29-09-2013

Architectural Solution

Fly With Us - Airline Reputation Management

Group Gr_d1_W1
Emrullah Cakir 10469990
Bartol Karuza 10495495
Wai Yi Leung 6162908
Robin Perz 0418447
Axel Polet 10591532

Table of Contents

Introduction

This document contains the architecture we designed for FlyWithUs. The system described by this architecture is meant to collect, analyze and deliver ratings and/or reviews about various airlines on multiple criteria.

We have defined all the concerns from the stakeholders we interviewed in Appendix A. That document contains a description of the stakeholders, business goals, requirements and quality attributes. Based on that document we have come up with an architecture we describe in three views. At the start of each view we reference back to Appendix A in order to show which concerns are satisfied by that view.

The first view (chapter 1) deals with the important business goals and the two choices that impact those goals significantly. The choices are about data storage and data processing.

Next we show a functional decomposition view in which the functional modules are laid out. This view focuses on the responsibilities of each module that the system will hold. We also zoom in on two important modules in this view, elaborating more on their functionalities.

Finally we present a more technical pipes-and-filters view in which the process of acquiring and transforming the source data (from various sources) is elaborated.

The choices we have made in order to come to this architecture are presented in Appendix B. The most important choices are described using decision tables, while the smaller decisions are elaborated shortly.

1. Business Goals View

Business Goals: G1, R2, M1, M2
Quality Attributes: Q3

The business goals view shows the business impact of available alternatives for several decision points in the architecture. These decisions have been selected, based on how much they affect business goals or system quality attributes.

The business goals in this context are:

- G1: the application should be easily extendable to support growth of the organization (extensions should be made within 1 month)
- R2: the application should be 'green' (CO2 neutral)
- M1: the application should have a low time-to-market (within 6 months)
- M2: the application should not be tight to specific vendors (vendor-lockin)

The other business goals are implicitly satisfied when implementing the systems functionality, as covered in chapter 2.

The choices in this context are:

- Choosing the platform on which the application will run and the analytics will be done
  - Hadoop (distributed batch processing)
  - Storm (distributed real-time processing)
  - Non-distributed (sequential)
- Choosing the database system where the 'big data' will be persisted
  - Relational SQL database system
  - Non-relational or NoSQL database system

Each choice has a positive or negative impact on one or more goals and quality attributes (defined in Appendix B). The view focusses on how the data can be stored and processed.

Not really Big Data

Before pointing out the different options and possibilities we need to define the size of the data we actually have to store/process. It turns out we are not really talking about Big Data:

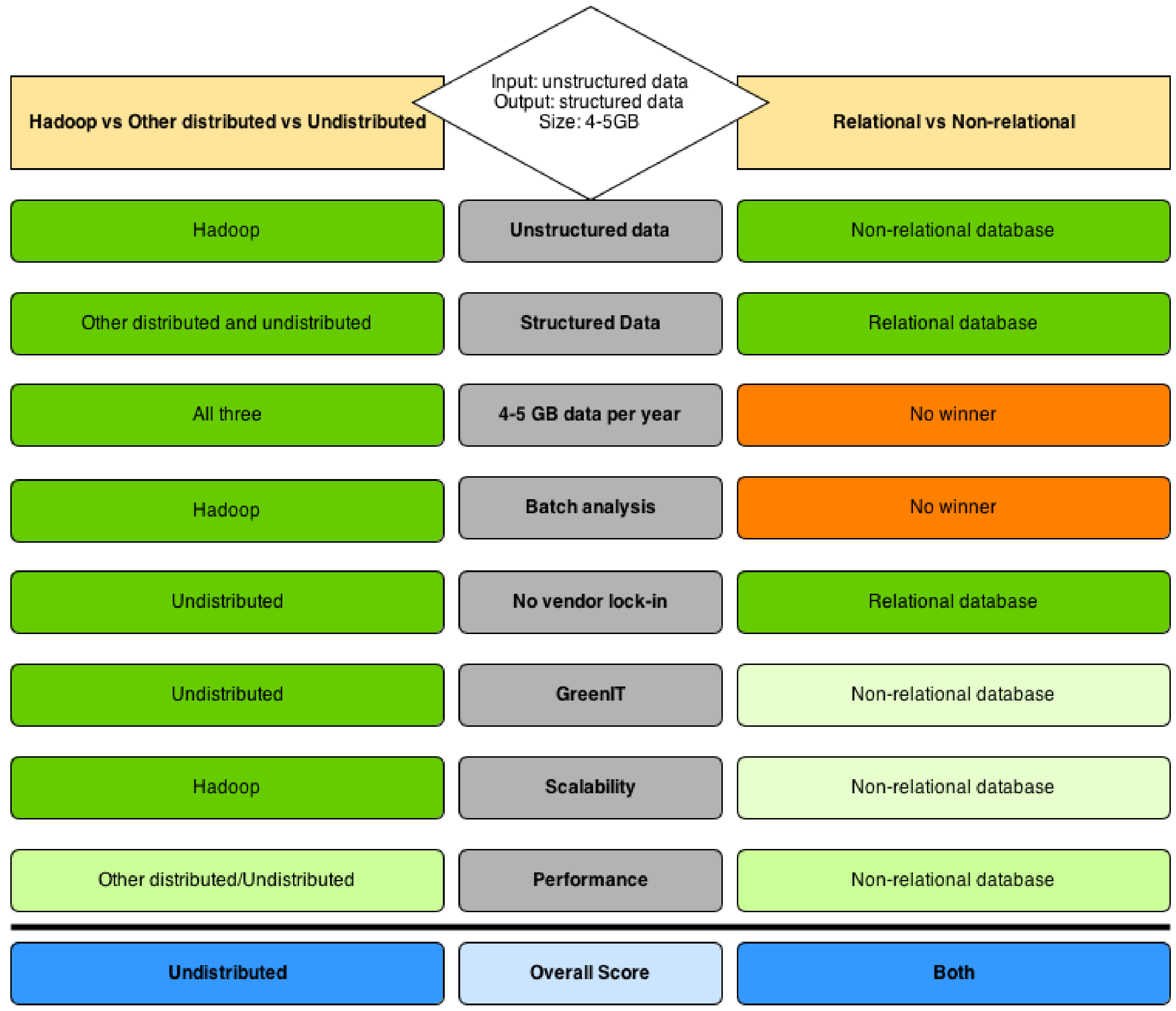
The data we have to process are mainly tweets from Twitter and review data from different review sites. As this data can contain different attributes depending on the review site and even depending on the review, this is unstructured data. We have prudently estimated the yearly data that is gathered by the system at around 5 GB. The 5 GB is estimated as follows.

Twitter

There are many analytic tools to analyze Twitter data. Using one of these tools[1], we have estimated the number of tweets about the top 50 airlines [2] in the world to be 77.000 per month. This amounts to almost a million tweets per year. The estimated size per tweet, when taken from the API, is said to be 4 kilobytes. This will lead to about 4 gigabytes of data.

Review sites

To be able to estimate the size of the data that will be gathered from review sites, we have taken a look at the number of flight reviews at a number of sites. We have concluded that the number of real flight reviews, so not the general airline reviews, is not that high. The largest flight review site[3] we have found amounts to about 700 reviews a year. As this site allows large reviews, we have made a (very) prudent estimation of the size of a review at 100 kilobytes. This amounts to about 70 MB a year.



Hadoop vs Other distributed vs Undistributed	Input: unstructured data Output: structured data Size: 4-5GB	Relational vs Non-relational
Hadoop	**Unstructured data**	Non-relational database
Other distributed and undistributed	**Structured Data**	Relational database
All three	**4-5 GB data per year**	No winner
Hadoop	**Batch analysis**	No winner
Undistributed	**No vendor lock-in**	Relational database
Undistributed	**GreenIT**	Non-relational database
Hadoop	**Scalability**	Non-relational database
Other distributed/Undistributed	**Performance**	Non-relational database
Undistributed	**Overall Score**	**Both**

Data

We have both structured as well as unstructured data. As we will perform some standard analysis on the data and save this data for requests by airlines, we have a lot of standard data, that is fast and easy to query for the B2B and B2C. This type of data is ideal to be stored in a relational database, because it is structured. This data can be queried, so no real analysis is needed after this step. Therefore, non-distributed analysis is best suited for the job as this has no performance or space overhead.

Next to this, we have the review data from different sources. All these review sites can have different ratings or, like Twitter data, sentiment values for different attributes. Some might even have a review text per attribute. Also, the flight can be specified in different ways. Some might be in Business Class and some might be in Economy Class. This data is really unstructured and is best defined in a document-oriented database[5][9].

Performance [Q3]

A main concern of the stakeholders is the performance of the application. As our data only needs to be analysed once or twice a day, this would lead us to look at a batch processor. One of the widely used options for batch processing is Apache Hadoop. It is a highly scalable application with a lot of tools. The real use of Hadoop is in Big Data, because it can scale to thousands of nodes without the need to refactor anything.The startup of a Hadoop job does lead to some overhead[6], which is why we think that it’s performance is worse for our relatively small dataset.

Green IT [R2]

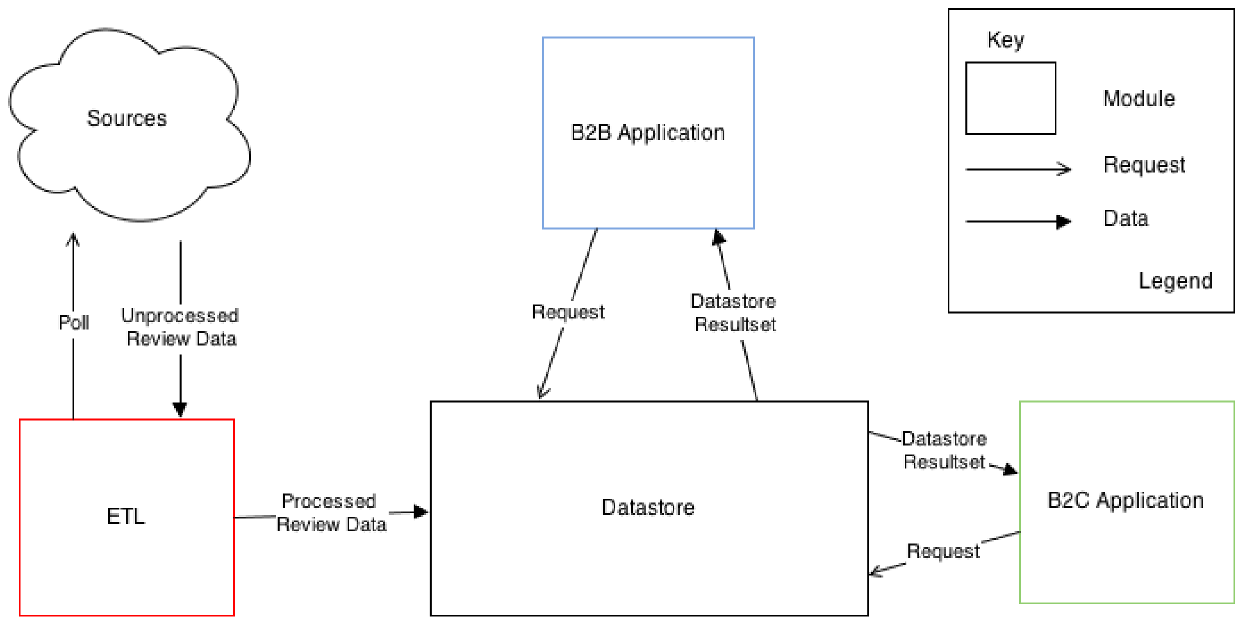
Another stakeholder concern is GreenIT. GreenIT does not only mean that green power supplies are used, but it also focusses on low energy consumption and avoiding unnecessary use of energy. The most efficient way of using energy is to use as little different servers as possible. That’s why we believe that the undistributed way of analysis is more green. Another way of minimizing energy consumption is to use a more efficient database. Although relational databases can replicate non-relational databases by optimization and efficient database design, non-relational databases are more time efficient in most cases. This is why we believe the non-relational database is more green than the relational databases.

Vendor lock-in [M2]

The last stakeholder concern we would like to address is to avoid vendor lock-in. Using Apache Hadoop would require an investment in both time and money, which would mean it would be a big step to turn away from Hadoop after it has been adopted. An own implementation of a distributed or undistributed analysis framework would avoid the vendor lock in completely.

2. Functional Decomposition View

Functional Requirements: 1-5, 7-9 Quality Attributes: Q1



The view above shows the functional layout of the system. Each module presented satisfies some of the functional requirements as we describe below.

Datastore [F3, F4, F8]

Central module in the system. This module houses storage of ETL processed data (which includes normalized review data and calculated ratings), analytics to produce reports and perform other analyses, a database to hold B2C customer data and an admin interface through which administrators perform moderations on databases. In chapter 2.1 we present a more detailed view of the module.

ETL [F1, F2]

Module to extract, transform and load the review data retrieved from the various data sources. This module houses a number of components to filter the retrieved data on certain aspects and apply the valuation algorithms on the reviews. This process is explained in detail in chapter 2.3

B2B Application [F5]

This module is responsible for displaying a user interface on which the B2B user (for instance KLM) can configure and retrieve specific reports on airline quality. We elaborate on this module in chapter 2.2.

B2C Application [F7, F9]

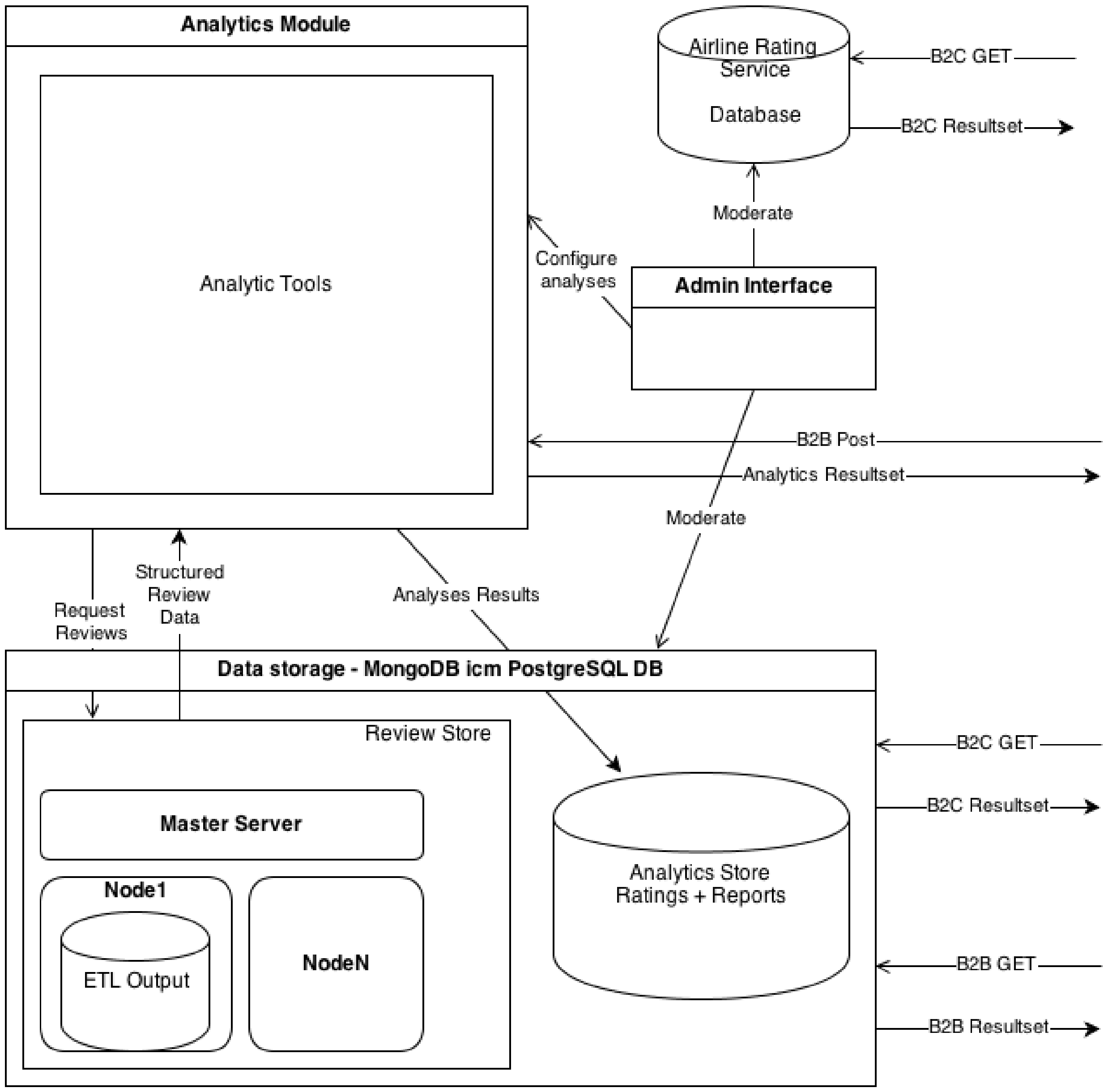
This module is the consumer front-end for the FlyWithUs system. It consists of a public website, with optional user authentication. The consumer will be able to find and post reviews on airlines and flights at this website. This module will also be used by B2B users to respond on posts of B2C consumers. We elaborate on this module in chapter 2.2.

Modifiability [Q1]

By separating the functionality in four modules each module allows for extension/modification without significantly affecting the other modules. Next to this, the ETL, B2B and B2C modules are also designed to deal with modifiability. (see chapter 2.2 and 3).

2.1 Datastore Zoom

Functional Requirements: 3, 4, 8
Quality Attributes: Q2, Q3, Q5
Business Goals: R1



The datastore module is the central part of the system, all other modules use this module. Each module is shortly elaborated below:

Data storage [F3, Q2, Q3]

This module is used to store review data (in the review store) and analytics data (in the analytics store). The review data will be passed by the ETL module (not shown in the view) and the analytics data comes from the ETL module and the analytics module. The key design decision made here is the combination of a non-relational (or NoSQL) and a relation database.

The non-relational database (this could be MongoDB[7]) is used to store the normalized (see chapter 3) review data (such as entire Twitter posts). Non-relational (in this case document-oriented) databases can store schemaless[6][9] data which is ideal for the data retrieved from various data sources.

The relational database (could be PostgreSQL, which is opensource and powerful[8]) is used to store results of the analyses performed by the ETL module and the analytics module in a specific data scheme (the data has become structured). Possible performance issues (relative to NoSQL databases) can be countered by good database design (see also Appendix B)

Analytics [F4]

The analytics module precalculates some standard analytics, which are available for B2B and B2C. This decision implicates that most requests can be fulfilled immediately from precalculated data, which betters the performance of the application.

The B2B user can also create a simple custom request which is directly handled by the analytic module. This request can perform additional analytics on the existing data. These analytics won't be stored in the database. The analytics module will return the calculated dataset directly to the B2B module.

To be able to perform the analysis, the analytic module can request review data from the datastore. The data that is returned can be either precalculated analysis or rating data as well as the structured review data.

Airline Rating Service Database [R1, Q5]

This (relational) database houses the user/application data for the B2C application. This includes users, roles, authentication data, and posts/ratings entered by B2C users. By splitting the user database from the other data storage we make it possible to deploy this database in the Netherlands or EU, while the rest of the system can be deployed elsewhere (possibly reducing costs).

Admin Interface

This module stands for an integrated user interface (client application or server connection) through which the administrators/moderators of the system can execute CRUD operations.

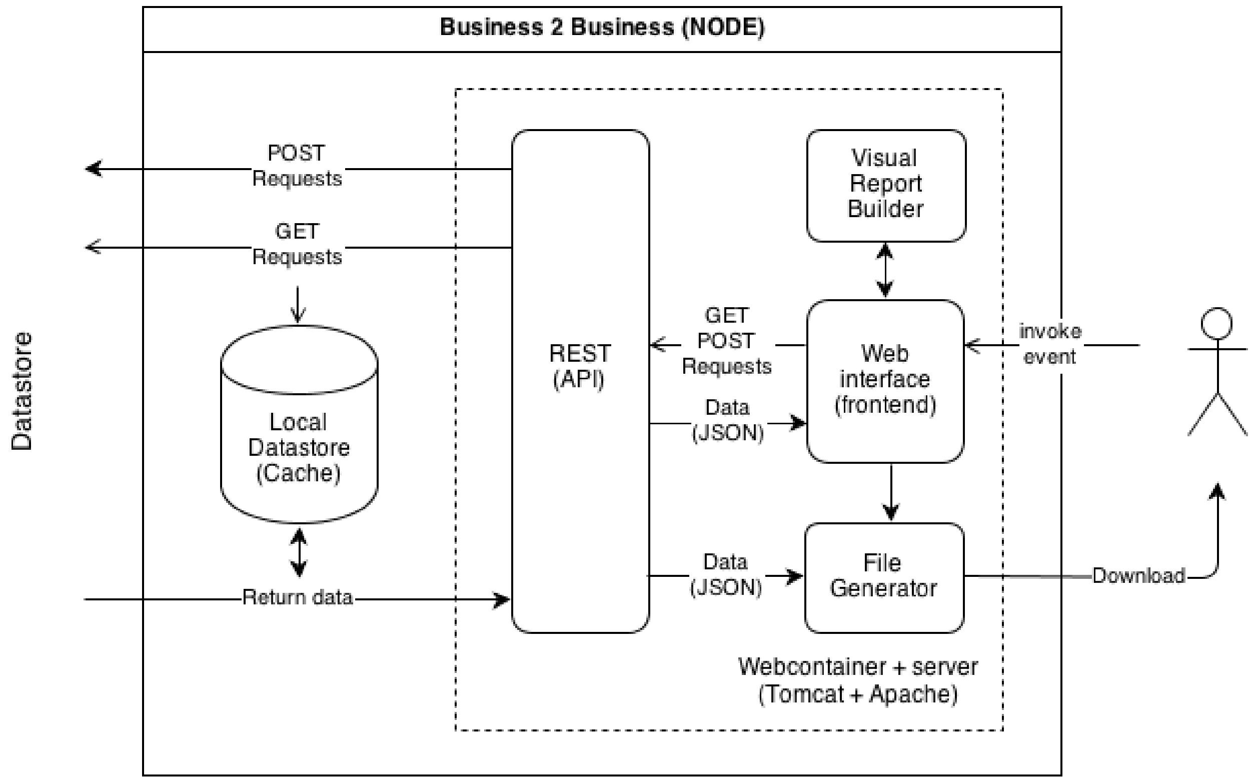
2.2 B2B/B2C Zoom

Functional Requirements: 5, 7, 9
Quality Attributes: Q1, Q2, Q3, Q5

The B2B/B2C zoom describes the disclosure of the information from the datastore to the business clients and consumers using the FlyWithUs rating service. We zoom on the 2 components as they have been separated as two different parts based on our analysis of the problem at hand.

Even though the B2B and B2C components resemble each other and could have been integrated into a single component we have decided not to follow this path based on our analysis of the requirements and quality aspect we wished to achieve for our stakeholders. One of the major reasons for implementing the B2B and B2C components as separate parts is because it is much easier to guarantee performance, modifiability and scalability [Q1, Q2, Q3]. These two components will be implemented as nodes. If more is required it will be plug-and-play, with geolocation load balancing. We could have multiple instances of the B2B and B2C running for the whole system to guarantee performance and scalability.

B2B Component [F5]

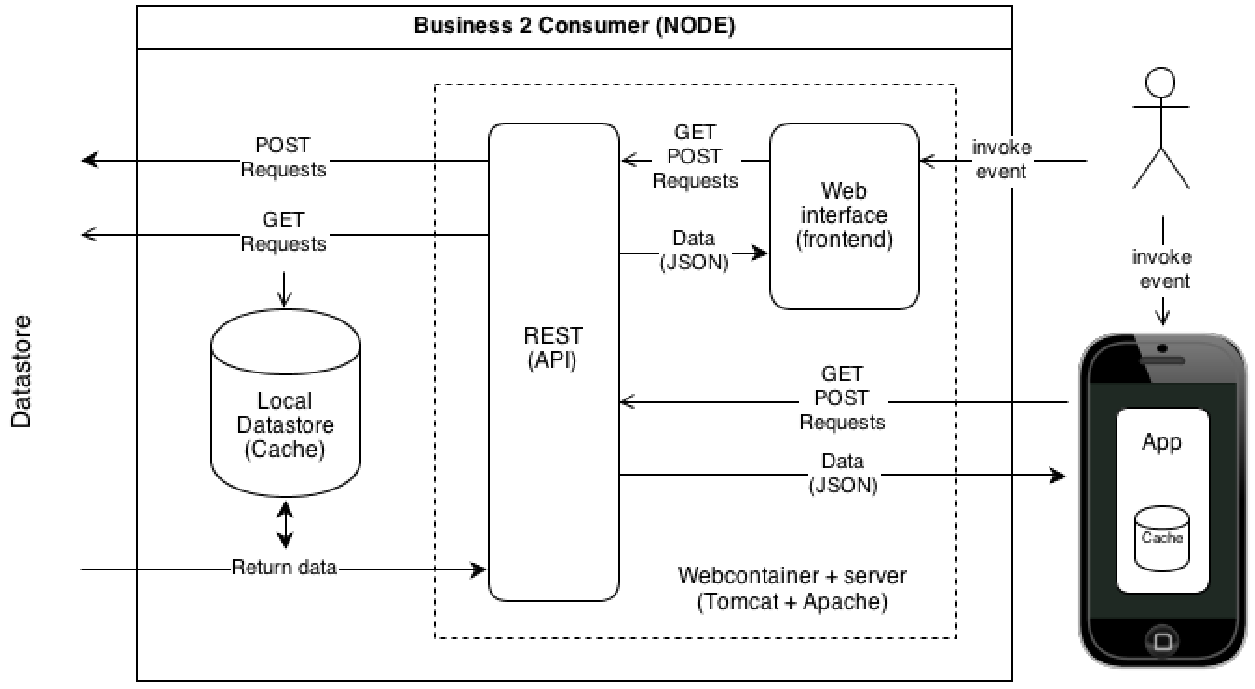


The B2B component is the part of the system that will be used by flight companies that have a contract with FlyWithUs. In the detail view above we can see the sub-components of the B2B module. The B2B module will be most likely a webservice with a local datastore for the purpose of caching frequently requested data. [Q2]

The front-end of the component (GUI) will be implemented as a web-client. This web-client will have all the same functionality as the B2C component. As extra the B2B component will have the ability to build “custom queries” to analyse specific situations from our datastore. This “query builder” will be a visual sub-component only available in the B2B module. By building custom queries contractors can extract useful knowledge from our datastore. These custom queries will be processed, on request, by the analytical module in our datastore. The results will then be returned to the B2B module for presentation purpose or generation graphs and/or reports. These reports can be intensive on our datastore and analytical module, as they may need a large data chunks if large time windows are specified for analysis purposes.

By doing batch processing in our analytics component for time intervals we strive to lessen the load on these raw data sets and try to use averages to increase performance. [Q2]

B2C component [F7, F9, Q5]



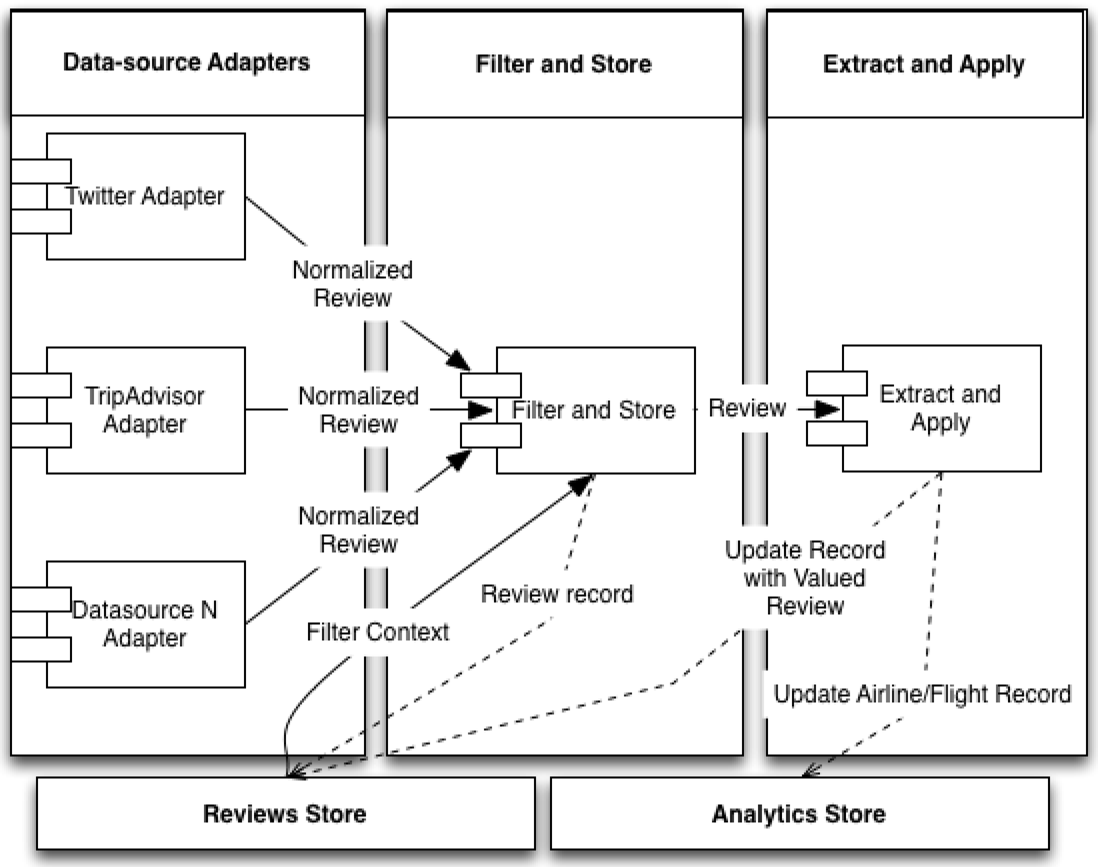
We concluded from our analysis that the B2C component will be under a higher load most of the time compared to B2B component. The B2C component will be serving preprocessed analytics results that get processed on batch runs by the datastore or the ETL component. Consumers will not have the abillity to dig into specific analytical results by building their own queries and straining the system for these results. By returning specific and static results the B2C consumer will not strain the analytics component. The B2C component will also have its own caching sub-component which in turn will store frequently requested data for a specifiek B2C node. By implementing a RESTful API it should also be easy to implement new connections to external apps. for example a iPhone (web)client.

User authentication will be coupled to our user database to have a central and single sign on for all the B2B/C nodes in our system. Communication between the nodes and datastore will be on https protocol. There also will be a special role defined for B2B users to respond on B2C customers via the same user interface [F9].

3. ETL Pipes-and-Filter View

Functional Requirements: 3
Quality Attributes: Q1, Q4

The pipes-and-filter view of the ETL module expresses the interactions between the components of the ETL module as reviews go through the system. The view has been enhanced with the specific interactions with the Data store component, to indicate the intermediate and final data that is being persisted.



Data-source Adapters

The adapter components represent an encapsulation of all extraction logic specific to the data source that the adapter is assigned to. A separate adapter will be created for each data source that the review system needs to handle.

The main purpose of the adapters is to contribute to modifiability[Q1]. The setup of the adapters enables modifiability, by allowing new adapters to be added without changes to the existing adapters or the following processing steps. Adding a new data-source can be done relatively quickly at the expense of being able to change the structure of the normalized rating, as those changes will impact every adapter.

The adapters will perform the first contribution to data quality by dropping records that do not contain the minimum specified fields for the data source. A large portion of the “empty” records can be prevented here [Q4]. All adapters will be able to transform data source specific data structures to the normalized (semi-structured) data entity that is represented by the Normalized rating.

Normalized Review

A normalized rating is a semi-structured data entity consisting of the essential information that every review should contain, such as timestamp, value, scale, etc.. The value is called normalized because all adapters will produce reviews in this general format.

Filter and Store

The filter and store component receives a stream of normalized ratings from the adapters and implements various algorithms to remove unsuitable and unusable reviews. This component is the most important part of the ETL system that ensures data-quality[Q4].

For this filtering process the component uses contextual data that it retrieves from the Reviews store. This data can be necessary for some filtering techniques that need to know if for example the same poster has posted before and how many times.

After filtering the reviews are persisted into the Reviews Store for further analysis. The reviews are passed on to the next component for further processing.

Extract and Apply

The Extract and Apply component does all the post-filter handling of the incoming reviews. It receives the filtered reviews from the Filter and Store component as a stream and processes them with the systems rating valuation algorithms. From this processing the review is given a rating on the global system scale and this rating is stored with the review in the Reviews Store. The relevant totals per airline or per flight are then updated in the Analytics Store. This ensures a constant up to date rating for all airlines from all data sources.

Any additional per review processing can be added to this component as these algorithms become required.

Appendix A - Requirements

1. Stakeholders

There are four stakeholders (including one from FlyWithUs) to be distinguished. Each stakeholder has his own interests and stakes in the system.

Stakeholder 1: Peter Klijn

The initiator of FlyWithUs. His main concern is the functionality of the system.

Stakeholder 2: Sven-Erik Haitjema

Represents EU-claim. Main focus is transparency. He wants the system represent the airliners fairly, no airliner should have any benefit over any other airliner in its reviews. He also wants the privacy of the European Citizen to be respected.

Stakeholder 3: Philipp Darkow

Represents the Dutch government, since the system needs to be hosted in the Netherlands. His primary concerns are the privacy laws and the fact that the system should be 'green'.

Stakeholder 4: Sinan Ceylan

Represents Air France-KLM, which is partly funding the project. His primary concerns are costs and ROI, but also the usability towards business clients as well as consumers.

2. Business goals

For describing and defining the business goals we derived from stakeholder interviews we use segments of the method stated by Paul Clements & Len Bass[4].

Meeting financial objectives

No	Definition	Stakeholder
F1	● **Goal Object (GO):** Fly With Us organization ● **Environment (E):** Start-up organization ● **Goal (G):** Make profit of Business to Business clients with the Fly With Us application ● **Goal Measure (GM):** Deliver working application	S1
F2	● **Goal Object (GO):** KLM Webcare customer service ● **Environment (E):** Operation costs of current customer service ● **Goal (G):** Decrease costs of KLM customer service by responding more effectively/appropriately to	S4

	incidents ● **Goal Measure (GM):** Cost indicators should show a lower cost after 1 year.	

The organizations growth and continuity

No	Definition	Stakeholder
G1	● **Goal Object (GO):** Fly With Us application ● **Environment (E):** New data source become available, other airliners want to join in, need for an mobile application ● **Goal (G):** Integrate/implement new data source, develop/create new client into the FlyWithUs system ● **Goal Measure (GM):** Each of the new 'features' can be implemented within 1 month	S1

Meeting responsibility to society

No	Definition	Stakeholder
R1	● **Goal Object (GO):** Users of FlyWithUs ● **Environment (E):** Privacy of webusers is valued in EU ● **Goal (G):** The FlyWithUs user data should be stored in a way that the US government should not be able to use it. ● **Goal Measure (GM):** Storage should at least be hosted in EU. Preferably in NL.	S2
R2	● **Goal Object (GO):** Fly With Us Application ● **Environment (E):** Society focuses on green products ● **Goal (G):** The Fly With Us system should be 'green' ● **Goal Measure (GM):** The system should be CO2 neutral and minimizing power consumption	S3

Managing market position

No	Definition	Stakeholder
M1	● **Goal Object (GO):** Fly With Us application ● **Environment (E):** Competitive environment and saturated market	S1

	● **Goal (G):** Application should be deployed in production as soon as possible. Low time-to-market. ● **Goal Measure (GM):** Product debuts in 6 months	
M2	● **Goal Object (GO):** Fly With Us application ● **Environment (E):** Possible development organizations tight to vendors, development team ● **Goal (G):** The application should not be subject to vendor-lockin (depending on services/support of third party) ● **Goal Measure (GM):** System built on libraries/applications/tools which are commonly known and interchangeable.	S1
M3	● **Goal Object (GO):** KLM ● **Environment (E):** Comparing to other airliners ● **Goal (G):** Improve KLM airliner quality in comparison with other airliners ● **Goal Measure (GM):** Measure qualities by using data/reports from the FlyWithUs System	S4

3. Functional requirements

The items stated below are the functional requirements of the system:

1. The system must be able to collect large amounts of structured or unstructured data from various datasources (f.e. Twitter or Booking.com) about reviews and ratings on different airliners and different destinations.
2. The system must be able to make sure the data is fair, filtered and validated.
3. The system must be able to store the data no matter it's form/schema.
4. The system must be able to run analytics on the received data to calculate ratings on/over different moments/ranges in time (f.e. days, weeks, years).
5. The system must be able to disperse reports on the analyzed data to Business to Business clients (such as KLM) based on report configuration in the B2B application
6. Initially the reports don't have to be delivered as a PDF file but served as a webpage or client application. However the system should support this to be added later.
7. The system must be able to render ratings and reviews about airliner A...Z and destination A…Z to consumer clients based on their input
8. The system should allow airliners in consenting to exchange information about themselves to each other, creating a fair information sharing system.
9. Business to Business clients need to be able to respond to posts done by Business to Consumer users via the FlyWithUs application

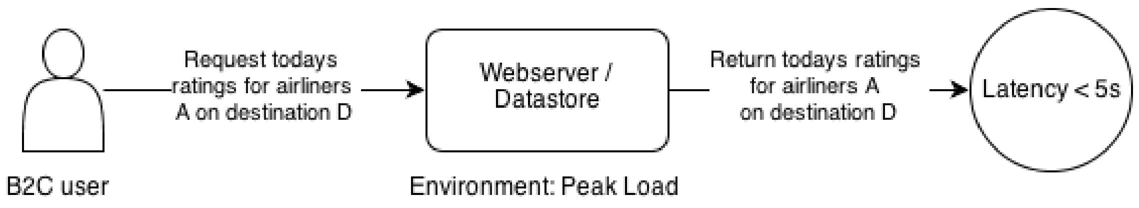
4. Quality attributes

Aside from the functional requirements stated in section 3, there are multiple quality attributes the stakeholders are concerned with.

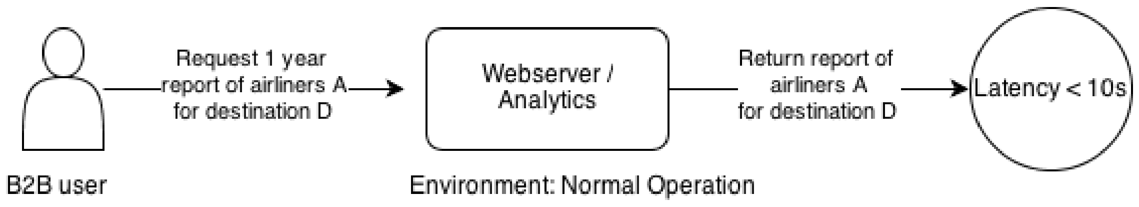
No	**Quality Attribute**	**Details/Refinements**	**Stakeholder(s)**
Q1	**Modifiability**	1. Adding a new datasource to the system should take no longer than approx 80 Working Hours 2. Modifying a client-interface to the system should be done without affecting the other components.	**S1, S4**
Q2	**Performance**	1. The latency for a request done from a B2C client application must be under 5 seconds. 2. The latency for a request done from a B2B client application must be under 10 seconds.	**S2, S4**
Q3	**Scalability**	If the system needs to handle more requests from B2C or B2B than it can handle in its current state, it needs to scale out in order the stay within the latency times defined in the **performance** requirements.	**S1, S4**
Q4	**Data Quality**	The data needs to be validated and verified in order for it to be useful. The data must conform with below conditions before being used: * data is not outdated * data is complete and not empty * data is not discriminating or sexually explicit * data posted with intention is filtered	**S1, S2**
Q5	**Privacy/Security**	1. The system should be able to authenticate actors when provided with username and password 2. The system should be able to allow actors to use certain functionality based on actor roles	**S2**

5. Scenarios

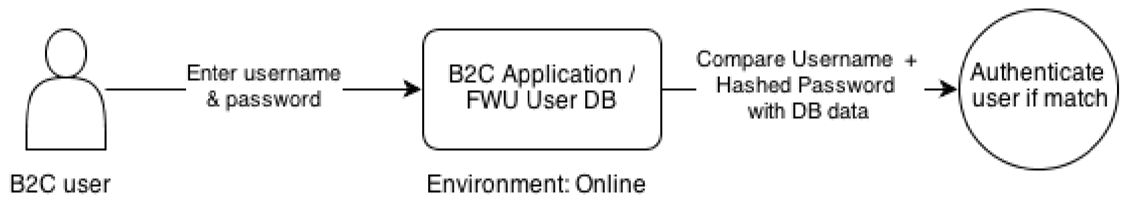
S1


B2C user
Request todays ratings for airliners A on destination D
Webserver / Datastore
Environment: Peak Load
Return todays ratings for airliners A on destination D
Latency < 5s

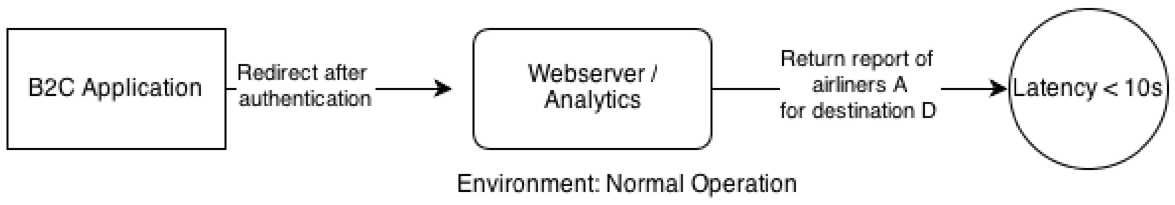
S2


B2B user
Request 1 year report of airliners A for destination D
Webserver / Analytics
Environment: Normal Operation
Return report of airliners A for destination D
Latency < 10s

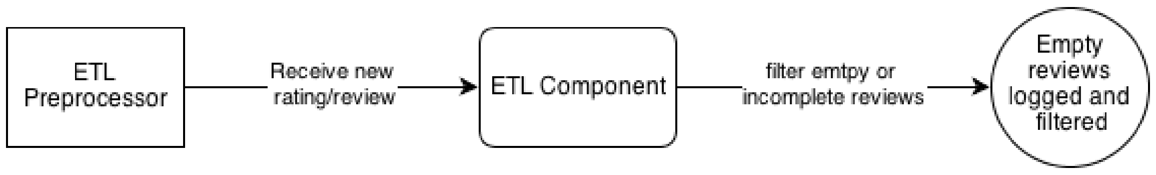
S5


B2C user
Enter username & password
B2C Application / FWU User DB
Environment: Online
Compare Username + Hashed Password with DB data
Authenticate user if match

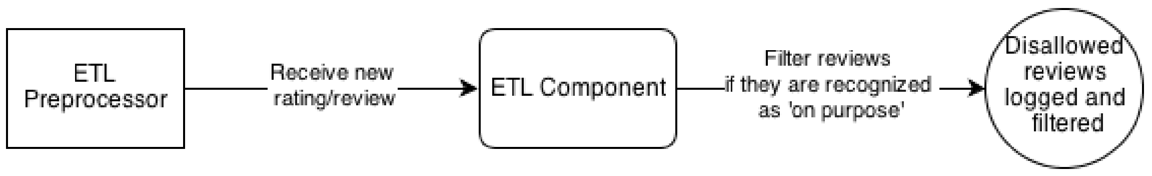
S6


B2C Application
Redirect after authentication
Webserver / Analytics
Environment: Normal Operation
Return report of airliners A for destination D
Latency < 10s

S3


ETL Preprocessor
Receive new rating/review
ETL Component
filter emtpy or incomplete reviews
Empty reviews logged and filtered

S4


ETL Preprocessor
Receive new rating/review
ETL Component
Filter reviews if they are recognized as 'on purpose'
Disallowed reviews logged and filtered

Appendix B - Design decisions

The decision tables below show our motivation of our decisions taken to design our architecture.

B2B and B2C decoupling system

Issue	Easy scaling and management of the B2B and B2C platforms cannot easily achieved if both systems are coupled.
Decision	We decided to split B2B and B2C as individual systems in our architecture.
Status	Decided
Group	Deployment
Assumptions	B2C systems receive a lot more traffic than B2B B2B systems receive regular updates B2B work on other API layer than B2C B2B works independent from B2C
Constraints	None
Positions	Implement B2B and B2C systems as VMs Integrate B2B software in B2C system.
Argument	Splitting the systems for B2B and B2C created atomic systems which are easier to deploy and scale. The B2C software and API is not the same as the B2B BI software. Creating a physical separation also improves scalability and manageability in case of system upgrades and maintenance.
Implications	Deployment of specific toolset for B2B or B2C will be faster. (isolated and dedicated purpose towards 1 system) Functional API tailored to its system, improve security. This setup will introduce an additional management procedure for sysadmins. management of 2 different systems requires more man-hours. High availability for both systems, might 1 system experience downtime, the other is not affected.
Related decisions	Integration of API into web server node.
Related requirements	Scalability, performance B2C/B2B
Related artifacts	None

Related principles	Load balancing (multiple B2C nodes)
Notes	None

Table B.1 Design decision for implementing Separate systems for B2B and B2C

Integration of API layer in B2B and B2C system

Issue	The API layer as a subsystem will have a high risk being overloaded and becoming a single point of failure.
Decision	The API can be installed in each (web)server node, to guarantee access to the datastore.
Status	Decided
Group	Deployment, integration, performance
Assumptions	B2C/B2B web layer communicates with API API layer works independent (from other nodes) API is the only layer providing access to the datastore
Constraints	B2B and B2C API layers facilitate other functionalities
Positions	Set up as dedicated API system with it’s own cache.
Argument	Having only one API system, the architecture on the presentation layer is very sensitive to high load (e.g. high amount of webrequest from B2C). This would introduce a change of having a single point of failure. By integrating the API layer into the webservernodes, this responsibility is moved to the node. Scalability comes with this new approach, caching and so is provided by the webserver node. Secondly, without the dedicated API system. The architecture simplifies and requires less servers to run.
Implications	During deployment of the web server nodes, the API layer should be included. Upgrades of the API layer is not on one location, but on all web server nodes. Web Server Nodes require higher technical assets (e.g. more memory), links from web server nodes to the datastore will transfer more (raw) traffic.
Related decisions	B2B and B2C decoupling
Related requirements	Scalability, performance B2C/B2B
Related artifacts	None

Notes	None

Table B.2 Design decision for implementing Integration of API layer into the B2B and B2C systems

Integration of analytics system into datastore

Issue	Performing statistical analytics will require huge amount of data to be transferred between computing node and datastore.
Decision	The computation will be run on the datastore machines.
Status	Decided
Group	Performance
Assumptions	Statistical analysis using the OLAP systems require lots of data querying. Analyses access the datastore often Analyses write results back to datastore Analyses will run often (initiated realtime by B2B clients)
Constraints	Choice of analytics system is limited to the choice of data store type (SQL, noSQL etc.), not all can run on the same system (OS differences) There are a few of backend independent analytics systems. (icCube, SSAS)
Positions	Separate OLAP system located next to the datastore for optimal data transfer (low latency) Partition datastore for analytics Design efficient queries to datastore. Optional Integration into web server nodes through the API for lightweight queries.
Argument	Allocating computation nodes for the OLAP system on remote sites from the data storage will increase latency on the analytics processing. Placing the analytics software in the same system as the datastore, minimizes the latency caused by datatransfer. Analytics are done both periodically and real time. For the real time analyses, this will improve the usability.
Implications	Integrating the analysis system into the datastore adds more computing power (thus computer systems) to the datastore.
Related decisions	None
Related requirements	Performance, GreenIT

Related principles	Reuse infrastructure, minimize overhead.
Related artifacts	None
Notes	None

Table B.3 Design decision for implementing Intergration of analytics into datastore.

Alternative approaches for implementing the ETL

Issue	We need to be able to extract the data from different sources and structure this data for further use. However, we need to address the concerns of the stakeholders, which comprise of low energy consumption, scalability and avoiding vendor lock-in. We are considering our options for the type of application (distributed/non-distributed).
Decision	We decided to create a non-distributed application to extract the data and structure it for further use.
Status	Decided
Assumptions	We have assumed that the implementation time of all three possible systems is about the same. Also we have assumed between 4 and 5 GB of data per year.
Constraints	It will prevent the application to scale on the intake end, which means that a dramatically higher number of reviews would choke the system.
Positions	We have considered options like Hadoop and Storm. Also we have considered to build our own distributed system or our own non-distributed system.
Argument outline	We have chosen this option as it is the easiest to implement, it avoids vendor lock-in, lowers power consumption and still gives the performance we need.
Related decisions	Is our input data really big data? No, it is not that big, only 4 to gb a year. Do we want to separate the ETL from the datastore? Yes, to make it easier to add new sources.
Related requirements	Table B.5
Related artifacts	None
Related principles	None

Table B.4 Design decision for implementing ETL

Will the system..	Hadoop	Distributed	Non-distributed
be able to handle the expected amount of data?	Y	Y	Y
be green?	N	N	Y
be able to return the requested data in time?	Y	Y	Y
be able to scale?	Y	Y	N
difficult to scale?	N	Y	Y
avoid vendor lock-in?	N	N	Y

Table B.5 Related requirements: Design decision for implementing ETL

Other decisions

ETL adapters (modifiability)

We have chosen to create adapters in the ETL for each of the data-sources. This way a new data-source can be added with a minimum impact on the system. These adapters will create a normalized review to be inserted into the database.

ETL filtering (data quality)

We have decided to create a filter in the ETL layer of the application. This way we can correct for incomplete data, data purposely influencing the system or explicit content. By doing this in the ETL, these unsuitable reviews are not persisted.

Separate users table (security/privacy)

To implement a secure user management system, we have chosen to separate the users table from the rest of the datastore. This will also allow us to host the rest of the datastore in another country, which will lower the costs of the hosting.

Relational DB for structured analysis data, non-relational for unstructured review data

After processing the data into ratings and analysis data for the reports, it becomes structured data which is best suited by a relational database. With good database design, the relational database will have better performance than the non-relational database. This database will be separate from the larger data sets which due to lack of structure can not be stored in a relational database. Therefore this data will be stored in a non-relational (document-oriented) database.

Non-distributed processing (business view)

Two contradicting stakeholder concerns are performance and green IT. After carefully estimating the data that will be gathered and taking into account the batch like processing that is allowed, we decided that performance is not that big an issue. Therefore we have chosen a non-distributed type of processing. This will use less energy than a distributed system, which addresses the green IT concern, while not creating an issue with the performance.

Removal of “general” API layer

Different to our last design, we have removed the general API layer. This layer would create a single point of failure, which we would like to avoid. Therefore we have added a caching method to the B2B and B2C side. This will also make sure that only relevant data is cached (i.e. European review data in Europe).

References

[1] Twitter Analytics, www.topsy.com

[2] http://www.routesonline.com/news/29/breaking-news/206883/
statistics-spotlight-top-50-airlines-july-2013/

[3] Airline review site, www.flightscore.com

[4] Paul Clements & Len Bass, *The Business Goals Viewpoint*

[5] Database comparison
http://www.cbsolution.net/techniques/ontarget/databases_relational_vs_object_vs

[6] Yahoo Hadoop tutorial: http://developer.yahoo.com/hadoop/tutorial/module1.html

[7] MongoDB product site http://www.mongodb.org/

[8] PostgreSQL product site http://www.postgresql.org/

[9] NoSQL overview http://newtech.about.com/od/databasemanagement/a/Nosql.htm