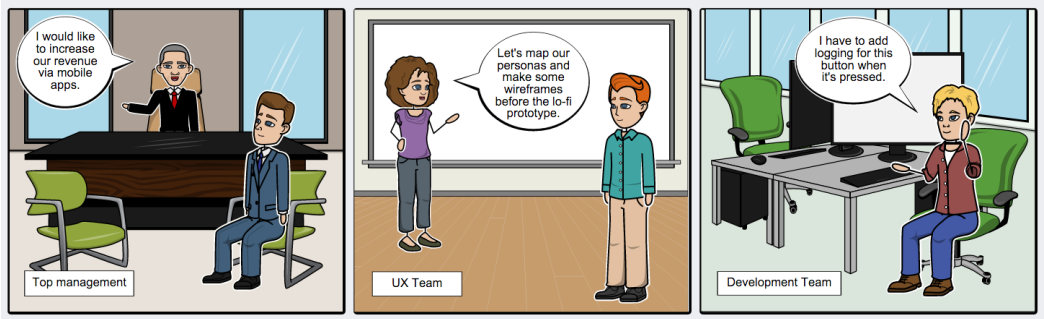


Figure 1.1: Real-World Example

This situation is mainly caused by:

1. Top management comes with a need to increase revenue via mobile devices.
2. Project management team takes over and breaks it down to user-stories, tasks etc.
3. Project management team asks the UX team, the mobile applications team and the backend team to perform their tasks in order to bring the product to life.
4. By the time product development is spread across three different teams, it is very much likely that each team will create their own jargon for every activity performed in the product.
    - The UX team is driven by the larger picture so they tend to refer to objects in a highly abstract way.
    - The backend team is driven by the inner processes happening between the database and the REST API, so they speak their precise technical language.
    - The mobile applications team is somewhere in-between, but never really aligned with either of the other teams.
5. When the product is delivered, the project management team has a really hard time translating the reported results into the top management's primary goals. Plus, the larger the teams are, the worse the situation actually gets.

This isn't a problem of poor project management. The company can use the best project management tools on the market and have the smartest people working on their teams and still encounter this problem on a daily basis. This is a larger problem resulting from the lack of intersection between different domains (Management, UX, Development, DevOps).

1.1 Work Environment

The context of this work is a large company, operating in a regulated market, where products are developed across multiple departments varying not only in size and experience, but also in project management approach - traditional and agile methods. Both methods implemented in the organization have a common denominator: measured KPIs to report the status of the work.

1.1.1 Key Performance Indicators

Each department/team/person has very specific Key Performance Indicators (= KPIs) [17] that represent the value of their work. Even products have their KPIs - "How much value has this product brought?", "How much time is this product saving us?", etc.

1.1.2 Software Development Life Cycle

Software Development Life Cycle (= SDLC) is a standardized traditional and formalized approach to developing a software product for regulated markets. It enables companies to follow specific steps in order to get their software product certified for use in constrained environments. It is used not for the sake of bureaucracy, but to protect the customers and increase the level of traceability of a problem, should one ever occur.

1.1.3 Agile Development Methodologies

In the modern era of software development, it has become "cool" to not follow traditional waterfall models and to adopt agile methodologies, such as scrum or kanban. For many years these were found exclusively in start-ups. But because time and time again, research [6] has shown that agile methodologies do impact the level of productivity and innovation, larger companies are working on adopting them in their constrained context as well. This thesis operates in such a place, where agile methodologies are being deployed across the whole innovation department.

1.2 Personal Interviews

In order to find out what needs there are, I have conducted several interviews with leaders of various departments. These are their reactions to what bothers them about product monitoring during the development phase:

1. Associate Director, Applied Technology

   > "The real problem I see is the fact that all the information I need is on somebody else's server. We can't store any sensitive, let alone confidential information *somewhere* with some random vendor. It's actually illegal in some countries. Unfortunately, sometimes sensitive data is exactly what we need to obtain from the applications to make an informed decision."

2. Associate Director, Mobile and Web

   "Our needs for tracking KPIs are variable throughout time and unfortunately the current tools we use are quite inflexible. Because we are in a regulated market, each change that requires a new build of an application takes a longer period of time. And time *is* money."

3. Mobile Application Development Lead

   "I have noticed that one of the biggest obstacles is how should we name what we measure. I have no vocabulary to help me during the development. The only thing we have is a robust Google Analytics toolkit that only allows us to gather low-level actions. We can log that a button was pressed, but what do we name such action? "Button pressed"?"

4. UX Lead

   "Our team looks at the high-level needs. We are trying to make the user activities in an application as smooth as possible. When we design a low-fidelity prototype, we know what we want to measure. Having the opportunity to add a high-level KPI would help us a lot in order to gather feedback for our prototypes. We are not programmers, we don't know how to add it to the code, but I would love the idea of including what to measure along with the prototype."

1.3 Scope of Work

First, I will present the context of agile software development in a regulated market. Then I will take a look at the current tracking solutions being used in mobile development (the focus is on iOS development, but most of the tools are multiplatform solutions). I will examine and analyze their strengths and weaknesses.

Next I will propose a workflow to fit the needs of a larger company with multiple departments, operating in a regulated market. I will utilize existing tools and build on top of them in order to drive value without reinventing the wheel couple times.

Lastly, I will develop a working PoC (Proof of Concept), verify its functions through user testing and further discuss with the stakeholders whether or not it is the correct path to take in order to unite all departments and connect the dots in the data.

Chapter 2

Analysis

2.1 Product Opportunity Assessment

Product opportunity assessment is a helpful procedure to separate the wheat from the chaff. It points out the real pressing issues that are mission critical for the company.

1. *What problem are we trying to solve?*

   Enabling alignment of business language and user interaction reporting is a key part to a success of a new product. No commercially successful platforms allow such alignment, not even any higher semantic analysis.

   In a regulatory market it is vital to have an option of storing the user data on custom servers. No such option along with the higher-level analysis is currently available at the market.

   There is no standardized toolkit and thus the lack of interchangeability between tools tends to end up with a vendor lock-in.

2. *For whom do we solve that problem?*

   For all departments in the company that are participating in product development, top to bottom. From setting a business goal to defining the needs, while also helping the developers and the designers.

3. *How will we measure success?*

   Having gathered data that is securely placed on custom servers. Data that is united and connected by shared domain vocabulary, ready to be analyzed and visualized in order to draw conclusions.

4. *What alternatives are out there?*

   There is some proprietary software on the market, but none is flexible enough for the defined needs.

5. *Why now?*

   The competitors [10] are really getting into the data-driven decisions. The intention is not to gather as much data as possible, but to generate valuable insights as soon as possible in order not to fall behind the competition.

6. *What factors are critical to success?*

   Alignment of all departments participating in software development in a non-invasive fashion. It is critical to primarily focus on helping the departments to connect with each other.

   The ability to run the whole solution on custom servers and have total control over all gathered data.

2.2 Regulated environment constraints

Regulated markets, especially pharmaceuticals have multiple rules that need to be carefully followed in order to be allowed to use their new IT products. Not only is important how much value does the final product bring to the end-user, but also how data storage is handled and how prone it is to exploitations and attacks.

From the development process point of view it is equally important to follow specific guidelines and processes during the development phase. Each step has to be carefully documented and approved by specific audit department. For that reason, most of the big pharmaceutical companies use the waterfall model implemented in SDLC. It enables companies to follow these specific steps in order to get their software product certified. This all hassle is not for the sake of bureaucracy, it is to protect the customers and increases the level of tracability of a problem, should one ever occur. After all, it is theirs and their customers' private data that goes on the server, so it is imperative that it is safe.

The limitation of this approach is the inflexibility of the waterfall model. The strict and rigorous requirements for the waterfall model are not allowing to go back half way through and re-define the objectives of a product. Therefore more and more large companies are trying to implement the best of both worlds.

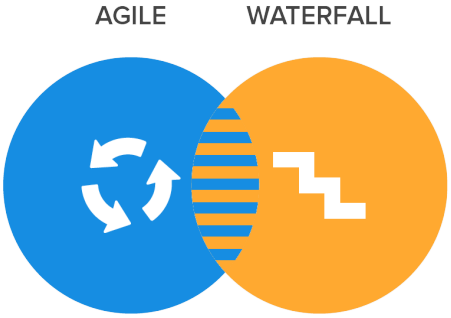


Figure 2.1: Balanced Approach to Waterfall and Agile Methodologies (source [11])

In the context of my work, the approach is that prototypes, PoCs and internal beta versions of new software products are developed using various Agile Methodologies (Scrum and Kanban) and once their purpose is verified, the work done is considered as the starting point for the SDLC.

2.3 Current Workflow

Currently, the top management is in the USA, while the innovation department is in Prague, Czech Republic. Some members from the top management occasionally fly to Prague, but very irregularly and mostly for a quick check on the overall status of the department. Most of the communication happens remotely - via video conference calls, e-mails etc. Obviously, this produces a lot of noise in the data being transferred. Sometimes some parts are misheard or misunderstood, get lost and are causes for a larger problem in the future.

Not only there is a difference in the culture and language, but it clearly leads to a difference in the domain vocabularies used in everyday work. Tools for productive data exchange are implemented - such as:

- Issue tracker
- Internal Wiki-style website
- Enterprise chat

But the problem is, that there is no tool that unites them all to make sure that the domain vocabularies are the same throughout the whole development phase. If such tool existed, a one that could connect to all of the deployed knowledge systems, it would make communication of measured performance of a product a lot easier.

Let's take a look at the current tools available on the market that specialize in application performance measuring.

2.4 Existing Tools

I will now analyze and assess the tools available on the market. I'll focus primarily on the flexibility of the deployment and their analysis features. Anything else has a minor priority.

2.4.1 Google Analytics

Google Analytics is by far the most popular and widely used framework for monitoring user interactions in applications. It reports everything the developer wishes to. By default it does not report anything - the tool has to be activated during application start. Then each action needs to be hard-coded in the code. An action can be two things - the first one is after a certain user activity has happened - a press of a button, pulling down a list to refresh the data etc. One thing gets reported - "this event has happened". The second one is more open - an identifier is set to an item, let's say, a button. Whatever happens with that button gets reported, be it touch, swipe or anything else.

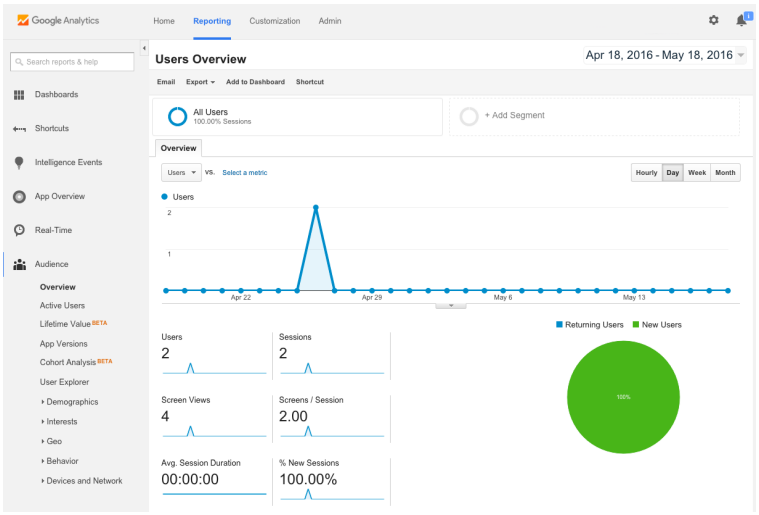


Figure 2.2: Google Analytics Dashboard

The dashboard website is very detailed and responsive. All data is nicely visualized in graphs and corresponds well with the whole GA ecosystem. Higher order semantics is remotely possible via definition of complex queries and dashboard setups. User management is very rich and enables forming multiple roles for different users. The main problem is, unfortunately, that all data goes to Google. There is no option of having the engine run on custom servers. All code is closed source and that simply wouldn't go through any risk assessments because Google might be sending the data to anywhere in the world.

Data is accessible via REST API, but registration for it is required (it is not possible for a "normal" user to start using the REST API). Single user account is free, enterprise account is paid.

2.4.2 Google Tag Manager

Google Tag Manager is a tag management system that allows quick and easy update of tags and code snippets in a mobile application. It allows adding and updating AdWords, Google Analytics, etc. from the Tag Manager user interface instead of editing the code.

A tag is a snippet of code that sends information to Google. It is not necessary to wire it up in code - it works through configuration files from the admin user interface.

Tag Manager is deployed in conjunction with the Firebase SDK, with support for both Android and iOS. The container replaces all other manually-coded tags in the mobile application, including tags from AdWords, Google Analytics etc. It basically builds on top of Google Analytics to make the integration a little easier. It is aimed to be used by people focused mainly on marketing, thus the tool itself serves as an overall performance dashboard. It allows them to create complex tags in a short period of time. Unfortunately it works well only on the web - the tags have to be hard-coded in mobile applications.

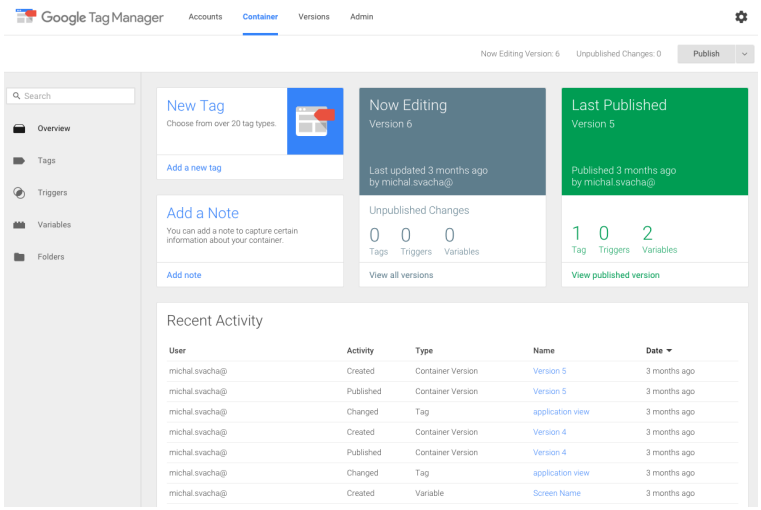


Figure 2.3: Google Tag Manager Dashboard

The dashboard is similar in style to the Google Analytics one. Tags can be created, modified and linked with a specific Google Analytics query. The same as for Google Analytics holds for Google Tag Manager - everything is closed source and runs exclusively on Google's servers.

2.4.3 Tealium

Tealium is a complex tool combining all marketing tools on the market (Criteo, Socialbakers, but also Google AdWords etc.). It does offer insights into usage and performance and it is very robust. Afterall, this is what they claim to be able to do:

"Combining the leading enterprise tag management solution, an omnichannel customer segmentation and action engine, and a suite of rich data services, the Tealium CDP enables organizations to unlock customer data trapped in siloed marketing systems, build a unified customer view, and take real-time action." [8]

As of December 2015, Tealium successfully completed a Health Insurance Portability and Accountability Act (HIPAA) and Health Information Technology for Economic and Clinical Health Act (HITECH) attestation examination [9], making it compliant to store data in the cloud securely. The only limitation with this is, that it is only valid in the USA - it does not extend to countries such as China, Japan or Russia, where customer data have to be stored within the country's borders.

The tool itself is intended to be wired up and used as-is. Tealium has their own servers, their won front-end and no API. While for smaller to mid-sized companies it may be sufficient, larger companies require code customizations and integrations into their systems that are currently not provided by this tool.

2.4.4 Fabric (formerly Crashlytics)

Fabric is a platform made by Twitter. Collected and fine-tuned I should say, as it comprises of multiple acquired start-up platforms. The focal point of this platform is a former crash reporting tool Crashlytics [2]. Aside from that Fabric is a full-featured developer platform used for a variety of tasks. It helps to overcome obstacles a developer may face - even the distribution of beta versions to testers, which has always been a problem for iOS developers:

"The Fabric platform is made of three modular kits that address some of the most common and pervasive challenges that all app developers face: stability, distribution, revenue and identity. It combines the services of Crashlytics, MoPub,Twitter and others to help build more stable apps, generate revenue through the world's largest mobile ad exchange and enable to tap into Twitter's sign-in systems and rich streams of real-time content for greater distribution and simpler identity. Installation takes minutes, and most features only require a few lines of code." [15]

Fabric has been taken it a step further and is not only about reporting crashes, but it also reports overall statistics, like Google. One really nice feature is a video visualization of user movements in the application. Formerly Appsee [1], now a UX part of Fabric, creates a video of steps users take in the application to give the developer a new perspective on how their application is actually used.

The source code of Fabric is closed source (except for some parts, like Fastlane [7]) and all of the statistics run on Twitter's servers. The Data is not accessible in any other way, than via Fabric's custom dashboards.

The whole platform is free to use. Registration of developers and applications is required along with the installation of a custom Fabric installation program which integrates the framework into existing projects.

2.4.5 App Pulse (formerly Pronq)

Hewlett-Packard Enterprise AppPulse Mobile is a mobile app performance monitoring tool that tracks the real user experience of mobile applications.

App Pulse is very different from the previously listed tools in a way that it reports everything at all times. The usage is fairly low - tens of kilobytes per week, but it is very thorough. Screen time, actions, movements - it is all there in the report and no setup is required for the start. That obviously leads to gathering a lot of junk by default.

Contrary to iOS standards, App Pulse's SDK is distributed via a compiled library that has to be drag-and-dropped manually into the Xcode project. This is not a good practice, as Apple's updates are regular and oftentimes break a lot of old code, so keeping the SDK up to date manually is a lot of pain.

Because it works out-of-the-box and "automagically" tracks everything that happen in the application, the one big issue is the need to have a consistent naming of all views, labels, buttons etc. - as it does everything on its own, without hooking up the actionable items to the framework manually, it can be hard to determine which button was referred to in the

report. One really interesting feature is battery level monitoring, though it is disputable, as the user may have multiple applications running in the background.

Free 30-day trial allows up to two mobile applications, 25.000 active users per application (everything above is discarded, or offered for the standard monthly price) and access to the App Pulse community. The tools are closed source and all of the statistics runs on Hewlett-Packard Enterprise servers as it is a Software as a Service (= SaaS). No API is provided.

2.4.6 Apteligent (previously Crittercism)

Apteligent doesn't stand out from previously mentioned tools - it runs on Apteligent's own servers, has custom SDK and works seemlessly. While it does have some benchmarking that others don't, at the end of the day, Apteligent again is the owner of your users' data. It seems very enterprise oriented - it enables 3rd party API integration into their system, which enables monitoring the performance of other APIs used in the application to really find what can be the bottleneck of the overall performance.

The tools are closed source and all of the statistics run on Apteligent's servers. API for data retrieval is provided.

2.4.7 New Relic

Other than the standard package (SDK, custom servers, periodical reports), New Relic has a nice alerting system - when a crash occurs, a web hook to ticketing system can be defined to streamline bug reports into the issue tracker.

Similar to Tealium, New Relic does stand out with the focus on data encryption:

"You have complete control over what, if any, sensitive information is sent to New Relic. We are unique among software analytics solutions. When you deploy our agent, by default, our security settings and regulatory compliance exceed industry standards, and all data is encrypted in transit." [12]

While this is a good start, it simply isn't compliant with the legislation of many countries, to store the users' data on a "random" cloud computer storage. The iOS SDK is distributed in a compiled binary file, having enormous 28,5MB. It is not open sourced. API for data retrieval is provided.

2.4.8 Apple

The last tool I will mention isn't considered as a framework, but is important nonetheless. As companies fight for user data from mobile applications - such as Twitter, who even gives out their platform to everybody for free, naturally the platform owners (Apple and Google) strive for keeping all that precious data for themselves. Apple announced in 2015 at their Wordwide Developer Conference (= WWDC) a new iTunes Connect portal with a new feature - App Analytics.

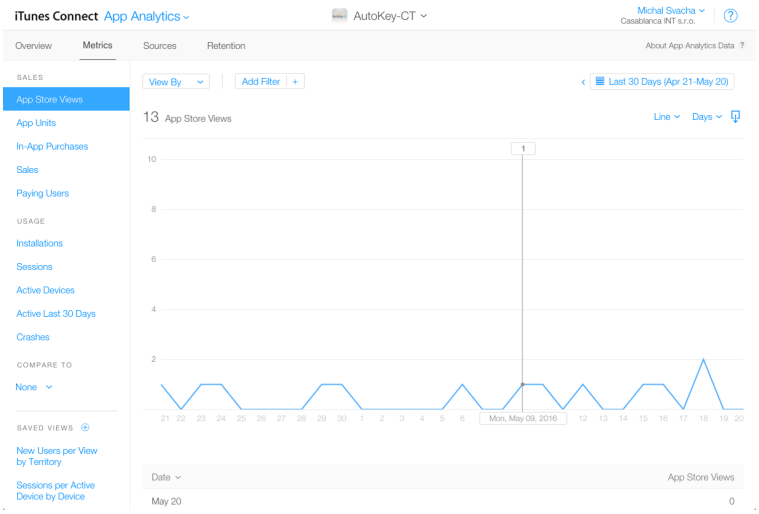


Figure 2.4: Apple App Analytics Dashboard

It is fairly thorough in means of usage, downloads, screen time etc., but the overall reporting seems very marketing oriented. Crashes do get reported, but not they are not very detailed compared to Fabric's Crashlytics tool.

One of the biggest obstacles every iOS developer faces is the fact that since iOS 5 [14] it is not possible to uniquely identify a device. Therefore it is virtually impossible to connect two user entries of the same user if the user deletes and redownloads the application. The only way it could be identified is through Apple's own analytics solution. However, the help tooltip for Installations says:

"The total number of app installations and redownloads. Installations doesn't include app updates."

Which looks like Apple disabled identification of the same device for themselves as well.

App Analytics in iTunes Connect is provided to every single developer automatically for free on the iTunes Connect website. There is no framework to be implemented, the statistics is gathered whether the developer likes it or not. Naturally Apple keeps all of the user data on their servers. Even though iTunes Connect has a JSON REST API, none of the sales or analytics data endpoints are exposed.

2.5 Conclusion on tools

1. Google Analytics

PROS: robustness, reliability, reputation
CONS: completely proprietary, very low-level

2. Google Tag Manager

PROS: abstraction above GA, flexibility
CONS: completely proprietary, hard initial setup

3. Tealium

PROS: versatility, omni-channel connectivity
CONS: completely proprietary

4. Fabric

PROS: versatility, community support
CONS: completely proprietary, required Twitter registration, data inaccessibility

5. App Pulse

PROS: automatic initial setup
CONS: completely proprietary, non-standard SDK distribution, data inaccessibility

6. Apteligent

PROS: 3rd party tool integration
CONS: completely proprietary

7. New Relic

PROS: user data encryption
CONS: completely proprietary, non-standard SDK distribution

8. Apple

PROS: no installation or setup required
CONS: completely proprietary, data inaccessibility

None of the tools meet the requirements for deployment (custom servers) and therefore privacy as well for a global enterprise in regulated market. It is necessary to design an architecture that would fit the needs - run the whole solution on custom servers and enable the mash-up of various knowledge resources deployed across the company.

Chapter 3

Design

3.1 System Architecture

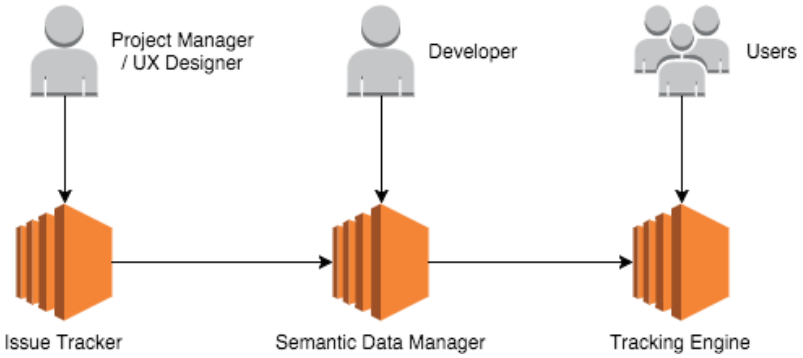


Figure 3.1: Proposed Workflow

Description

The workflow should be as linear as possible with respect to the natural flow of data. The central knowledge base is the issue tracker, because that's where the project management team defines objectives set by the top management. The UX team has specific needs for measuring certain tasks in the application so it is absolutely valid that they would add their needs in the issue tracker.

From then on, the domain vocabulary is created and stored for everybody to see. In order to measure the tasks performed by the developer (for example "implement login form") using the same domain vocabulary, something has to deliver a computer-readable version of the vocabulary. That is the task of the Semantic Data Manager - to extract the knowledge from the Issue Tracker and serve it to the developer so he/she can use it easily within the application.

The user data is then reported to the Tracking Engine using the same domain vocabulary that was defined at the beginning. That will enable alignment and easy reporting of the results.

3.2 Issue Tracker

Issue Tracker is a software deployed in a company for managing and tracking the work done by multiple employees across different departments. It serves the project management team to define projects, milestones and to break the work down to tasks and sub-tasks. Those are then assigned to specific people and marked as done when they're done and reviewed. It is considered a good practice that when a task/issue is created and assigned, it becomes immutable with respect to its name (and of course project label and ID). The detailed description and comments can be added, but in order to preserve some stability, the name should remain the same.

There are many solutions on the market, such as Bugzilla, JIRA or open-source Redmine. Most of the solutions provide a REST API for 3rd party integrations.

3.2.1 Knowledge structure

Every company uses a slightly different way of structuring their knowledge in their project management tools. In my situation, the structure is as follows:

- Teams
- Project
- Issues

Different tools have different naming conventions, but this structure is as generalized as possible so it is applicable to most of the tools on the market.

3.2.1.1 Teams

Everybody is distributed into teams in various departments. While departments are important in the company structure, in the project management, teams are more relevant. Every team has its own Project Board (either Scrum or Kanban) that reflects the state of the team in time - how much work has been done, what will be done and what's currently being done.

3.2.1.2 Projects

Each team works on their own projects. Often times, collaboration does occur, but the owner of the project is still the team. If other person collaborates with other team, it is reflected in his/her work statistics (storypoints, hours spent), but the ownership stays within the scope of the project and thus the team as well.

3.2.1.3 Issues

Issue is the most granular entity in the system. It defines a step or set of steps to be performed and is usually assigned to one person. Workflows for reviewing etc. vary from team to team. There is a possibility to break down the work on single issue (if it's especially large one) to sub-tasks, but those are not given IDs, so they are not unique per se. They only serve for clarity and visibility of work being done during the day. When all sub-tasks are finished, the whole issue is finished. That is the state reflected by the API.

3.2.2 Knowledge Extraction

The extraction of the knowledge will build on top of the premise that some parts of the data at certain point of time are immutable. In order to make it as bulletproof as possible, I will only assume immutability of the name of the issue, its automatically assigned identifier (ID) and, obviously, the project it belongs to. Everything else is considered fully mutable and is welcome to be changed in order to provide flexibility.

The idea behind the tracking itself in the same domain vocabulary is that the beginning of an issue/task/sub-task and the end is reported. Giving a direct feedback of success or failure of the implemented workflow.

Example

An issue is created in the Issue Tracker - "Create form for user registration.". The Issue Tracker automatically assigns an identifier, for example "SA-01". The resulting database table should look like this:

Issue ID	**Name**	**Type**	**...**
SA-01	Form for user registration.	Start	...
SA-01	Form for user registration.	Finish	...
SA-01	Form for user registration.	Start	...
SA-01	Form for user registration.	Finish	...
SA-01	Form for user registration.	Start	...

Table 3.1: Example Database Entries

It is clear what is measured here[1] - the success of a registration of a new user. For each commenced registration, there is a "Start" entry. For each successfully finished registration, there is a "Finish" entry. There was not defined any other KPI to measure, so without storing any junk data, it is easy to calculate the percentage (2/3) of users that successfully registered versus that didn't (1/3).

[1]The "..." marks obvious missing columns, such as timestamp, device type etc.

3.2.2.1 DSL

In order to track more than just the beginning and the end of the workflow defined in the scope of the issue, it is necessary to give the stakeholders a way to define certain observable events to pay attention to. Usually, every issue has a field specifically for description, that contains some human-readable set of instructions. Why not piggyback on that and give it just enough structure to make it also computer-readable?

Let's assume it would be desired, to measure how many times the user clicks on the help button in the registration form. Then the database table would be extended to:

Issue ID	**Name**	**Type**	**...**
SA-01	Form for user registration.	Start	...
SA-01	Form for user registration.	Finish	...
SA-01	Help button pressed.	Action	...

Table 3.2: Extended Example Database Entries

Another type has appeared in the database table - "Action". Some measurable item, linked to the registration form itself. Neatly labeled with the relevant Issue ID, easy to find and in the same domain vocabulary as was intended.

3.3 Semantic Data Manager

This is the most crucial component of the whole solution - connecting the Issue Tracker and tracking capabilities. The main task is to obtain actionable items from the Issue Tracker and create computer-readable domain vocabularies for application engineers to include in their source code. The computer-readable format should in the for of a configuration file. Therefore, there must be a framework that parses such configuration file, and offers in-code assistance with implementing the tracking itself. It goes without saying that the usage should be as smooth as possible.

The produced configurations should be persisted for the sake of reproducibility. Though it should avoid storing any metadata (detailed measurement description) as it is subject to change (DSL used in the description). Only the unique issue identifier and its name should be persisted, in order to be able to trace back what was the configuration file meant for. Persisting anything else could lead to disconnection between data and that's exactly what I am trying to solve.

Connecting to the Issue Tracker will be handled via its REST API. It should automatically retrieve a list of all projects and their subsequent issues/tasks/sub-tasks that may or may not contain specific metadata regarding fine-grained measurement demands.

Semantic Data Manager will run as a stand-alone microservice and will provide a REST API for any kind of UI to be implemented on top of it. It can easily be a website, mobile application or integration into a 3rd party tool.

3.4 Tracking Engine

The Tracking Engine itself is a completely separate component from the Semantic Data Manager. It can easily be any existing solution - Google Analytics or Google Tag Manager. It should be a stand-alone service into which data is stored using the already obtained domain vocabularies.

For reasons listed in previous chapter, mainly in-house data storage, I will also develop this component and deploy it alongside the Semantic Data Manager. This gives me more flexibility in optimizing the storage and provides the company with a custom tailored solution.

3.5 Tracked Device SDK

The SDK has to have two layers:

1. Interpreter of the configuration file from the Semantic Data Manager - load the vocabularies and provide them during the development.
2. Interface for storing the events - should the Tracking Engine change, the only thing that would be necessary in the application code would be to download dependencies for it and implement the interface. This approach ensures that the in-code measuring always remains the same, only the underlying engine changes.

The whole company runs on iOS devices, so I will develop an iOS framework to be used while developing applications. While iOS may seem limited to only iPhones and iPads, the ecosystem actually enables the frameworks to be written for all Apple platforms - iOS, tvOS and Mac OS X.

Chapter 4

Implementation

4.1 Deployment

The whole solution is designed for and deployed in the Amazon Web Services (= AWS) cloud environment. It is a managed Virtual Private Cloud (= VPC) environment that is only accessible with company credentials and certificates.

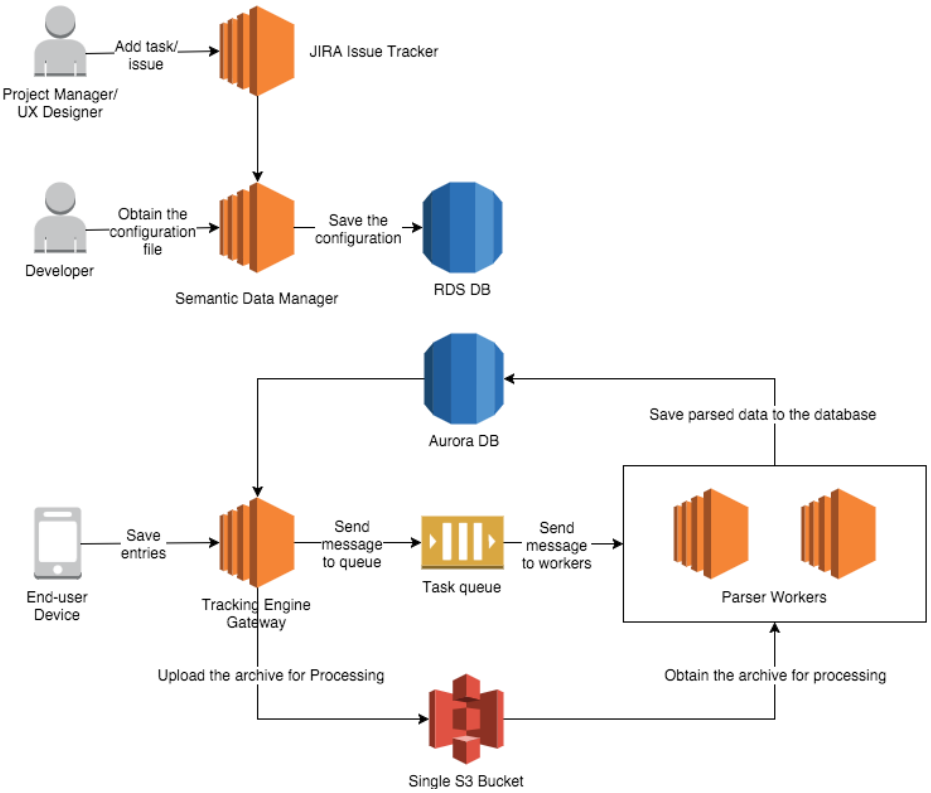


Figure 4.1: AWS System Architecture

4.2 Issue Tracker

The Issue Tracker used in my work environment is JIRA[1] by Atlassian. It is a standard project management tool providing bug tracking, issue tracking and many other functions. It is conveniently synergistic with other Atlassian tools such as Confluence (for documentation and wiki) and Bitbucket (formerly Stash), which is a server for version control (Mercurial and Git). The advantage is that all these three components are deployed in the AWS VPC environment. Thus, as a programmer I have fairly easy access to its internal APIs without being afraid to leak data where it shouldn't.

4.2.1 API

JIRA REST API is quite a mess, which is because Atlassian didn't develop all their products from scratch (Bitbucket was acquired) and it is still visible that the usage isn't seamless. In order to access Teams, Projects and Issues, two API endpoints have to be used:

1. JIRA REST API
2. JIRA REST AGILE

Both of them are APIs (= Application Programming Interface), but I will use their names to distinguish one from another.Both are similar, but also slightly different from each other.

To illustrate the subtle differences that drive any software engineer mad:

- When JIRA REST API is used to obtain the issues, the resulting array uses pagination, because there could be a lot of issues and loading them all at once could take a significant amount of time. In order to determine whether the array I have is final, a parameter "total" is present in the response. This parameter tells how many issues in total there are. In order to load the whole list, it is necessary to keep track how many there are, and how many are left on the stack.
- When JIRA REST AGILE is used to obtain the teams, the resulting array also uses pagination. In order to determine whether the array I have is final, a parameter called "isLast" is present in the response, having, surprisingly, a boolean value true/false. Obviously, when the value is false, one has to load the next page with the last index that came before.

There are plenty of these little surprises that are so easily breakable with any update of the whole system. I honestly do not know, why it isn't the top priority for Atlassian to unite their APIs.

What struck me most though, is the absence of OAuth or Token-based communication. Every query is done via basic auth. While for development it is fine as it allowed me to quickly prototype on top of the API without the need to develop a complex token manager, for production it is quite inconvenient. Even though SSL certificates are all valid and in place, it simply is a terrible architectural choice to not have a proper way to authenticate other applications using the APIs.

[1] No abbreviation here. It is short for GOJIRA, which is Godzilla in Japanese. Rumor has it that it is because the main competing product is Bugzilla.

4.2.1.1 JQL

JQL stands for JIRA Query Language [13]. It enables the API user to query the JIRA knowledge graph and extract information.

Figure 4.2: JQL syntax (source [13])

1. **Field** - Fields are different types of information in the system. JIRA fields include priority, fixVersion, issue type, etc.
2. **Operator** - Operators are the heart of the query. They relate the field to the value. Common operators include equals (=), not equals (!=), less than (<), etc.
3. **Value** – Values are the actual data in the query. They are usually the item for which we are looking.
4. **Keyword** – Keywords are specific words in the language that have special meaning. In this post we will be focused on AND and OR.

I used JQL in order to get all issues for a certain project:

```
"https://jiraURL/issues/search?jql=project=SAUI"
```

Here I used simple query to search all issues where the project is SAUI (= Semantic Analysis of User Interactions). All URL encoders handle the double "= =" and it has never happened to me, that it would encode the parameters badly.

It can obviously be even more powerful, but I was glad it helped me easily get what I needed.

4.2.1.2 Methods used

All communication is handled via HTTP GET and all responses are in JSON format. Cross-site request forgery (= CSRF/XSRF) token system is disabled.

1. To get all teams, method /**board** has to be called on JIRA REST AGILE.
2. To get all projects, method /**projects** has to be called on JIRA REST API.
3. To get all issues, method /**search** with JQL query has to be called on JIRA REST API.

Interestingly enough, even though, there are two API endpoints, the data is connected, so no further processing was necessary. It is important to note, that the responses are **very** verbose and it is possible to tell in the query to the server not to send some fields back.

4.2.2 DSL

After a discussion with a UX lead, I came up with a simple solution - add keyword "WATCH:" on a new line and describe what to observe. Parsing is done line by line where the code searches the line for "WATCH:" (case insensitive) and extracts whatever follows until the end of line or occurrence of another "WATCH:". In order not to make it complex, end of line is the end of any description, it does not carry over to the next line.

4.2.2.1 Examples

Here are some examples how the parser for the DSL works. Validation of this technique will be covered in the Testing chapter.

Example 1 - Success

```
As a user, I want to be able to list all projects in the mobile app currently
    being tracked along with the number of tracked versions in the tracking system.

WATCH: Number projects expanded to the highest detail
WATCH: Filters used to extract information
```

This succeeds perfectly, as it parses everything without any hassle.

Example 2 - Success

```
As a user, I want to be able to list all projects in the mobile app currently
    being tracked along with the number of tracked versions in the tracking system.
Watch:    Number projects expanded to the highest detail
Watch:    Filters used to extract information
```

This succeeds too, because the search is case insensitive and after the search, white spaces are extracted, so the result is the same as in the previous example.

Example 3 - Semi-success

```
As a user, I want to be able to list all projects in the mobile app currently
    being tracked along with the number of tracked versions in the tracking system.

WATCH: Number projects expanded to the highest detail, Filters used to extract
    information
```

This is a semi-success, almost a failure, but it still yields all the information that the user wanted. It just isn't nicely separated and would need some changes. The error is visible to the programmer and is easy to fix.

Example 4 - Failure

```
As a user, I want to be able to list all projects in the mobile app currently
    being tracked along with the number of tracked versions in the tracking
    system. WATCH: Number projects expanded to the highest detail. WATCH: Filters
    used to extract information
```

This yields only one result - "Number projects expanded to the highest detail.". While it seems like it is a good solution, the fact that it seems that way is unfortunately the worst thing about it - because it is hard to discover that there is an error. The programmer sees one observable action item and doesn't see that some got lost during the process, because it was all on one line.

4.3 Semantic Data Manager

The central part of the project should be robust and reliable. For that reason I chose Java as the main technology. For convenience and standardization of the code-base I opted for Spring Boot framework to help me with bootstrapping the heavy work (scheduling, threading, persistence etc.).

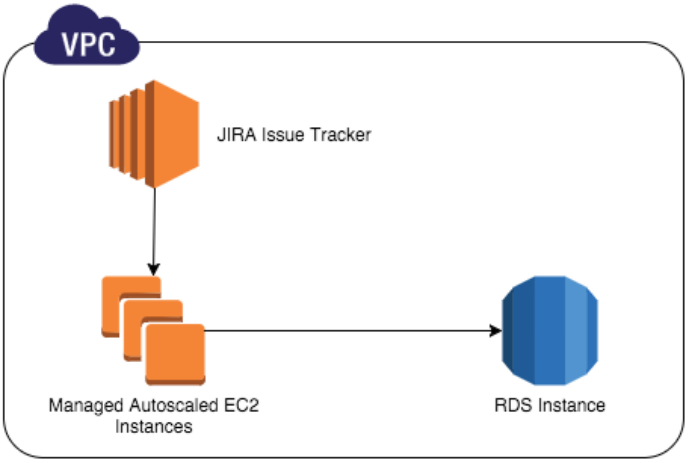


Figure 4.3: Semantic Data Manager Deployment

Semantic Data Manager is deployed in AWS Elastic Beanstalk (= EB), which is Amazon's Platform as a Service (= PaaS) for Java web applications. It reduces management complexity without restricting choice or control. All it takes is to upload the application, and Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.

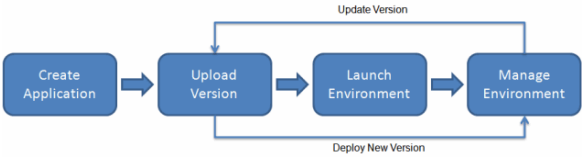


Figure 4.4: AWS Elastic Beanstalk Application Deployment (source [4])

It is important that the EB application runs in the same availability zone of the VPC as JIRA, otherwise it wouldn't be reachable at all.

The database runs on managed Amazon RDS (Relational Database Service), which is fast, secure and scalable deployment of database engine. It manages backups, software patching, automatic failure detection, and recovery. The default database engine is MySQL.

4.3.1 Spring Boot

I first tried to use play2 framework for educational purposes, but I encountered too many obstacles deploying play2 application to AWS:

- It does not support WAR packaging.[2]
- It is not possible to run play2 packages (packaged by Activator tool) on Tomcat Server.
- It comes with its own Netty Server, which is really clumsy to set up in AWS environment.

All three combined resulted in inability to synchronize the play2 application on port 9000 and NGINX running on port 5000. Unfortunately Netty Server does not support compile-time port configuration and NGINX does not support running Activator to set up the port during run-time, so I had to drop the idea of using play2 as I was simply unable to deploy the application. After researching and discussing with my peers and coworkers I looked up Spring MVC and stumbled upon Spring Boot, also recommended by my classmate. I tried few sample apps and found out it supports WAR packaging, runs natively on Tomcat and comes with almost the same perks like play2. I was ready to give it a try.

Primary goals of Spring Boot are:

"Spring Boot aims to make it easy to create Spring-powered, production-grade applications and services with minimum fuss. It takes an opinionated view of the Spring platform so that new and existing users can quickly get to the bits they need." [16]

- Provide a radically faster and widely accessible getting started experience for all Spring development.
- Be opinionated out of the box, but get out of the way quickly as requirements start to diverge from the defaults.

[2] There is an unofficial tool that packages the code in a WAR file, but it is not recommended for production environment. Being constrained by highly regulated market, something that already says that it is not production ready is an instant "No thanks".

- Provide a range of non-functional features that are common to large classes of projects (e.g. embedded servers, security, metrics, health checks, externalized configuration).
- Absolutely no code generation and no requirement for XML configuration.

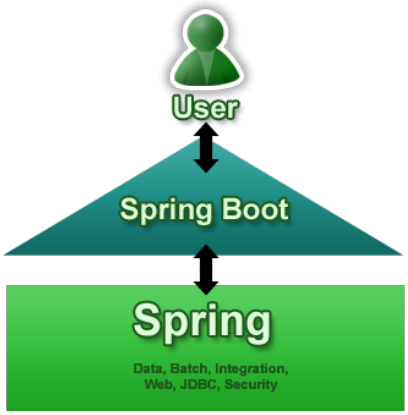


Figure 4.5: Spring Boot architecture (source [5])

The advantages seemed to be strong, development environment was convenient (native support in IntelliJ IDEA) and after some validation with Amazon support regarding AWS EB deployment in our VPC environment, I was confident this would be a good choice.

4.3.2 Workflow

The server makes the workflow very easy and exposes a REST API to obtain the information. Basic workflow is as follows:

1. List available projects.
2. List available issues in a project, along with parsed watched values.
3. Prepare a configuration XML file (streamed as UTF8 encoded string) with selected issues.

No registration, no other setup is necessary - it runs in the VPC environment, so only the employees can access it with their approved devices with the network certificates. Also, JIRA is open for everybody to read, so why bother with extra signing in/out and session management, if it's actually not desired. The component works as-is and serves only one purpose - to connect JIRA and the application code as easily as possible.

4.3.3 Technologies Used

In order to make the development as comfortable as possible, I used various open-source frameworks/technologies to help me out with things that would otherwise take a lot of time.

4.3.3.1 H2 Database

H2 is a Java SQL database that I used in in-memory mode during the development (before switching to RDS). It is great for fast prototyping and validating of design ideas before using production database (RDS). Due to the fact that the entire database is deleted after a server is restarted, it enabled me to quickly change the schema without the need to do tedious database wipes.

Another great thing is that it only needs to be defined in pom.xml and all other linking is provided out-of-the-box:

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

Important note: as it wires up all connections automatically, it is absolutely vital not to have **any other** database engine loaded via Maven, otherwise it crashes on start-up of Tomcat.

4.3.3.2 Flyway

Flyway is an open-source database migration tool. It aims to be clean and simple rather than robust and too complex. It uses 6 main commands:

1. Migrate - Migrates the schema to the latest version.
2. Clean - Drops all objects in the configured schemas.
3. Info - Prints the details and status information about all the migrations.
4. Validate - Validates the applied migrations against the available ones.
5. Baseline - Baselines an existing database, excluding all migrations upto and including baselineVersion.
6. Repair - Repairs the metadata table.

Since I didn't need to migrate the database, thanks to the flexibility of H2, I mainly used Flyway for cold start (empty database). I defined, what will be in the database after the server has started and Flyway filled it up for me.

4.3.4 Code Deployment

The code is packaged by Maven and deployed as a WAR file to AWS EB instance, running Tomcat 8 server. There is no need to configure anything with regards to basic networking - AWS EB is a back-to-back fully managed Platform as a Service (PaaS). Once deployed to VPC, the only thing that needs to be addressed is the availibility zone - making sure, it runs in the same zone as JIRA so they can communicate. It can be set up very quickly in the configuration section, however it can bring some frustration in the beginning when the developer doesn't know about it.

There are multiple things to consider when deploying to AWS EB, even though it seems "super-easy" in most instructional videos and ads:

1. Contrary to programmer's logic, one has to first create an "Application" and under that custom "Environments". In other words, Application is the main context, and environments are servers running in the same context, by default having the permission to communicate between each other.

2. In Environment setup, we can choose either a Web Server Environment or a Worker Environment. Web Server is the hub of any application and is used in both Semantic Data Manager and Tracking Engine. Workers are only used in Tracking Engine and will be explained later.

3. Chosen configuration was obviously Tomcat and I also opted for automatic load balancing and scaling.

4. Opting to automatically create an RDS instance along with the environment is a really bad design. Once an environment is terminated, so is the database instance which causes a serious data loss.

5. In order to access the servers via SSH, it is necessary to define a EC2 Security Group and assign it accordingly. Otherwise, all outside access is prohibited.

6. It is also crucial to wisely choose the instance type. I opted for m3.medium, because Java applciations by themselves are quite demanding and I didn't want to risk being on the edge when strange errors occur because of insufficient memory capacity.

7. Permissions and roles are absolutely crucial when it is desired to connect the Environment with other AWS services, such as object storage (S3 = Simple Storage Service) or a messaging queue (SQS = Simple Queue Service).

The size of the instance is recommended for any application running JVM. The configuration is Intel Xeon E5-2670 v2 (Ivy Bridge), 4GB SSD storage and 3.75GB RAM. Any additional memory is handled via S3.

4.3.5 User Interface

As mentioned before, Semantic Data Manager provides a REST API to be consumed by any kind of client capable of HTTP requests. Because my specialization are mobile applications, I chose to implement a mobile application for iOS as an administrating user interface for this component.

4.3.5.1 Application Flow

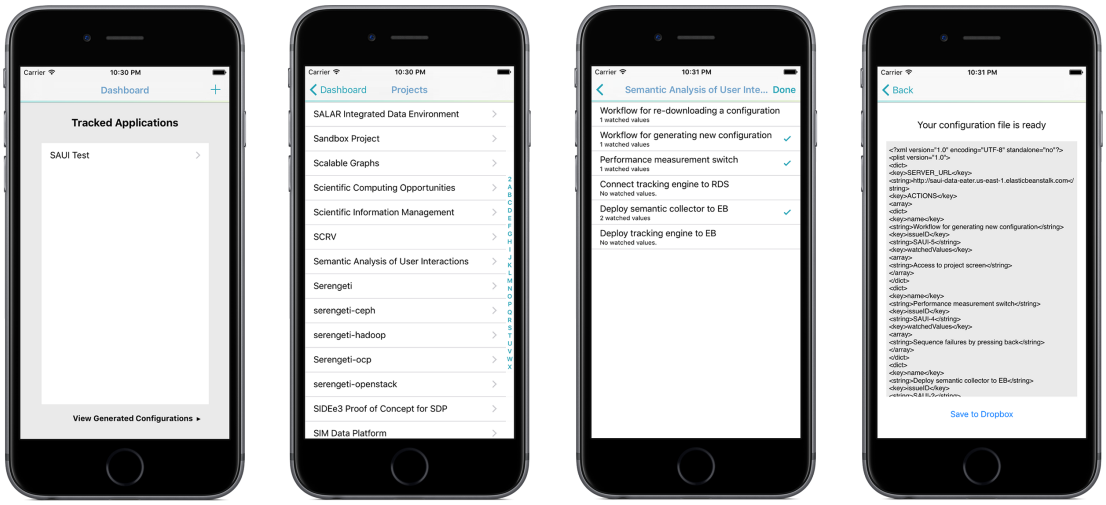


Figure 4.6: Application Flow for Adding Configuration

Creating a new configuration

- First screen presents the user with currently tracked applications and two actionable items - add button and previously generated configurations. Add button leads to next step in order to generate a new configuration.
- Second screen has all available projects in a list, scrollable with alphabet index on the side. Naturally, selecting one, leads to the next step.
- The third screen shows all available issues along with their watched values. User is free to choose which one should be in the configuration (multiple-choice selection). When ready, the button "Done" proceeds with the process.
- The last screen shows the final configuration file that has been saved on the device. In order to send it to a computer, Dropbox integration has been implemented. When teams cooperate, it allows the user to automatically deliver the configuration file to all members of the team.

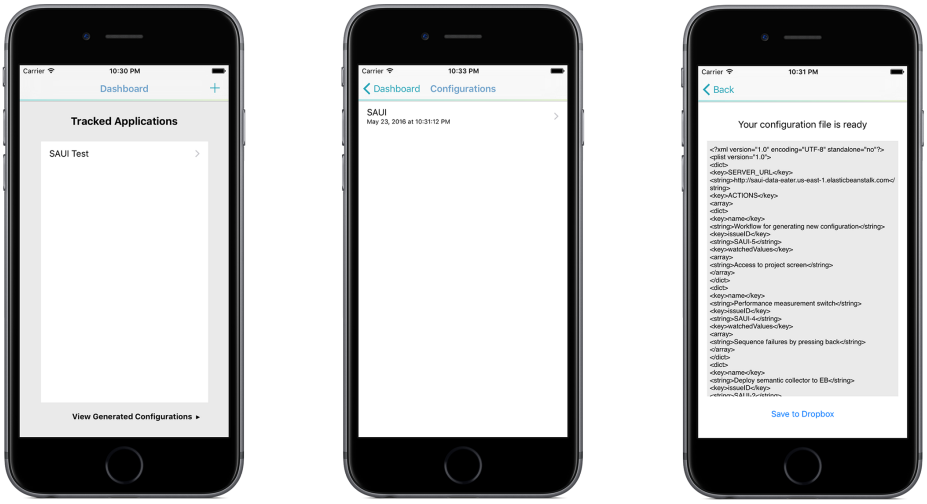


Figure 4.7: Application Flow for Recreating a Configuration

Recreating a previously generated configuration

- Instead of using the "Add" button the user proceeds with the "View Generated Configurations".
- Second screen shows previously generated configurations - those are immutable, because they are snapshots in time. They can only be re-downloaded again. If metadata has changed on JIRA, it is automatically updated.
- As the issue list is immutable in previously generated configurations, user is automatically redirected to the last screen with the generated configuration file.

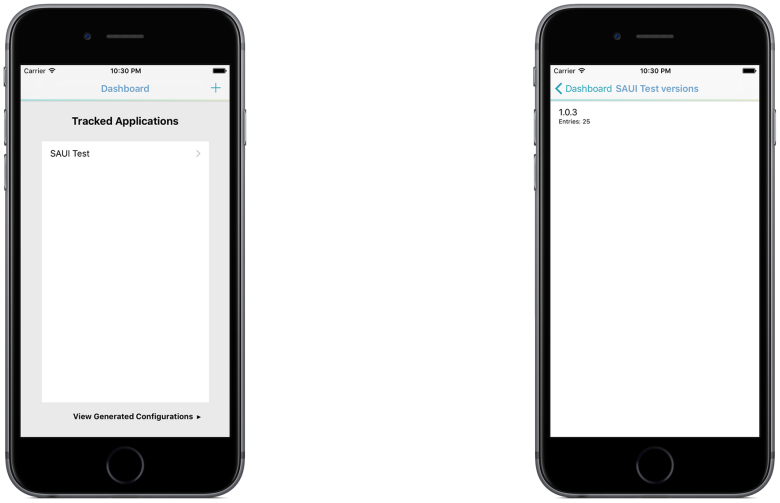


Figure 4.8: Application Flow for a Quick Overview of Gathered Data

Quick overview of gathered data

- Tapping on an application in the list of Tracked Applications leads to a quick overview of gathered data.
- The quick list of overview data is a simple list of the applications versions and number of entries for each version. This application doesn't serve as a statistical front-end, this screen only serves for quick overview for the user to know what's the current configuration status in the applications.

4.3.5.2 Technologies Used

Moya
Moya[3] is an abstraction layer above an open-source networking library Alamofire[4]. It not only forces a cleaner code structure, but also makes it very well testable as it is a single gateway to the internet.

The main advantages are:

1. Compile-time checking for correct API endpoint accesses.
2. Ability to define a clear usage of different endpoints with associated enum values.

SwiftyDropbox
SwiftyDropbox[5] is an official framework for iOS to integrate Dropbox functionality directly in the application. In order to use it, it is necessary to register the application in the Dropbox Developer Console. Personal usage requires no review by the Dropbox team, fully integrated one does. For the purposes of this thesis, the personal was sufficient.

Both Moya and SwiftyDropbox are distributed via standard dependency management tool Cocoapods, which is a similar dependency tool like Maven or Gradle for Java. Written in Ruby, it is less verbose and therefore the dependency management file is fairly short (MBProgressHUD[6] is a "loading" dialog when network operation is in process):

```
platform :ios, '9.0'
use_frameworks!

pod 'Moya'
pod 'SwiftyDropbox', '~> 3.0.0'
pod 'MBProgressHUD', '~> 0.9.1'
```

[3] https://github.com/Moya/Moya
[4] https://github.com/Alamofire/Alamofire
[5] https://github.com/dropbox/SwiftyDropbox
[6] https://github.com/jdg/MBProgressHUD

Implementation Speciality

Swift is a multi-paradigm language and allows the combination of object-oriented programming approach and functional programming approach. Because it was released in 2014 and doesn't have any legacy code carrying along, it has these features all built in. Along with proper generics and type safety rules.

In one of the views (multiple-choice selection of issues), I had an interesting problem that I solved with the combination of the two mentioned paradigms. The situation was, that I had a list of immutable Issue objects representing the data displayed in the list. In order to keep track of which cells have been selected, I had to create another separate array containing boolean values, representing selected/unselected. This is important, ebcause the list view is dynamically generated and not keeping the record somewhere results in complete loss of selection on scroll, as the cells are generated as the user scrolls down/up. Unfortunately this results in having to perform manual filtering through the issues after the user is done selecting them. This is where the multi-paradigm comes in:

```
// declaration of properties - when issues are set after downloading,
    selectedIndicies is populated with false.

var selectedIndicies: [Bool] = []
var issues: [Dictionary<String, AnyObject>] = [] {
    didSet {
        selectedIndicies = [Bool](count:issues.count, repeatedValue: false)
    }
}

// filtering

let result = zip(self.issues, self.selectedIndicies).filter{$0.1}.map{$0.0}

// which the same as its longer version

let result2 = zip(self.issues, self.selectedIndicies)
   .filter{ (sequence: (object: Dictionary<String, AnyObject>, isSelected: Bool))
       -> Bool in
      return object.isSelected
   }.map{ (filteredSequence: (object: String, isSelected: Bool)) -> String in
      return filteredSequence.object
   }
```

This feature of the language is great, because it keeps the code clear and structured. There is absolutely no need for long and verbose code structures, using for cycles and being afraid that it might exceed the maximum index of an array. Also, the global functions on collections are better optimized by the compiler, resulting in 100% speed increase of collection operations.[7]

[7] http://stackoverflow.com/questions/29301577/performance-issue-while-finding-min-and-max-with-functional-approach/29305300#29305300

4.4 Tracking Engine

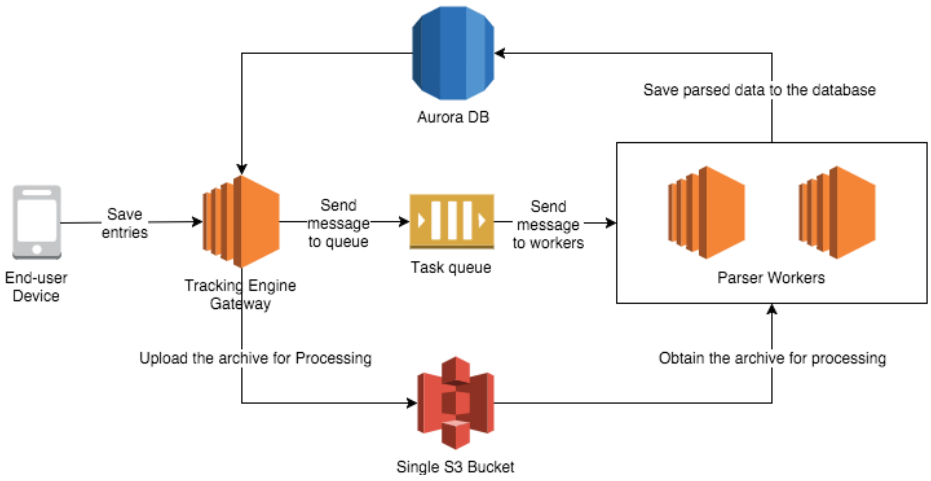


Figure 4.9: Tracking Engine Diagram

Tracking Engine is also deployed in AWS EB but it its workflow is a bit more complex. The whole system is designed to be horizontally scalable on every component:

1. **Gateway** is scaled automatically by the Elastic Beanstalk environment.
2. **S3 and SQS** are scaled automatically by AWS.
3. **Aurora DB** is scaled automatically by AWS, along with replication and availability.
4. **Workers** are scaled automatically instance to instance (vertical scaling), or can be deployed multiple times as is necessary (horizontal scaling - in the console, simply choose "Clone Environment"). Sometimes it is cheaper to deploy more workers on less powerful instances instead of letting it scale automatically with just few.

4.4.1 Workflow

The gateway is written in Java, also using the Spring-Boot framework. It's the main intersection for all the requests coming in an out. The workers are also another Spring-Boot application (deployed multiple times). Both of the applications use AWS SDK for connecting to S3 and the Aurora Database. The queue is implemented automatically in Amazon's environment and the only thing that a developer has to define is the method called on the worker (using HTTP POST). If the worker returns "200 OK", the message is considered as processed, so it is crucial to really throw an exception if one occurs, otherwise the message is lost.

The general workflow is:

1. When a client application sends in the stats of usage, it is designed to save as much data as possible - it is sent archived using GZIP[8]. Because the data is serialized into a JSON file, GZIP performs really well and saves up to 70% of data otherwise spent on sending the giant list of events.

2. The gateway server takes the archive as it is and saves it to S3 object storage.

3. After uploading the archive, the gateway server sends a message into the task queue SQS. The message is absolutely straight-forward containing only the name of the file. The name of the file is generated by the gateway server using the UUID library. If that's not enough, S3 makes sure there is never a collision in the names - if there is, it proposes a different name and returns it after the upload, so it's always unique.

4. SQS then tries to deliver the message to an available worker. If it cannot be delivered it ends up "in flight", which is Amazon's implementation of dead letter queue. From there the Task queue system tries to deliver with a lowering frequency.

5. When the message is delivered to the worker, it obtains the archive from S3, unzips it, parses and saves to the database. When done, it also removes the archive from S3. Then it confirms with "200 OK" marking that the message has been successfully processed.

4.4.2 Aurora Database

Amazon Aurora is a fully managed, MySQL-compatible, relational database engine that combines the speed and reliability of high-end commercial databases with the simplicity and cost-effectiveness of open-source databases. It delivers up to five times the performance of MySQL.

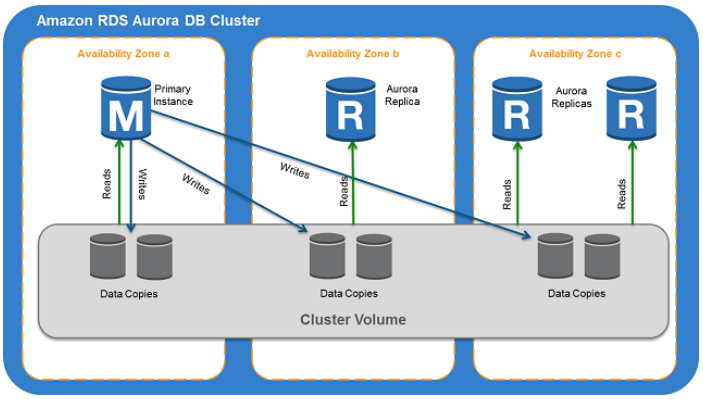


Figure 4.10: Aurora Replication Flow (source [3])

[8] http://www.gzip.org/

Every application operating with large volumes of data eventually one day runs into the wall having a bottleneck in the database. I decided not to compromise when I was choosing between the database engines. There was also a simple RDS instance available, just like the one I am using in the Semantic Data Manager, but this one will have exponentially more entries in the database then the Semantic Data Manager ever will.

Interestingly enough, Aurora DB wasn't officially supported by in the jdbc library enabling applications to connect to it. I managed to find it was in the Release Candidate 2 of the version 1.1.0 of the library.

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-aws-jdbc</artifactId>
   <version>1.1.0.RC2</version>
</dependency>
```

The release of the official update is scheduled to June 2016.

4.4.3 Deployment

The networking capabilities of AWS are very broad and because the architecture of the Tracking Engine is quite complex, it is important to first introduce couple entities in the AWS cloud environment.

Identity and Access Management

Identity and Access Management (= IAM) is a web service that allows the management of access to compute, storage, database and application services in the AWS cloud. It uses concepts known in other similar services in different environments - Users, Groups and Permissions. It is very detailed and allows specifications of which users have access to certain services, the kinds of actions they can perform and which resources are available (ranging from virtual machines, database instances and even the ability to filter database query results). Good thing is, that it works seamlessly with already existing identity management solutions, such as Active Directory[9]

Permissions

Permissions allow specifications of who has access to AWS resources, and what actions can be performed on those resources. Every IAM user starts with no permissions at all. In other words, by default, users can do nothing, not even view their own access keys. Permissions can be added to the user (attach a policy to the user) or the user can be assigned to a group that has the desired permissions.

[9] https://technet.microsoft.com/en-us/library/dd448614.aspx

Permissions can be assigned either as identity-based or as resource-based:

- Identity-based, or IAM permissions are attached to an IAM user, group, or role and allow specification of what that user, group, or role can do.
- Resource-based permissions are attached to a resource. Resource-based permissions allow specification of who has access to the resource (S3, SQS) and what actions they can perform on it. Resource-based policies are inline only, not managed.

Policies

To assign permissions to a user, group, role, or resource, a policy has to be created. Policy is a document that explicitly lists permissions.

Policy allows specification of the following:

- Actions: which actions should be allowed. Every service has its own set of actions.
- Resources: which resources should be allowed to perform actions on.
- Effect: which effect will a request to access by a user - deny or allow.

All policies are defined by a JSON file and can be assigned in totally granular fashion to every service and user.

Security Group

A security group acts as a virtual firewall that controls the traffic for one or more instances. When an instance is launched, it can be associated with one or more security groups. Each group has rules that can be added that allow traffic to or from its associated instances. All rules for a security group can be modified at any time. The new rules are automatically applied to all instances that are associated with the security group. When a decision has to be made whether to allow traffic to reach an instance, all the rules from all the security groups associated with the instance are evaluated.

The whole AWS cloud environment is very complex and has way more to offer than just the couple points I listed above. For the sake of deploying my architecture it is sufficient. For anything else, AWS documentation[10] is of very high quality and virtually any question can be answered there.

[10] http://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

Having the knowledge of all necessary foundations of the inner workflow in AWS, all of these prerequisites have to be met in order to deploy the Tracking Engine:

1. AWS EB application with IAM role with permissions to access SQS and S3.
2. Policy on S3 for the IAM role set up to be able to read/write via JSON configuration:

```
"Statement": [
     {
        "Sid": "sid123456789",
        "Effect": "Allow",
        "Principal": "*",
        "Action": "s3:*",
        "Resource": [
           "arn:aws:s3:::iam-role-name/*",
           "arn:aws:s3:::iam-role-name"
        ]
     }
  ]
```

3. Aurora Database instance up and running with the same Security Group as the AWS EB application.
4. Open 3306 port in the Security Group between the AWS EB application and the Aurora Database.

Everything else is automatic when the AWS EB application is set up as described with the Semantic Data Manager. Database URL is for security reasons injected through environment variables (it should never be hard-coded!).

4.4.4 Timeseries Application

The administration application for the Semantic Data Manager does provide a quick overview, but that is not meant to serve as a dashboard or any decision-making helper tool. It's merely just a peak on whether or not the desired application version is being tracked. Clearly that is not optimal for deriving any conclusions.

In order to show the power of the gathered data I developed another mobile application meant just for displaying the flow of the data in time. It is inspired by the popular application Numerous[11], available on the Apple App Store.

The requirements for such an applications could be endless - graph mash-ups, benchmarking and so much more. In order to simply demonstrate the movement in the KPIs and the potential in the data, I chose to display the total increase/decrease of entries during the last 5 days.

[11] http://numerousapp.com/

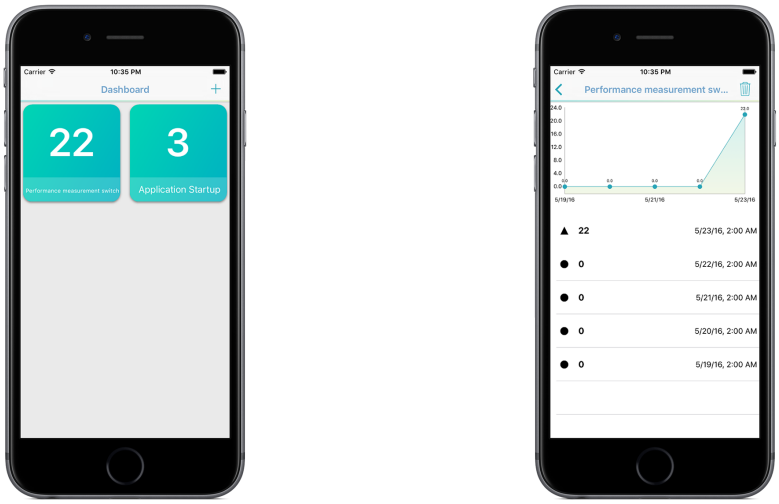


Figure 4.11: Time Series Detail

1. After choosing an application through the "Add" button (the list looks identical to any previous lists), the application automatically loads all KPIs available on the server. The number marks the total number of entries in the last 5 days.

2. By tapping on a number in the grid, a detailed graph is displayed having total number of entries per day - summing up to the same number as is displayed on the dashboard. The KPI can be deleted from the application so it's not displayed anymore (it remains on the server), because the application persists what was selected in the choice of applications and refreshes it on the next start of the application.

4.4.4.1 Technologies Used

Aside from the technologies already listed in the administration application (which I have used again) - Moya and MBProgressHUD, I also used:

ios-charts

ios-charts is a charting library based on MPAndroidChart[12]. It is written in Swift and supports plethora of chart types - line charts, bar charts, pie charts, scatter plots, box plots, bubble charts and radar charts.

The setup is quite "procedural" and requires a lot of boilerplate code, but when it's well set up, the result is very satisfying.

[12] https://github.com/danielgindi/Charts

Realm.io Database

Realm is a mobile database made by a start-up of the same name. Developed by two engineers from Nokia - Alexander Stigsen and Bjarne Christiansen, it is currently one of the most used frameworks used for iOS development.

The main advantages are:

- Mobile-first: built from the ground up to run directly inside phones, tablets and wearables.
- Simple: Data is directly exposed as objects and queryable by code.
- Fast: Realm is faster than SQLite on common operations.

The great thing in comparison to CoreData, a native Apple framework on top of SQLite, is not having to keep the context object everywhere, where a anything can be written in the database. Another great tool shipped with the framework is a migration tool, that has a similar philosophy to Flyway described before.

4.4.4.2 API used

Obviously, the goal is to send the data in as condensed format as possible. For those reasons, the automatically exposed API by Spring-Boot was too verbose. In order to load the data, I created another Controller endpoint in the Tracking Engine application that those handles special requests from the Timeseries application:

1. */mobile/applicationEntryTypes* - returns a list of all KPIs in an application.

```
{ "entryTypes" : [
      {"applicationName":"SAUI
          Test","applicationVersion":"1.0.3","name":"Performance measurement
          switch","total":12654}
   ]
}
```

2. */mobile/applicationEntrySummary* - returns 5 day entry count for a given KPI.

```
{ "entries":[ ... ],
  "metadata":[
      {"entries":11,"timestamp":1464048000}, ...
   ]
}
```

4.5 Tracked Device SDK

In order to integrate the domain vocabulary generated by the Semantic Data Manager, it is necessary to have an interpreter of such vocabulary. For the implementation for Apple devices, I chose native development in Swift.

As stated in the previous chapter, the SDK has two layers - the configuration interpreter and an interface to log an event.

4.5.1 Interpreter

The interpreter enables the storing interface to call functions on it that extract the information from the configuration file, without the need to handle the low-level file access. It is made to be as precise as possible - no errors allowed. When the configuration file is not present it calls a system level fatalError function.

Methods exposed:

- *getDictionary* - returns the whole configuration file as a dictionary object
- *getURL* - returns the URL of the Tracking Engine
- *getAllActions* - returns all actions available in the dictionary

4.5.2 Interface for storing

The interface for storing provides couple functions:

1. *startEvent(itemIndex: Int)* - logs beginning of an event on the given index in the configuration.
2. *finishEvent(itemIndex: Int)* - logs the end of an event on the given index in the configuration.
3. *watchValue(itemIndex: Int, valueIndex: Int)* - logs an action of watched value on the given index assigned to an event on the given index in the configuration.
4. *reportData(compressed: Bool, success: (() -> Void)?)* - reports data to the server, either in compressed format or not and after finishing it calls the optional (not mandatory to use) completion closure.

If an event is to be logged and the interpreter doesn't find it, fatalError is called again.

To ensure clean code and the level of "security"[13] of the code Swift's if-let and guard statements came in very handy:

```
func startEvent(itemIndex: Int) {
   guard let action = ConfigurationReader.getAllActions()[safe: itemIndex]
      else {
         fatalError("Action at given index (\(itemIndex)) does not exist!")
   }

   if let name = action["name"] as? String, issueID = action["issueID"] as? String
       {
      self.addEvent("START", name: name, issueID: issueID)
   } else {
      fatalError("Action at given index (\(itemIndex)) has details in wrong
          format!")
   }
}
```

The guard statement and the if-let statement are somewhat similar - they make sure that an object the developer is trying to access is not null and will not cause any problems. The main difference is that if-let goes inside curly brackets and creates more complexity by indented code. Guard statement allows in-line handling, however it does not allow chaining of the statements, like the if-let statement does.

In fact, even more interesting feature is used in the code snippet above - Swift extension. The safe look-up of an action in the array of all actions is a separate function written as an extension to the standard collection functions in Swift. The extension is written this way:

```
extension Array {
    subscript (safe index: Int) -> Element? {
        return indices ~= index ? self[index] : nil
    }
}
```

There is no need to override the whole Array object and implement a new method. The compiler takes it as a new first-class function on the collection and it can be used in the whole project.

[13] Security here refers to the Defensive Programming coding style leading to reduced number of bugs and problems.

4.5.3 Distribution

When a developer wants to use an external library, there are two standard ways how to get it to the project:

1. Cocoapods
2. Carthage

Cocoapods is more invasive - it links the source code into the project as another sub-targets and creates a new workspace for Xcode. Sometimes it can be frustrating, especially during clean builds when having more than 5 dependencies can take 5 minutes or more to fully compile.

Carthage is a non-invasive way of importing dependencies, however it is not standardized by Apple - it requires initial setup in Xcode and is subject to break during Xcode updates.

Requirements were to create both, which is creating a Podspec file for Cocoapods and a Cart file for Carthage:

```
// Cartfile
github "Moya/Moya" ~> 6.4.0
github "realm/realm-cocoa" ~> 0.99
github "nicklockwood/GZIP" ~> 1.1

//Podspec
Pod::Spec.new do |s|
  s.name         = "SAUI"
  s.version      = "1.0"
  s.summary      = "SAUI Swift framework for logging events."
  s.author             = { "Michal Svacha" => "svachmic@fel.cvut.cz" }
  s.ios.deployment_target = '8.0'
  s.osx.deployment_target = '10.9'
  s.watchos.deployment_target = '2.0'
  s.tvos.deployment_target = '9.0'
  s.source       = { :git => "https://url/SAUI-Swift.git", :tag => s.version }
  s.default_subspec = "Core"

  s.subspec "Core" do |ss|
    ss.source_files = "Source/*.swift", "Source/Plugins/*swift"
    ss.dependency "Moya", "~> 6.4.0"
    ss.dependency "RealmSwift", "~> 0.99"
    ss.dependency "GZIP", "~> 1.1"
    ss.framework = "Foundation"
  end
end
```

4.5.4 Workflow

The usage of the framework after linking is very straight forward. The main object to use is called *Reporter* - it is singleton and accessible via it's proeprty *sharedInstance*. The Reporter class is the interface for storing and offers all methods listed in previous section.

It can be initially set up whether the reporting to the server will be handled automatically or not. If not, the developer has to handle the upload scheduling himself and call the function *reportData* whenever needed. If the automatic reporting is set up, an NSTimer thread object runs in the background, firing every 120 seconds to check if there is anything in the database to be reported. If yes, it gets uploaded to the server.

4.5.5 Technologies Used

Aside from all the technologies previously listed - Moya and Realm, in this component I had to also use Alamofire from which I tried to abstract myself away with Moya. Unfortunately Moya does not support uploading binary data via POST, so I had to use a lower-level tool to get the archived data to the server.

Another technology I used here is a GZIP library handling the archiving of data. The library comes with a choice of level of encryption, which for a text can be naturally selected to the highest number. The object that is archived can only be an NSData object, so I serialized the JSON that is supposed to be archived into UTF8-encoded data:

```
let jsonData = try! NSJSONSerialization.dataWithJSONObject(events, options:
    NSJSONWritingOptions.PrettyPrinted)
let archived = jsonData.gzippedDataWithCompressionLevel(1.0)!
```

Chapter 5

Testing

The whole system has many components and it is a widely accepted fact that the more complex the system is, the more prone to errors and bugs it is. In this chapter I will describe the benchmarking tests I performed on the most complex part - the tracking engine and the user tests I performed not only on the front-end applications but also the whole workflow of the system as well.

5.1 Benchmark Tests

To verify that the Tracking Engine handles extensive load it is important to simulate a real world peak that could occur during everyday usage. I identified the biggest bottlenecks to be the client side (networking issues like poor EDGE connection) and the workers' parsing job (how big tasks can they manage?). The Gateway server has an automatic load-balancer, so there shouldn't be a problem just like with all other automatically scaled components by Amazon (S3 and SQS).

5.1.1 Workers

To verifyhow big load a single worker can handle, I created data on the front-end of various sizes and sent them to the server. I turned on a logging system CloudWatch that told me precise times when a job started on a worker and when it finished.

Number of entries	Execution Start	Execution End	Duration
100	15:46:35.966	15:46:41.595	$5.629s$
500	16:06:57.851	16:07:15.265	$17.414s$
1000	15:51:23.311	15:51:59.060	$35.749s$
5000	N/A	N/A	N/A
10000	N/A	N/A	N/A

Table 5.1: Worker Processing Times

To give it a visual perspective:

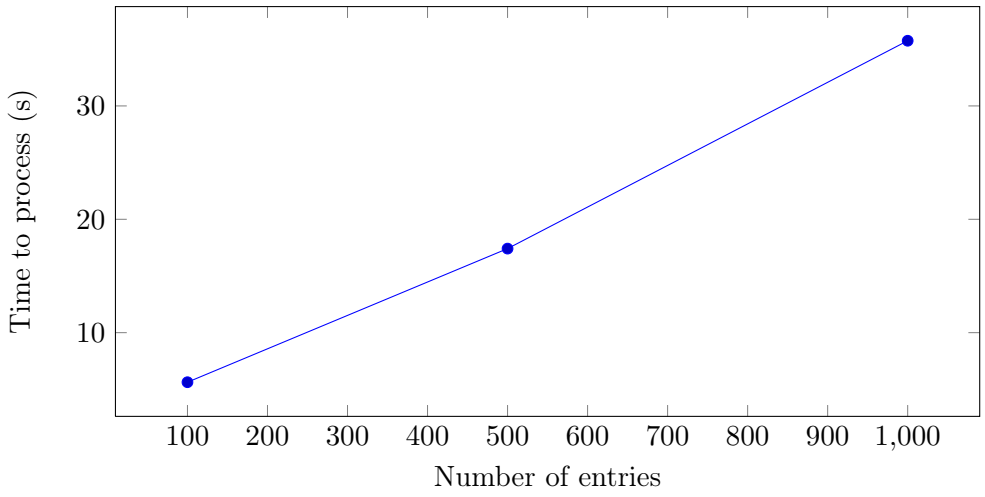


Figure 5.1: Worker Processing Times Graph

The graph itself doesn't show anything that would be out of ordinary, so why did the workers fail to process 5000 and 10000 entries? The scenario of what actually happened is interesting in both cases and it isn't the fault of the workers themselves:

1. **5000 entries:** When the first worker got the message to handle 5000 entries sitting on S3, he reached out for them and started parsing. However he did not make it in the given time of message execution and therefore the message was marked as failed. When the SQS tried to deliver it again, the worker was still processing the data and so it was sent to another worker, who started parsing them. Then finally the first worker returned "200 OK" which resulted in inconsistent state and the whole system went into "Warning". CloudWatch stopped the servers and I couldn't get the execution times. The result was that, unfortunately, the data in the database was saved twice.

2. **10000 entries:** In the last round, it simply took too long and the whole worker crashed and so did the second one soon after the first one.

Conclusion

Even though compression does save some data on the client side, it also introduces a new complexity on the server side that is very hard to parallelize, so it is important to find a "good enough" amount of entries to be uploaded in a single batch report.

5.1.2 Client-side Compression Rate

To continue the search for optimum load on both sides, I measured the performance of the GZIP compression library. I performed it on a real device (iPhone 6S), using logging capabilities in Xcode.

Number of entries	Before compression	After compression	Compression rate
100	51.41kB	0.67kB	76.7x
500	252.98kB	1.65kB	153.3x
1000	504.92kB	2.87kB	177.8x
5000	2520.55kB	12.65kB	199.3x
10000	5040.08kB	24.89kB	202.5x

Table 5.2: Client-side Compression Rate

To give it a visual perspective:

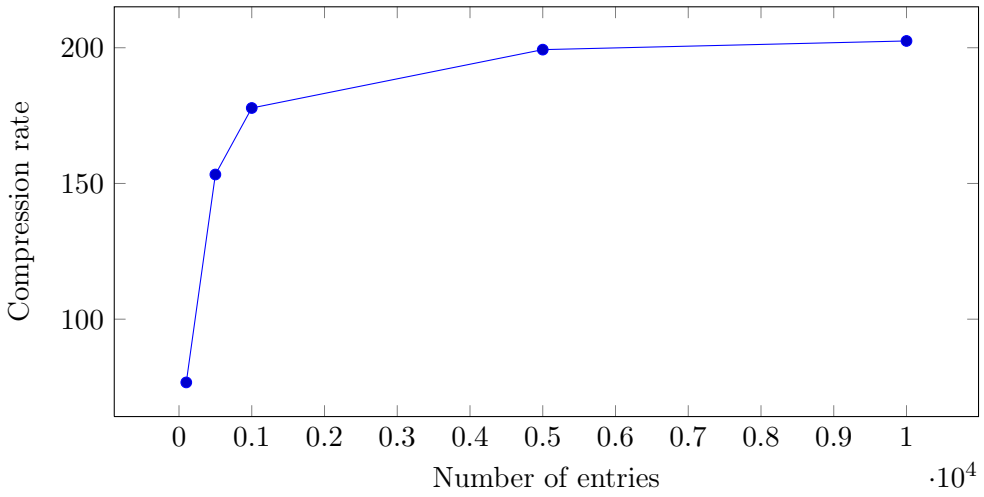


Figure 5.2: Compression Rate Graph

I was quite surprised by the performance, even though I expected it to be performing very well, given the fact that the JSON file structure is very repetitive. What I was equally suprised by, was the logarithmic shape of the curve. In other words, I did not expect the compression rate to slow down.

Conclusion

The best amount seems to be somewhere in the range 400 - 700 entries. Combined with the knowledge from the previous measurement (worker processing time) I would aim to be in lower numbers, perhaps 400 - 600. Than ensures a very good compression rate (~150x) and also good processing time on the backend (~18s)

5.2 User Tests

I performed user tests in three different ways:

1. Personal interviews
2. Scenario test
3. Panel discussion

Each tested different aspects of the whole solution and each brought me some new insights into which problems I overlooked during the development.

5.2.1 Personal Interviews

The personal interviews were held with three different people, each from a different department. I showed them the whole workflow and asked them to give me feedback on anything they want.

5.2.1.1 Test Setup

The interviews were held in a company kitched - an area where people relax, chat and hang out during their personal breaks. The work was presented on a laptop (JIRA workflow and the deployment in the AWS console, should it be of any interest) and an iPhone 6S. Both devices were directly accessible to the person being interviewed.

I extracted the highlights from each interview:

1. Associate Director, Applied Technology

   > "I like the fact that I define the domain vocabulary once. Once and that's it. That's great."
   >
   > "Could we wire-up the installation IDs to the user IDs to ensure that 10 installations on the same device are counted as one?"
   >
   > (Looking at the Timeseries Application) "Yes! That's exactly what I want to see."

2. Mobile Application Development Lead

   > "I didn't quite understand the concept of WATCH when you first said it, maybe name it differently, like "Event" for example?"
   >
   > "Is it possible to produce a user flow? Like how the user moves throughout the application?"
   >
   > "How do you address a change in the configuration file? Is there a way to ensure I am not reporting something I was reporting before?"
   >
   > "How do you ensure that the content is sent later if it fails? Are you using operations.io or some other framework?"

3. UX Lead

   "Could you show me, how would I, as a UX designer use this? Would I use it only at the start or towards the end as well?"

   "Can I edit measurable items?"

   "So here, I would add "WATCH: Button Start in the header" and the programmer would receive it in his configuration file, right?" (a task a programmer would refer to as "Back button in a UINavigationBar")

The interviews were so fruitful - they validated the main idea as all three were very interested in the project and were asking how soon we could deploy it to test it.

At the same time, they uncovered the questions I didn't ask myself, such as the configuration file versioning and change management.

And lastly, they brought up ideas for future development.

5.2.2 Scenario Test

The scenario test was performed on the administrative application, made for creating configuration files for developers to implement. The people I asked to test the application were from different departments and also different positions (technical support, assistant, programmer etc.)

5.2.2.1 Test Setup

The test was performed on an iPhone 6S device, running the real application, connected to real data. Issues in JIRA were created by me ready to be used during the test (there was no need for the testers to create those issues, because that is JIRA workflow, not the workflow of the application).

The test was carried out in a semi-informal environment - in the company library on a comfortable sofa, surrounded by bookshelves. I sat in front of the tester, not peaking over his/her shoulder.

I asked each tester to narrate every step he/she made and point out any inconvenience, should one ever occur. I ensured everybody, that criticism is welcomed and that there is absolutely no problem if they fail performing the task. That is what the test should be about and it is vital that the tester is aware of it.

5.2.2.2 Scenario

The scenario was made up this way:

1. Start creating a new configuration.
2. The configuration should be made for project "Semantic Analysis of User Interactions".
3. Select to measure the performance of the measurement switch.
4. Are there any special values to be tested?
5. Generate and upload the configuration file to Dropbox.
6. Can you tell me precisely what was in the configuration file again?

The designed path to achieve these was:

1. Click on the plus button in the top right-hand corner in the navigation bar.
2. Scroll to letter "S" and select the project "Semantic Analysis of User Interactions".
3. Click on the issue "Performance measurement switch".
4. The correct answer is yes - it is written in the comment below the name of the issue. There are two watched values.
5. Click done and then click on the button "Upload to Dropbox". After clicking done after a successful upload, the application is automatically brought back to the first screen.
6. Click on the button in the bottom right-hand corner "View Generated Configurations". Select the one with the most recent time by clicking on it and it leads directly to a list with all the issues in the configuration.

How did the testers perform:

Step	Person 1	Person 2	Person 3	Person 4	Person 5
1.	OK	OK	"I guess this plus?"	OK	Tracked Applications
2.	OK	OK	OK	OK	OK
3.	"How is this sorted?"	OK	OK	"Is it this?"	"Is there a search field?"
4.	OK	OK	OK	OK	Clicked too fast
5.	OK	OK	OK	OK	OK
6.	"In Dropbox or in the app?"	OK	OK	"Oh that is a button!"	OK

Table 5.3: User Test Results

The user tests showed some new insights into the workflow of the application and my perception of the user interface. I thought it was crystal clear, but apparently some people don't see certain things the same way and that is OK. It's important to reflect that in the user interface, so it tells a story or is self explanatory.

Conclusion

There should definitely be a search bar in the list of issues. And the sorting algorithm should be revisited too. It seems like it is sorted by the ID of the issue in JIRA. That can become really messy with a large amount of issues in a project. Also the button leading to previously generated configurations should be highlighted a little better to evoke the button-y feeling.

5.2.3 Panel Discussion

This test I expected to be the most cruel one - I invited programmers with varying levels of experience and wide range of specializations:

1. Person 1 - Technical Lead
2. Person 2 - C# enthusiast, Machine Learning Engineer
3. Person 3 - Polyglot Programmer with experience in every aspect of development
4. Person 4 - C/C++ Programmer with 20+ years of experience

I presented them the whole idea and also the implementation and deployment. Discussion was surprisingly not very heated, but it sure opened couple doors I didn't consider.

In a wide range of questions from topics like deployment, scalability and usability of the mobile applications, the most interesting questions were:

- "What is semantic about this?"

  The "semantic" in this is the fact that I am working with the domain vocabularies defined by people with real business needs. Semantic here refers to the content and meaning of the reported events. It is the higher-level analysis.

- "Is it possible to do dynamic loading of the configuration after it's been compiled?"

  Unfortunately not yet, but it is something to be considered. It is definitely doable.

- "Are you handling versioning of the configurations?"

  Not at the moment, but it is a possibility to explore, especially if dynamic loading was to be supported.

- "So if JIRA API changes, the whole workflow is broken?"

  Yes, very broken.

Conclusion

An eye-opener as well. In the era of "Semantic Web" it is important to be able to explain the system clearly - what it does, what it's meant to do. The focus is the semantics - the meaning of all the user interactions.

Some nice ideas were discussed as well, especially with dynamic loading - it could be very interesting to be able to reload the names, but I can't imagine at this moment how to address the already coded events assigned to a workflow with given start and finish.

Last but not least - it is important to watch JIRA and inform the department who is in charge of updating JIRA to keep in mind that there is an application that is dependent on it. Thankfully it is not the end of the world, if the Semantic Data Manager is down for few hours, because the main load is done by the Tracking Engine. However it is out of my hands, if the JIRA API changes.

Chapter 6

Conclusion

The aim of this work was to identify disconnected data endpoints and figure out a way how to connect them and make use of them. I demonstrated the need on misunderstandings between different teams and pointed out where value could be more driven.

In the second part, I analyzed the existing tools on the market and I assessed their usability in the given context. Not being fully satisfied with their functions, I took the initiative and designed a system that would be in-house, deployed in the AWS VPC environment.

In the third and fourth part, I designed and developed the system into a fully viable product (= MVP). As I went along, I kept validating my ideas with my peers and supervisors.

In the last part, I set myself a goal to test the system from many aspects - idea, usabilty and performance. While some tests went better than the others, it brought me so many valuable insights and pointed out places with room for improvement that I hadn't even considered before. For that I see the tests as an immense success, regardless of the actual results - it shows that the idea is valid, people are interested in it and there is something to continue working on.

6.1 Technical debt

TBD

6.2 Future development

TBD

6.3 Closing remarks

TBD

Bibliography

[1] P. R. Appsee. Appsee partners with twitter.
https://www.appsee.com/press-releases/appsee-fabric-integration, 8. 12. 2015.

[2] W. Chang. Crashlytics is joining forces with twitter.
http://www.crashlytics.com/blog/crashlytics-is-joining-forces-with-twitter/,
28. 1. 2013.

[3] A. Documentation. Aurora on amazon rds.
http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Aurora.html,
up to date 24. 5. 2016.

[4] A. Documentation. What is aws elastic beanstalk?
http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html,
6. 8. 2013.

[5] S.-B. Documentation. Spring boot reference guide.
http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/,
up to date 16. 5. 2016.

[6] J. Highsmith and A. Cockburn. Agile software development: the business of innovation.
Computer, 34(9):120–127, Sep 2001.

[7] F. Krause. The easiest way to automate building and releasing your ios and android
apps.
https://github.com/fastlane/fastlane, up to date 22. 5. 2016.

[8] B. Leonard. Tealium iq - the world's leading enterprise tag management solution.
http://tealium.com/products/tealium-iq-tag-management-system/, 16. 12. 2015.

[9] B. Leonard. Tealium's customer data platform achieves hipaa milestone.
http://tealium.com/press-releases/tealiums-customer-data-platform-achieves-hipaa-mil
16. 12. 2015.

[10] K. MacDonald. The data driven pharmacy today.
https://kitcheck.com/2015/06/data-driven-pharmacy-today/, June. 2015.

[11] B. Matsugu. The limitations of linear methodology – waterfall or agile... or both?
http://www.blueprintsys.com/the-limitations-of-linear-methodology-waterfall-or-agile
20. 10. 2014.

[12] P. R. New Relic. People, process, privacy.
https://newrelic.com/why-new-relic/security, up to date 22. 5. 2016.

[13] D. Radigan. Jql: the most flexible way to search jira.
http://blogs.atlassian.com/2013/01/jql-the-most-flexible-way-to-search-jira-14/,
29. 1. 2013.

[14] C. Rawson. ios 5 deprecates udid as identifier for developers, but it's not the end of the world.
http://www.engadget.com/2011/08/19/ios-5-deprecates-udid-as-identifier-for-developers-but-i
19. 8. 2011.

[15] J. Seibert. Introducing fabric.
https://blog.twitter.com/2014/introducing-fabric-india, 24. 10. 2014.

[16] P. Webb. Spring boot – simplifying spring for everyone.
https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone,
6. 8. 2013.

[17] A. Weber and I. R. Thomas. Key performance indicators. *Measuring and Managing the Maintenance Function, Ivara*, 2005.

Appendix A

List of abbreviations

API Application Programming Interface

AWS Amazon Web Services

CSRF/XSRF Cross-site Request Forgery

EB Elastic Beanstalk

EC2 Elastic Compute Cloud

JSON JavaScript Object Notation

KPI Key Performance Indicator

MVP Minimum Viable Product

PaaS Platform as a Service

PoC Proof of Concept

RDS Relational Database Service

S3 Simple Storage Service

SaaS Software as a Service

SDK Software Development Kit

SDLC Software Development Life Cycle

UX User Experience

WWDC The Apple Worldwide Developers Conference

Appendix B

Instalační a uživatelská příručka

Tato příloha velmi žádoucí zejména u softwarových implementačních prací.

Appendix C

Obsah přiloženého CD

Tato příloha je povinná pro každou práci. Každá práce musí totiž obsahovat přiložené CD. Viz dále.

Může vypadat například takto. Váš seznam samozřejmě bude odpovídat typu vaší práce:

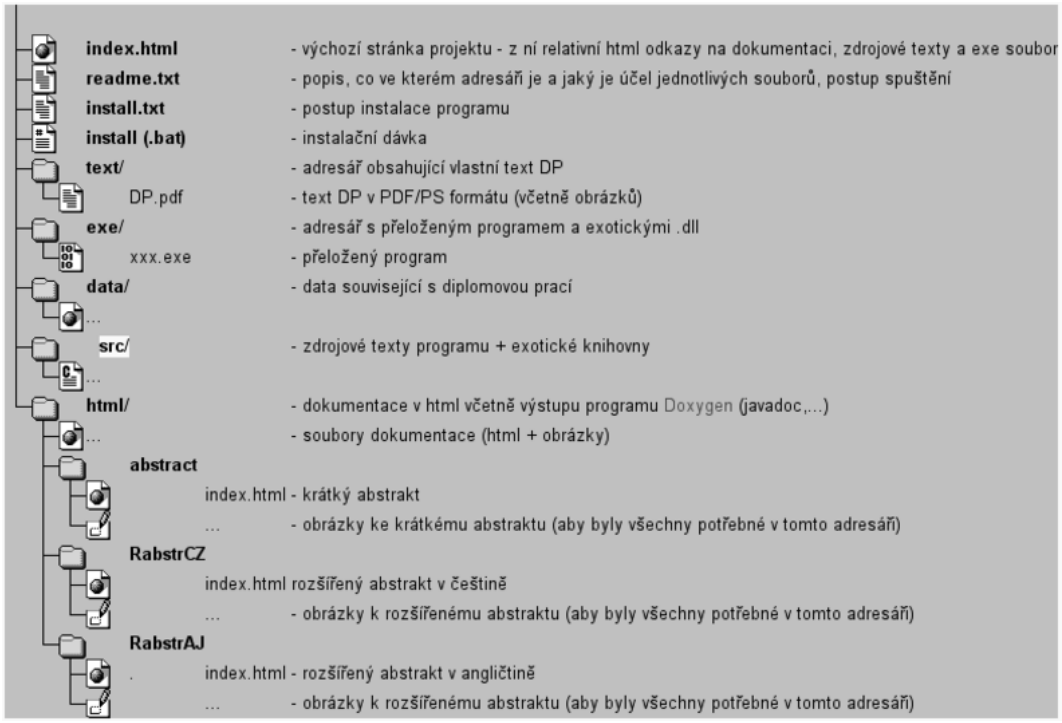


Figure C.1: Seznam přiloženého CD — příklad

Na GNU/Linuxu si strukturu přiloženého CD můžete snadno vyrobit příkazem:

```
$ tree . >tree.txt
```

Ve vzniklém souboru pak stačí pouze doplnit komentáře.

Z **README.TXT** (případne index.html apod.) musí být rovněž zřejmé, jak programy instalovat, spouštět a jaké požadavky mají tyto programy na hardware.

Adresář **text** musí obsahovat soubor s vlastním textem práce v PDF nebo PS formátu, který bude později použit pro prezentaci diplomové práce na WWW.