CHALMERS

Development of Picwear

A Multi-Platform Mobile Clothing and Fashion Service for Android, iOS and the Web

Bachelor of Science Thesis in Software Engineering

SIMON BENGTSSON
PETER ELIASSON
CHRISTOFFER HENRIKSSON
ANTON JANSSON
JOHAN MAGNUSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2015

Development of Picwear
A Multi-Platform Mobile Clothing and Fashion Service for Android, iOS and the Web

S. BENGTSSON
P. ELIASSON
C. HENRIKSSON
A. JANSSON
J. MAGNUSSON

Examiner: J. SKANSHOLM

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: The logotype of the Picwear organization.

Department of Computer Science and Engineering
Göteborg, Sweden June 2015

Abstract

The purpose of this report is to describe the development of Picwear, a mobile service, in collaboration with a startup with the same name. The service offers consumers to find fashion from local clothing stores, using applications for Android and iOS. The report focuses on the development of native applications for the two mentioned platforms, a web portal for stores to upload pictures of their products, and an accompanying backend that connects these services. Particularly, the usage of a mobile backend as a service (MBaaS) is covered, along with how such a service can simplify the development of mobile applications. The report also discusses NoSQL databases, which often are part of MBaaS solutions, and how they with advantage can be used when building scalable services.

Sammanfattning

Denna rapport syftar till att beskriva utvecklingen av mobiltjänsten Picwear i samarbete med ett nystartat företag med samma namn. Tjänsten låter konsumenter hitta kläder och mode från lokala butiker med hjälp av applikationer för Android och iOS. Rapporten fokuserar på utveckling av plattformsspecifika mobilapplikationer för de två nämnda plattformarna, webbportal för butiker att ladda upp bilder på deras produkter, samt tillhörande backend för sammankoppling av dessa tjänster. I synnerhet behandlas användandet av en mobil backend-tjänst (MBaaS) och hur en sådan tjänst kan förenkla arbetet vid framtagning av mobilapplikationer. Rapporten diskuterar även NoSQL-databaser, då dessa ofta ingår i MBaaS-lösningar, samt hur de med fördel kan användas för att bygga skalbara tjänster.

Vocabulary

API	Application Programming Interface, protocol for software endpoints (for example a client and a server) to communicate
IDE	Integrated Development Environment, a program that assists with software development
JSON	JavaScript Object Notation, lightweight data format
MBaaS	Mobile Backend as a Service
REST	Representational State Transfer, set of best practices for designing the communication between network applications
SDK	Software Development Kit, frameworks and libraries for software development
Startup	A newly established company
UML	Unified Modeling Language, language used for general purpose modeling
XML	Extensible Markup Language, data format that can represent a wide variety of data

Contents

1 Introduction

The internet has introduced many new ways for consumers to find and explore the latest in fashion. There are for example many popular fashion blogs and big online clothing stores with a vast number of products. Online services can be accessed from any location and at all times, but despite those benefits, less than 10 percent of purchases in the Swedish fashion industry took place online in 2013 [1]. This suggests that people still prefer buying clothes in physical stores, with advantages such as being able to try on clothes before buying them.

Picwear is a startup that wants to provide consumers with some of the benefits of online shopping, while still enabling them to try and buy clothes in physical stores. It is run by three people from Chalmers School of Entrepreneurship, and they intend to create an online service where fashion retailers upload photos and metadata of their products, which people then can browse and interact with. Retailers will benefit from the service by getting visibility of their store and products, and Picwear's business model is that retailers are prepared to pay for this [2]. When this project started, in the beginning of 2015, they had initiated discussions with clothing stores in Gothenburg, and many of them had expressed an interest in the service.

The service Picwear wants to create is intended to have two major parts, one for fashion retailers and one for consumers. The vision for the retailer part is simple and basically only includes features related to uploading pictures and metadata of their products. For consumers, the vision is more extensive and includes features to follow clothing from specific stores, discover new clothing and search for specific products. The vision also includes features for sharing products and viewing information about stores.

1.1 Purpose

The purpose of this project is to develop the software required for the Picwear service. It will include building user applications for Android and iOS, a separate store admin application and an accompanying backend. To fulfill this purpose, several technical issues need to be researched. One of these is how mobile applications targeting more than one platform can be developed with focus on good user experience and minimized development time. Another issue is how a backend can be created with great performance and security in mind, while enabling a service to increase its user base quickly.

1.2 Scope and Limitations

The scope of the project does not include designing user interfaces, conducting user tests or developing the vision of the service, as these parts will be covered by Picwear.

However, the project team will participate in meetings regarding these parts and will be able to influence decisions.

This report will mostly focus on more general issues rather than specific features of the Picwear service. The reason is mainly to make the discussions held in the report applicable to a wider range of projects. Furthermore, the project members have signed a contract and a non-disclosure agreement (NDA) with Picwear. These states that Picwear owns all software developed in the project and that the team is not allowed to talk about certain parts of the Picwear service.

2 Method

The first part of this chapter will cover how the project team worked in collaboration with Picwear. The following sections will introduce how the project team worked together and the technical tools that were used to develop the software for the Picwear service. Many of the tools mentioned in this chapter will be covered in detail in chapter 3 Technical Background.

2.1 Collaboration with Picwear

The project was developed in close collaboration with Picwear, and most of the work was conducted at Picwear's office. Instead of having formal meetings at specified times, informal meetings were held when needed. As soon as new ideas or questions arose, they were shared and discussed with the relevant people. The vision and features of the Picwear service were constantly changed during these meetings. The initial functional requirements written by Picwear can be found in Appendix C Functionality Description.

2.2 Development Process

No specific software development methodology was chosen for this project, but inspiration was taken from for instance Kanban [3] and Scrum [4]. Picwear was initially asked to add features to a prioritized list. These features were then organized into five milestones by the project team, each scheduled to take three to five weeks. All milestones, including the features they contained, can be found in Appendix A Development Time Planning. The last milestone contained features to be implemented outside the scope of this project. The idea was to implement features in a milestone for all platforms in parallel. Two developers were assigned to work on the Android client, two on the iOS client and one on the backend and the store admin application.

2.3 Development Tools

Many platform specific tools were used in the project, which can be seen in Table 1. When developing the iOS and Android clients, tools for these platforms were chosen based on recommendations in each platform's developer guidelines [5, 6]. The backend has been developed with tools provided by a mobile backend as a service called Parse. The store admin client has also been influenced by Parse, since it too is hosted on the Parse platform.

	Android	*iOS*	*Store Admin*	*Backend*
Language	Java/XML	Swift/XML	HTML5	JavaScript
Framework	Android SDK	Cocoa Touch	Node.js/Express	Node.js/Parse
IDE	Android Studio, IntelliJ	Xcode	WebStorm	WebStorm

Table 1. *An overview of the tools used in the development of the Picwear service.*

Code for the backend and the different clients were mostly developed independently, but certain tools were used for all parts of the project. One of these tools is a collection of libraries provided by Parse used to simplify the communication between clients and the backend. Furthermore, all code and related files have been hosted on Bitbucket [7] and managed with the version control system Git.

2.4 Project Evaluation

To evaluate the work in the end of the project, Picwear was asked to complete a survey with questions about the project process and results. The survey included open questions asking for their expectations, reflections and general comments about the project. They were also asked to grade their level of satisfaction in a few areas. The survey questions were inspired by an evaluation form from the consultancy firm Chalmers Teknologkonsulter AB [8]. The full evaluation form, together with Picwear's answers, can be found in Appendix B Project Evaluation by Picwear.

3 Technical Background

This chapter will cover background information on technical aspects of the project that will be discussed later in the report. Specifically, it will introduce important concepts regarding Android and iOS development, providers of mobile backend services, alternatives for developing multi-platform applications, as well as database storage focused on NoSQL.

3.1 Android Development

Android development is primarily done in the Java programming language. It is not tied to a specific environment. However, there is an official IDE called Android Studio [9], and using it is recommended. Even though the Android platform has much in common with the default Java platform, there are still aspects where they differ. Some of the more important differences are presented below.

3.1.1 Activities and Fragments

An Android activity is much like a window in a traditional computer operating system in the sense that it is the primary user interface component of the application. However, as opposed to windows, only a single activity is displayed on the screen at a time. Most Android applications consist of multiple activities, which are switched between depending on the user's interactions. Each activity should be responsible for one specific part of the application, like the screen a user sees when the application is started or a login screen. Switching between activities is done by passing an intent to the Android system, which in turn activates the appropriate activity and provides it with any data included in the original intent.

To make sharing similar layouts easier and to ease development of applications running on both smaller phones and larger tablets, fragments were introduced. A fragment is, as the name implies, a smaller component of a larger layout and is therefore often combined with other parts to create a complete layout for an activity.

3.1.2 Lifecycle

An important part of developing for Android is managing the lifecycle of the application. As a user navigates through the application, the Android system manages said lifecycle by notifying the different components, such as activities and fragments, of changes to their state. Currently displayed components are in a state called resumed, while other components are either paused, stopped or destroyed, depending on internal factors of the Android system, such as available memory [10].

As a component is killed by the Android system, its current instance state (which contains. instance variables and other data used by the activity, not to be confused with the different

states of a component) is also destroyed and removed from memory. However, before this happens, the component will be provided a special type of data structure, called a bundle, to store any current instance state. If the component later again becomes active, it will be provided the same stored data [10].

Notably, not all types of data can be stored in a bundle. To directly store a more complex object, it must implement a specific interface, either Serializable or Parcelable, where the latter is specific to Android and requires the serialization logic to be manually implemented [11]. However, both said interfaces require converting the object into a binary representation, which might prove problematic. Other solutions include serializing the object to text, by for example using one of the many JSON libraries available [12], or storing the object in the local database.

3.2 iOS Development

The core concepts and techniques for making iOS applications will be presented below, including how the user interfaces for such applications are created and what programming languages that can be used for development.

3.2.1 Objective-C and Swift

Until 2014, Objective-C was the only programming language available for creating mobile applications for iOS [13]. Objective-C is based on the programming language C, but with additional features such as support for object-oriented programming [14]. In 2014, Apple released the first version of the programming language Swift as an alternative to Objective-C. Swift uses the same runtime as Objective-C, which means developers can use both these languages together in a single project [15].

Swift comes with some new ideas and some features from functional programming languages such as Haskell [16, 17]. Something that differentiates Swift from other languages is that all variables have to reference values and cannot be set to empty values such as null in Java or nil in Objective-C. To replace the need for this, Optionals were introduced. They are wrappers around variables and can be set to nil, and the idea is that this will reduce some common software development errors. Something else that differentiates Swift is that it has tuples, which are collections of different values. A tuple can contain multiple object types and can be returned from a function as a single object without making a custom class for that type of data structure.

3.2.2 Storyboard, .nib and .xib Files

Xcode provides a drag-and-drop editor, Interface Builder, for designing graphical user interfaces (GUIs) of iOS applications. The layout of the GUI and the navigation flow between view controllers is drawn and visualized in a *storyboard* (see Figure 1), which is

an XML-formatted file containing all the views and their properties. The file consists of a header with version numbers describing how the rest of the file is formatted and how it should be interpreted.

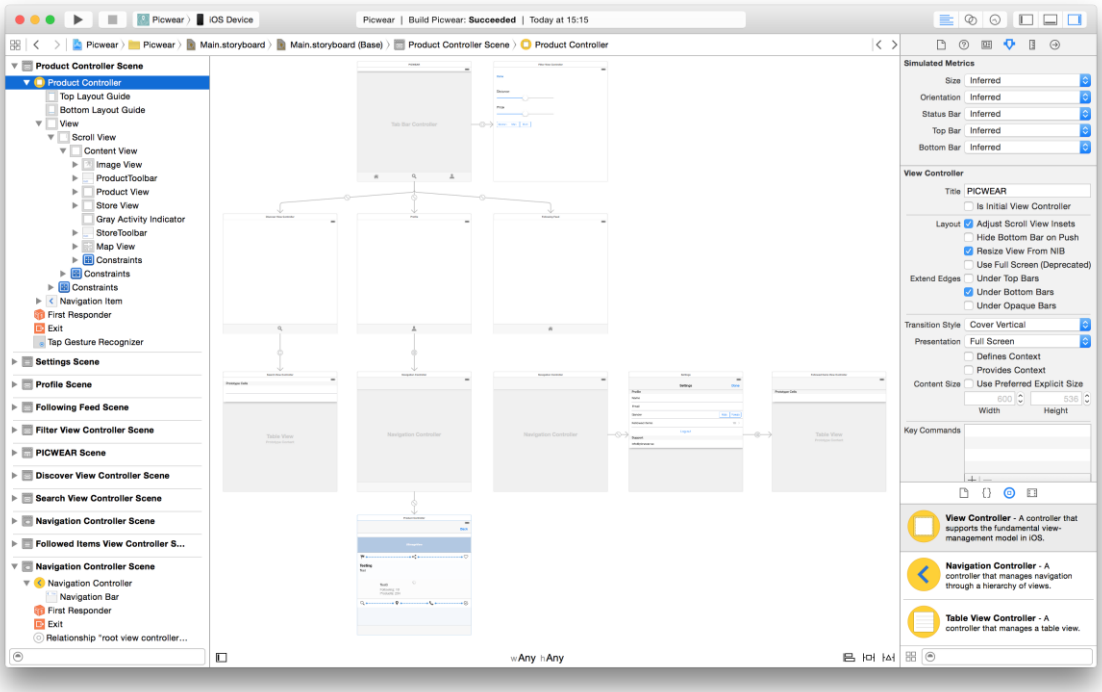


Figure 1. *A screenshot of a storyboard in Xcode with a graphical representation of view controllers in the center of the screen.*

Instead of designing all views and subviews in a single file, views and view controllers can be designed in separate .nib or .xib files. These files can later on be used in a storyboard. An application can also have several storyboards, which can ease development when designing large universal applications for both iPhone and iPad.

The linking between the GUI and the code is also done by drag-and-dropping. For example, a button action can be created by dragging a button into the code, inserting a connection between the button and the method handling the action when, for example, the button is tapped. This drag-and-drop procedure is also used to insert an outlet (equivalent to Java's instance variables) in order to modify properties of a view object. A reference to the view is then created in the corresponding view file and can be used in code after the view is loaded and shown on the screen. Layout and design of views and view controllers as well as transitions between them can also be written in code, but the recommended way is to use storyboards [18].

3.3 Node.js

Node.js is a JavaScript runtime environment running applications outside of a browser. It is designed to be used for servers focused on network communication and has good

concurrency performance as a result of the absence of blocking threads. Therefore, for a web application with lots of requests, such as a web API service, Node.js often satisfies its requirements.

3.4 Asynchronous Programming

Most platforms today provide ways to perform multiple tasks in parallel. Oftentimes, this is performed via the use of threading. However, how a platform performs such tasks can often be hidden by ways of API design, leaving the consumer with a high-level abstraction of a background job. Such abstractions, as well as to why the ability to perform asynchronous tasks is necessary, are discussed in this section.

3.4.1 The UI Thread Pattern

A common pattern for applications having some type of user interface is to designate a single thread for handling all interactions with said user interface. By having all interactions with the user interface performed on a single thread, one avoids the generally troublesome synchronizing that would otherwise be required. Both Android and iOS are built on this pattern. In turn, this pattern also necessitates the usage of non-blocking asynchronous tasks, as if some operation blocks the designated UI thread, the entire application will, from the perspective of the user, become unresponsive [19, 20].

3.4.2 Callbacks

One abstraction that is used on both Android and iOS, as well as in JavaScript, is what is known as callbacks. An API designed using callbacks means that when the user of the API wants to perform a background job, the user also provides the code that should be executed once the job completes. In JavaScript, this code is often provided as an anonymous function, while in Java, such code is typically enclosed in an anonymous class (similar to the strategy design pattern [21]). To assist the developer, such callbacks are often invoked on the UI thread. An example could be to fetch information and, once the data is available, display it to the user. Presenting the data would then be done in the callback [21].

3.4.3 Promises

Another approach to handling the execution of tasks once some background work has been performed, is using so-called promises. A promise is an object returned instead of a value, representing the task of obtaining this value. One can then attach listeners to this promise that will be called once the promise is completed either successfully or with an error [22]. Some promise implementations, like the one used in Parse's SDKs for Android and iOS, also has additional states that can be reacted to, such as the task being cancelled [23, 24].

3.5 Mobile Backend as a Service

Mobile Backend as a Service, MBaaS for short, is a quite recent development intended to ease the development of mobile application backends. An MBaaS provider enables users of their service to use features that would previously require a custom-made backend. These features are generally provided as part of an SDK for various mobile platforms. They often include access to some database for storing and querying data, as well as more application specific features such as mobile push notifications and user management. Providers of such services include, among others, Microsoft Azure [25], Firebase [26], Parse [27] and Kinvey [28].

3.5.1 Parse

The Parse MBaaS, owned by Facebook, includes all features described above and uses a popular NoSQL database named MongoDB for its data storage. It provides SDKs for multiple platforms – including Android, iOS and JavaScript – as well as a REST API web service for accessing data [29].

Backend code, in Parse called *Cloud code*, can be published to servers hosted by Parse. This code can be written to define custom API functions for common logic in the application or listen for save and delete events on objects, which can be used to validate incoming data or perform actions when data is changed. Cloud code uses Node.js for both writing API functions and website hosting, which is another feature provided by Parse.

Data in the Parse database is grouped by classes, where each class holds objects of the same type. There are a few pre-defined classes such as User, Session and Role, which already include some basic properties and implementations. For instance, Parse has built-in functionality for user authentication integrations to Facebook and Twitter. Additional classes can be created manually, and data types for properties in those classes can be restricted via an administration console.

3.5.2 Security Using Access Control

Different MBaaS providers have implemented different ways for restricting access to the database and the stored data. To restrict the scope of this section, only methods available for use in Parse will be discussed. However, these are likely to exist in similar shapes also for other MBaaS providers.

There are two methods to secure access to data stored in the Parse database. The first and simplest one is called class-level permissions (CLP) and is used to allow or restrict access for all users to a single database table, or *class* as it is called by Parse. The method controls both authenticated and public users. If all CLP permissions are denied, the only way to modify objects of this class is via the master key – a secret key providing unrestricted

access to all data in the database. Different permissions can be set for reading and writing, or even creating, modifying and deleting [30].

The second method to enable security as well as privacy is called access control lists (ACL). ACL is not Parse exclusive and is commonly used in different types of software [31]. Instead of allowing and restricting access to classes, the permissions are set to specific rows inside classes. The permissions determine which application users that should be given access. Users not explicitly granted permission cannot reach the data. As CLP, ACL can also have reading and writing permissions set independently.

ACL permissions can be set for both users and roles. A role contains a group of users that all have the same permissions and can be placed in a hierarchy inheriting permissions of ancestors [30]. The concept is called role-based access control and simplifies sharing content with groups of people, for instance friends of a user, or allowing only administrators to edit information.

3.6 Multiplatform Web Based Applications

An alternative to native development – development using the tools and APIs provided by the platform – is to create an application based on web technologies. This can be achieved in multiple ways, and with each different technology, the possibilities and limitations vary.

Using a WebView is one such alternative. A WebView is a UI component available on both Android and iOS that allows the developer to embed a website inside of the context of an application. This technique can then be used to create a single website that is later displayed as if it was a native application. However, it is not possible for a website rendered in a WebView to interact with the underlying platform in the same way as a fully native application would [32, 33].

A second example is using one of the many frameworks for mobile web application development, such as Apache Cordova or PhoneGap. These frameworks generally make additional native features available to the web application, making them able to more closely mimic the functionality of a truly native application. Furthermore, most of these frameworks provide additional abstraction of the underlying platform, making it possible to share most, if not all, code between multiple platforms.

There are also additional frameworks built on top of the ones mentioned earlier. One such framework, called Ionic, uses Apache Cordova together with a JavaScript library called AngularJS [34] to further increase productivity.

3.7 Database persistence

Since the development of relational databases started in the 1970s, they have been the default choice for storing data in applications [35]. This type of data storage is efficient for data where entities have relationships between each other and allows strict constraints for schematic data. These constraints include specifying data types, having unique and foreign keys as well as custom validity checks, to give a few examples.

When an operation is performed on data in a relational database, it can be run as a transaction to ensure consistency and to make sure that all relationships between the modified tables remain valid. However, this requires that all of the affected tables are locked during the whole transaction to ensure that no other operation modifies the same data, which would have made the consistency state invalid. Most often, a relational database is run on a single server, and therefore, there are no problems running these transactions. However, while expanding a relational database over multiple servers, the complexity of ensuring consistency, using transactions, increases.

When a relational database needs to be scaled, it is therefore most often scaled vertically, meaning that the machine running the database gets equipped with more powerful hardware [36]. Horizontal scaling can instead spread data across multiple machines, and thus additional servers can be added when in need of more performance. It is possible to run a relational database on multiple servers, but using other types of databases can improve the ease of maintenance and installation. A set of database types this applies to is described below and is covered by the term NoSQL.

3.7.1 Overview of NoSQL

The usage of NoSQL databases as data stores in applications has since the early 2000s been rising in popularity [35]. Compared to relational databases, many NoSQL databases are built to run well on clusters and are therefore easier to scale for large volumes of data. The data is typically schemaless or uses dynamic schemas, which allows the application to store both structured and unstructured data.

Various types of NoSQL databases differ widely in functionality, but one point that differentiates them in general from relational databases is relational consistency and transactions. A NoSQL database cannot ensure that a referenced object actually exists, which means that a reference can be invalid. These constraints are instead left for the application to implement. The advantage of not supporting this type of consistency is that different tables of data can be distributed over multiple servers, because one server does not need to communicate with others to ensure the consistency. This concept of data distribution among multiple nodes is called sharding.

NoSQL databases use transactions, but not in the same way as relational databases. In a relational database, multiple operations can be placed into a transaction manually, which all will be rolled back in the case of an error. A transaction in a NoSQL database covers only one operation, which can be compared to cover only one insertion or update of a table row in a relational database. Therefore, the application layer must perform the validation of rows. An example of this is if any property other than the key should also be unique.

There are four dominant types of NoSQL databases: key-value stores, document databases, wide-column stores (also known as column-family stores) and graph stores. The key-value store is the simplest of them and can be compared to a hash table representing a map of values each attached to a key. The value does not have to abide to any type constraints and normally it is a string of JSON, XML, a text or a binary object, depending on the database implementation. Two popular key-value databases are Redis and Memcached [37].

Document stores are the most used type of NoSQL databases, where MongoDB and CouchDB top the list [37]. The document stores have more similarities with relational databases than key-value stores have, because a document store enables queries that are more advanced and relationships between values, called documents. In contrast to key-value stores, applications can query on fields within the document, and not only query the primary key. In other words, this means that a two-way relationship, which in a key-value store has to include references from both ends, only needs to be stored in one of the ends. This can be related to the foreign key in a relational database.

A wide-column store enables more structure than the document store, because it has multiple key levels. Each row contains a set of key-value pairs, or columns, which can be fetched by direct lookup using the combined row and column keys. Cassandra and HBase are examples of this type of database and generally, they are fast and optimized for running computations in clusters [38].

Graph stores differ from the other three described NoSQL databases, because the data is stored in a graph instead of in a map structure. This type is used to model networks and to efficiently query on relationships between objects. An example would be to in a social graph query friends of friends, or finding the shortest path between two nodes. Graph stores do not scale as well as the other types and are often run on a single machine [35].

Although graph stores can be good for designing social networks, from here on in the report, document stores will be the main focus while discussing NoSQL databases.

3.7.2 Comparing Aggregation and References

There are two ways to model relationships in a document store, either by aggregation or by using references. Aggregated relationships include all data about a related document inside a parent document. In the Unified Modeling Language (UML), this relationship is called a composition and is represented by a black, filled, diamond. This type of relationship should normally be used for aggregated entities that only exist if their parent exists.

Another scenario where aggregation is preferable is if data should always be included in a read, without having to do a second call in order to fetch it. However, if the same data is aggregated in many documents, some consistency issues may arise, just like in any other database with unwanted redundancy. In this case, the same object can have different properties depending on which document it is fetched from [39].

Using references instead of aggregations can solve consistency issues. A reference is holding an ID of the object to be referenced, acting as a pointer. To fetch both the referencing and referenced object, two queries are necessary, which may decrease the overall performance. An advantage of using references, apart from reducing redundancy, is that the table containing the referenced objects can be queried independently.

3.7.3 Modeling Tree Structures

While discussing aggregations and relationships, one interesting structure to model is the hierarchical tree. This structure, where each node has one or zero parent nodes, can be modeled using either aggregations or relationships. In an aggregated tree model, all the root nodes contain all their children themselves, and to find a specific node, each node has to be searched for recursively.

A more common approach is to store all tree nodes as separate rows, or documents, in a database. This method applies for both NoSQL and relational databases. In order to represent the hierarchy, there are a few alternatives to choose from, all using references for the relationships [40].

The simplest model is for each node to store a list of references to its children, or to store a reference to its parent. Only storing the parent makes sure that each node only has one parent, which can be compared to nodes having a list of its children where one node can be referenced from multiple nodes.

Some database management systems support queries that are recursive [41]. Recursive queries can be used to perform operations on subtrees in a tree structure. There are other more streamlined approaches, though. One such approach is to store an array of ancestors for each node [40]. Then, all descendants for a specific node can easily be queried by finding the nodes that contain the parent node ID in the ancestor array.

The *materialized paths* pattern is another method similar to using the *array of ancestors*. Instead of an array, the ancestors of a node are concatenated in a string delimited by some specific character, for instance a comma. Because the materialized path field is a string, nodes can be found using regular expressions. Compared to *array of ancestors*, the materialized paths pattern is more efficient in the aspect of performance and brings more flexibility in querying. As an example, it is possible to create a query to search for all nodes that have a grandparent with a specific name or ID [42].

3.8 Git

Git is a version control system for managing code and related files in a development team. It is based on a tree structure with nodes, so called commits, representing a state of the project file system. By using multiple branches – sequences of commits diverging from a base commit – one can develop multiple features in parallel and then merge each individual feature, as they are completed [43].

3.9 Agile Software Development

Agile software development methods has recently seen a rise in popularity at the expense of more traditional sequential methods [3]. This can mostly be attributed to that they make it easier to handle changing requirements.

3.9.1 Scrum and Kanban

Two of the most common agile software development methods are Scrum and Kanban [4]. One distinct feature of Scrum is that work is divided into *sprints*. A sprint contains tasks and has a fixed deadline at which a review is conducted about the overall progress of the project. Kanban is not as strictly specified as Scrum in terms of how a project should be organized [3]. The main feature of Kanban is that work should be organized in a prioritized list that can be updated continuously. When an item on the list is completed, work on the subsequent item is started.

4 Overview of the Developed Service

This chapter describes all parts of the final service and how they work together. The service can be divided into four major parts, also represented in Figure 2: backend, user applications, sharing website and store admin application. The backend is the central component that connects all client applications with the data from the stores. The user applications component contains the two mobile applications for Android and iOS, which are the only parts of the service that the end users actually will use. A simple website was developed for sharing of links to products. Finally, the store admin application is used by stores to upload new content for the service. All components are described in the subsections of this chapter.

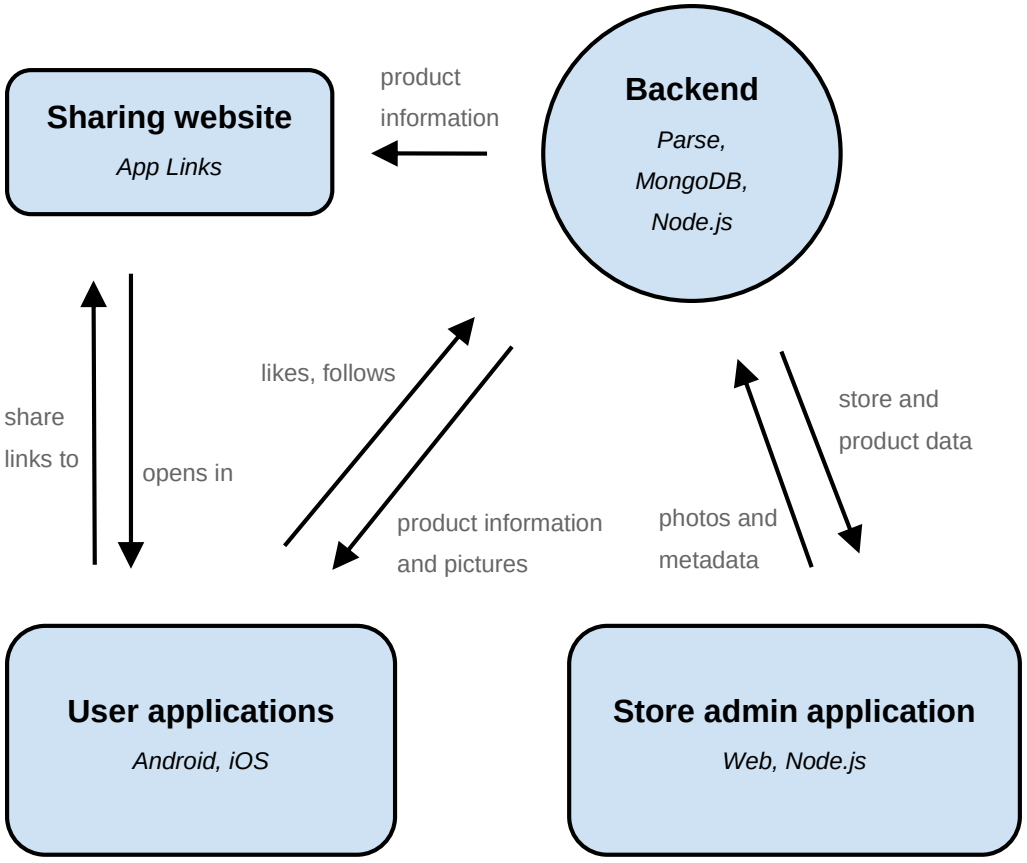


***Figure 2**. Overview of the four components of the service and their interactions.*

4.1 Backend

As mentioned earlier, the backend is built using the MBaaS Parse. The backend is a REST API web service [44] for the client applications with functions to, for instance, retrieve product information and feeds, like products and follow search terms. These functions are implemented using custom Cloud code functions in Node.js using the Parse

JavaScript SDK. As visualized in Figure 2, the backend also gets information from the store admin application, which it saves in the Parse database.

The backend service uses *method hooks* for saving and deleting objects available in Parse. These hooks run whenever an object of a specified class is saved or deleted respectively, and their methods are used to validate data before any action is performed or to do computations or cleanups. An example in the service is that additional user data is fetched from Facebook when a user is successfully signed up.

4.2 User Applications

Two user applications have been developed in this project, one for Android and one for iOS, and they both include the same functionality, which will be covered in this section. Many of their features have been envisioned by Picwear and have been refined throughout the project.

4.2.1 Login and Signup

In order to use the application, the user has to be authenticated. The user can choose to sign up using either Facebook, or email and a password. The preferred authentication method is Facebook since additional user data such as name, gender and age can be gathered automatically, as well as the user's profile picture. The primary login screen is shown in Figure 3.

Figure 3. *The login screen on Android.*

If an email address is already taken, the signup should be denied. However, when signing up with Facebook, the user's email address registered on Facebook might be the same as another already existing Picwear account's email address. If that is the case, there is no safe and easy way from the client application to revert and delete the incomplete user just created. Instead, all this is done by the backend. Additionally, if the user has allowed Picwear to use his or her Facebook information, the backend also fetches this information in the signup process.

4.2.2 Following and Discover Feeds

Feeds are central to the mobile applications and act as an initial point of entry to other parts of the applications. The primary feed, displayed as the first view for an authenticated user, is the *following feed*, which can be seen in Figure 4. It works in a similar way to feeds in many other popular applications – such as Facebook, Twitter and Instagram – in the sense that it displays items from what the user is following. The second feed is the so called *discover feed*. This feed shows suggested items that may not be visible in the following feed, as to encourage the user to discover new items that might be of interest. The discover feed can also be filtered by various parameters, such as specific stores or price ranges. In doing so, the discover feed turns into the search view, as discussed in the next section.

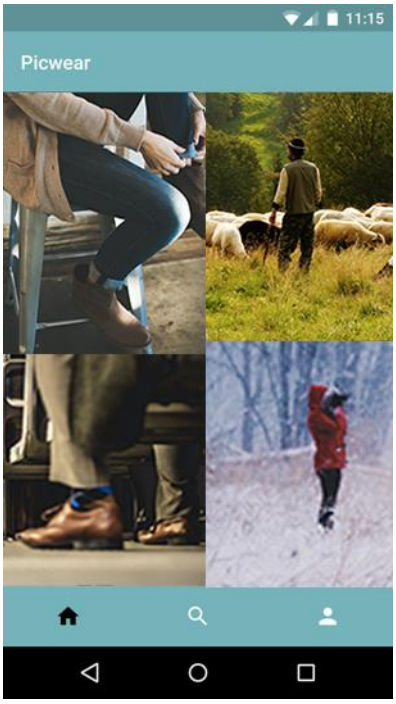


Figure 4. *The following feed on Android.*

Both the following feed and the discover feed are filtered by various global settings to provide more relevant results. Such settings include the user's gender, which is used to show only clothes that matches said gender, and the user's location, which is used to only show stores located within a specified distance. As the user might want to find clothes for someone else, these settings can also be changed in the settings of the application.

4.2.3 Search

The search feature enables users to filter products by category, store or brand, which can be done by either picking items in a list or searching using free text. Products matching the search are displayed in a feed similar to the other feeds in the application. Figure 5 shows an example of the search view.

Searches for multiple categories, stores and brands can be combined. If a user searches for several categories, all products matching any of these categories are displayed in the results. The same applies for stores and brands. If a search combines for instance a store and a category, only products matching each of these terms are displayed. Combining for instance multiple

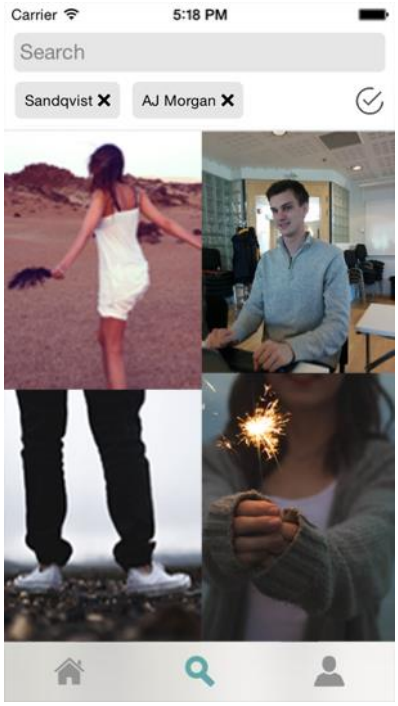


Figure 5*. The search feature on iOS.*

categories and multiple stores gives the user a result for products that matches any of the categories and any of the stores.

The user can also choose to follow specific search queries, which means that the results of that query will continuously appear in the user's following feed. For example, a user might be interested in something specific, such as shirts from a particular brand. In such a case, the user can search for shirts and the brand and then follow the selected search filters instead of following all shirts or everything from the particular brand.

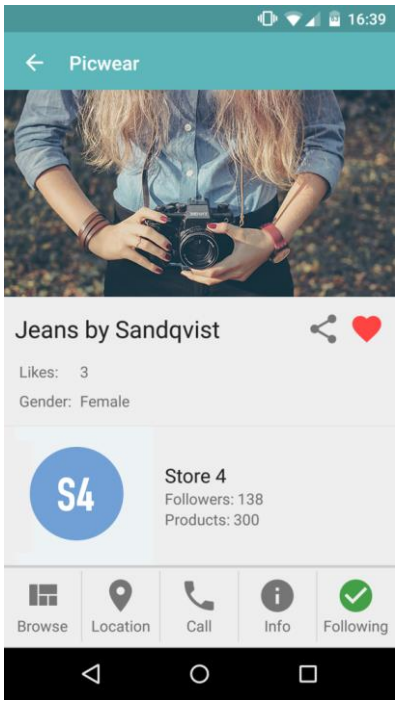


Figure 6. *The product details view on Android.*

4.2.4 Product Details View

Tapping an image in a feed opens up a detailed view of the corresponding product, as can be seen in Figure 6, where the focus is a big high-resolution image of the product. Below, follows some general information about the product and buttons for sharing, liking and other common purpose actions.

At the bottom of the product details view, there is a panel with information about the store supplying the item. From here, the user can follow the store, call it and see its location on a map. Tapping the 'Browse' button starts a new search query with the store as parameter, which will list all the products uploaded by that store.

4.2.5 Profile

The profile page for a user, seen in Figure 7, contains personal details, a profile photo and a feed containing all the products the user has liked, called a *lookbook*. The lookbook is a collection where the user can show clothes and styles he or she likes and will be able to share it with other people.

From the user's profile page, the account and application settings can be reached. These settings include changing gender and age, in order to get feeds customized for the audience, and account details such as changing name, email address or password. Among

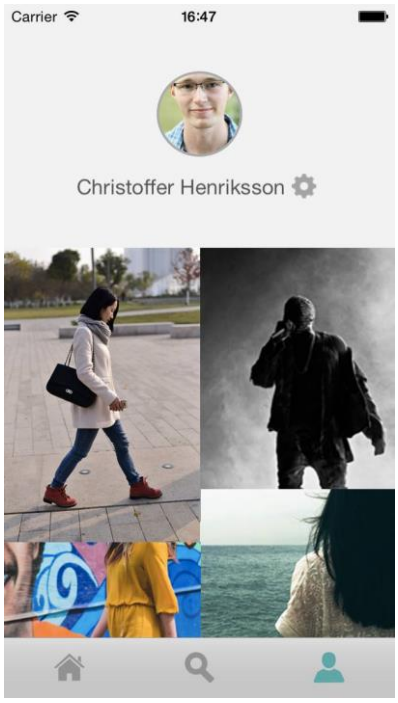


Figure 7*. The profile page on iOS.*

these settings, users can also list all of their followed search terms and remove any of them.

4.3 Sharing Website

A website has been built to enable sharing of products. This website contains metadata that enable external mobile applications to refer directly to products within the applications. The metadata attributes use a protocol called App Links, which is a cross-platform solution for linking between applications, developed by Facebook [45]. When a user opens a link to this website on a device that has the Picwear application installed, the specific product page will be opened in the application. Otherwise, a page on the sharing website is opened, containing an image of the product along with a link to install Picwear. On iOS devices, a banner that links to the application in App Store is shown, as seen in Figure 8. The website also provides additional meta information, like a thumbnail image and the name of the product, enabling richer previews for services that support this type of metadata, such as Facebook.

***Figure 8**. The sharing website on an iOS device with the Picwear application not installed.*

4.4 Store Admin Application

The administration portal for stores allows store employees to upload new products and change the stores' public information such as descriptions and addresses. The portal is a web application with a responsive design to enable store users to upload pictures from either their phone or their computer. Pictures can be cropped and rotated during upload. Furthermore, information about gender, category, brand and price is added for each product. Existing products can be changed and deleted as well. A screenshot of the store admin application can be seen in Figure 9.

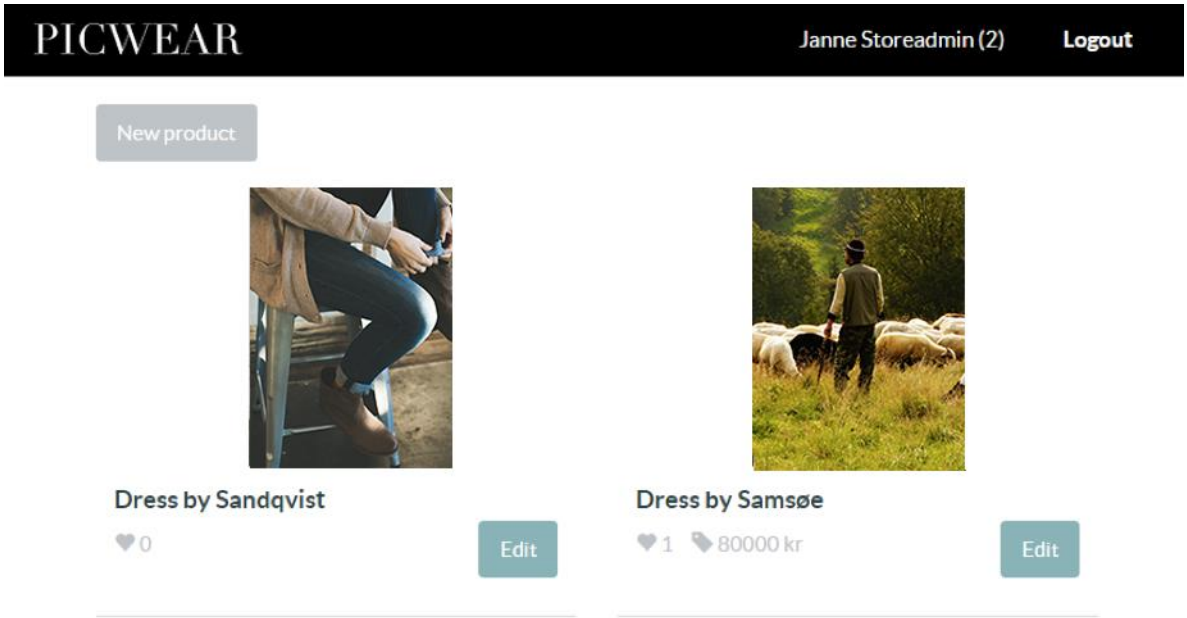


***Figure 9**. Overview of uploaded products in the store admin application.*

5 Technical Results and Discussion

This chapter presents technical results concerning the development as well as discussing topics regarding the technical issues presented in the purpose of the report. Starting with Android and iOS development, the chapter will also discuss the choice of building applications for multiple platforms. Furthermore, performance and security aspects of the persistence layer will be tackled. Finally, the overall management of the project is evaluated.

5.1 Android Development

The finalized product includes an Android application written for the Picwear service. The application had to be adapted in various ways to work with the Parse SDK. Larger such adaptations as well as other architectural code decisions and implementation will be discussed in this section.

5.1.1 Maintaining State

To minimize the required number of remote calls the application performs, maintaining a local state is important. The locally stored state can be used as a cache, thus reducing the need for fetching data. Using such a cache is important for mobile applications, as establishing new connections over a mobile network might lead to additional data charges and can incur significant delay [46], severely harming the user experience.

The Parse SDK provides ways to store individual, or groups of, so called *ParseObjects* locally. A ParseObject is essentially a local representation of a specific row for some type of class that exists in the backend database. However, these methods are not available for objects that are not direct subclasses, and as such cannot be used for the data structures returned for most of the custom backend calls implemented as part of the service. Consequently, the Android application had to implement custom ways to retain the instance state between different activities, without much help from the Parse SDK.

As discussed in the section 3.1.2 Lifecycle, the Android system destroys activities that it deems no longer needed. However, the system provides ways for the activities to store their current instance state in a bundle, as long as the data abides by some restrictions. ParseObjects does not follow such restrictions and can therefore not be directly stored in a bundle. This further complicates the storing of local state as activities cannot easily store their state themselves. To alleviate this problem, a concept of services was introduced. These services are responsible for fetching and maintaining data from the backend, moving this concern away from other components. This further allowed for separating the stored data, from the individual lifetime of the various consuming components, into a more global state. In doing so, the need for storing data in bundle objects could often be avoided. To make sure all components are accessing the same global state, the various

services are accessible only from a single service manager object, which in turn is implemented as a singleton.

One issue with said approach is that data remains in memory even when it no longer is needed, as the individual components of the application cannot easily know when to tell the service to remove data. To lessen the impact, each service is lazily instantiated, meaning it is created only once it is needed. However, this does not decrease the memory usage once all services have been created. Arguably, this might not make much of a difference since most of the stored data is relatively small. Even so, for less powerful Android devices, it might prove beneficial to aggressively destroy services, and potentially perform additional requests if the service has to be recreated.

5.1.2 Managing Asynchronous Tasks

Due to most of the data of the application not being locally stored, but rather located on the Parse backend, requests for data is performed frequently within the application. As mentioned in 3.4.1 The UI Thread Pattern, for such requests not to affect the perceived responsiveness of the application, it is important to perform them on a thread separate from the UI thread. This poses the problem on how to handle such multi-threading in a way that does not make the logic of the application overly complicated. Furthermore, the Android lifecycle might result in destruction of a component initializing a request before the request has finished, which is also something that should be considered.

The Parse SDK provides an API based on callbacks. These callbacks are guaranteed to be run on the UI thread. However, as already mentioned, it is possible for such a callback to be invoked after the calling component has been destroyed by the Android system. Trying to access the component during such a situation will be unsuccessful, most likely resulting in the application crashing. Avoiding these crashes requires callbacks to verify that the component still exists before continuing with any task that required the component. These verifications can easily be forgotten, resulting in bugs that are unlikely to occur but have severe effects once they do. Additionally, callbacks, especially as implemented in Java where each callback is based on creating anonymous classes, make for verbose code. Even more so for situations when callbacks has to be nested, for example when a request depends on another request to finish before.

An alternative, or rather a complement, to using callbacks is what is known as promises, or tasks as referred to in the Parse SDK. This is a separate, but from a functionality perspective identical, API also available in the Parse SDK. Using the task-based API makes chaining of multiple requests possible in a more compact way, reducing the code verbosity as compared to using callbacks. Furthermore, tasks can be composed in such a way that only a single error handler is required for multiple operations. Tasks can also be cancelled, making handling of the problem discussed above concerning components

being destroyed easier as one can cancel all currently executing tasks when the component is destroyed.

For the many benefits the task-based API brings, it has been the API of choice for most of the requests made in the application. However, a problem arose about handling errors that should be considered before using this type of API. Namely, task chains resulting in an error without having an error handler will silently be disregarded instead of being thrown.

5.1.3 Abstraction Using Asynchronous Methods

As already stated, many operations associated with the various services are performed asynchronously, and as such, the callers must handle these methods differently. Since this can be somewhat hard to change later on, from the perspective of the caller, a decision was made to write all methods, which might in the future become asynchronous, as if they were already asynchronous. Concretely, this means that these methods return completed promises for the return value instead of returning the value directly.

A benefit of this approach is that the caller, which in the application is oftentimes the UI layer, does not require changing if for example database caching would be introduced in the services. The services would then simply return a promise for fetching data from the cache instead. In essence, this allows for much greater flexibility in the underlying implementation than if the caller would expect a direct result. This is naturally only applicable when the caller performs the call on a thread where blocking is to be avoided, such as the UI thread. If that is that not the case, the method might as well have blocked the caller.

5.2 iOS Development

This section will cover two of the most interesting aspects of how the iOS application was developed in the project. At first, the programming language chosen will be discussed and then a few issues concerning the creation of user interfaces.

5.2.1 Objective-C and Swift

When this project started in the beginning of 2015, a stable version of the programming language Swift had not been released. This forced the project team to either choose to develop with the Objective-C language, which might make the codebase obsolete in the future, or start developing with Swift and deal with any flaws or inconsistencies that prerelease software comes with. Swift was chosen and Optionals, tuples and other features that were introduced in Swift have been utilized in the project. This is believed to have improved the quality of the code and to have speeded up the development. Some third party libraries that were used are written with Objective-C, but since Swift is compatible with Objective-C this has not been a problem.

5.2.2 Creating User Interfaces

Mainly storyboards were used to create the user interface of the iOS application. They provide a simple way to layout components and to get an overview of the different pages. One issue with storyboards, which were experienced in the project, was that it was hard for multiple people to work on the same ones simultaneously. The reason is that they are not optimized for use with version control systems such as Git. For example, the version of Xcode, Interface Builder and even the Mac operating system are part of the storyboard file. If developers have different versions of those, Git handles the storyboards as they are constantly changed, and it can be hard to keep them in sync. In the project, this was solved by creating parts of the user interface with other technologies such as .nib or .xib files. The most challenging parts of the interface, for instance those with custom animations, were created programmatically with Swift.

5.3 Development of the Web Application for Stores

The server-side code for the web application for stores is written in Node.js and is coupled to Parse via the Parse JavaScript SDK. There are no special API functions for the administration site in the backend as the communication goes through the SDK directly. Since both the backend and this web application are written in the same language, they share some common code modules. Currently, this web application is hosted by Parse just like the backend, but the code can be moved to any Node.js server and then run independently. The Express framework (StrongLoop, 2015) is used for organizing web routes, error handling and authentication of Node.js applications.

5.4 Multiplatform Development

Since the service created is available on multiple platforms, various decisions had to be made in regards to how this would be achieved. The choices made and reasoning behind them are presented in this section.

5.4.1 Native Applications in comparison to web technologies

As previously mentioned in this chapter, the user clients for the Picwear service was built as native applications for Android and iOS. The primary reasoning behind this choice was that users seem to prefer the experience of native applications compared to the experience of alternative technologies such as web applications [48]. Many frameworks such as Phonegap and Ionic have been developed to increase the user experience of web applications. The project team decided not to use these frameworks either, as their featured applications [49] [50], for instance TripCase, Sworkit and Untappd, were deemed to have an insufficient user experience.

Furthermore, the Picwear applications have high demands regarding performance. This is mostly due to the, at times, large number of pictures displayed simultaneously, as well as the ability to scroll through many pictures at a time. Such image handling is generally

performed more efficiently if allowed access to lower levels of the platform, as opposed to the higher level abstractions often provided by web frameworks. However, had the performance requirements of the service been less strict, web frameworks would have been a possible solution. Oftentimes, they provide the means necessary to share large portions of the codebase between different platforms, which could significantly reduce development time.

In contrast, the store administration application was built as a web application to be able to publish a version of it earlier. The site is not visible to end users and thus it has not as strict design and functionality requirements as the user applications, which should be as polished as possible. A website for uploading pictures and edit store information is easier to set up than building an administration application for both Android and iOS. A web administration portal would also have been a necessity later, for easier access to administrative tasks from for instance desktop computers.

5.4.2 Backend in the Multi-Platform Environment

To avoid implementing the same features on each platform, the backend API contains functionality shared between clients. It also restricts clients from performing any operations on actual data that are not called via the API. These enforcements simplify the work concerning data security, which will be discussed more in depth in the following section 5.5 Database and File Storage.

5.5 Database and File Storage

This section includes subjects regarding database and file storage used by the service, which is closely linked to how the backend of the application is implemented. The following subsections will discuss problems and solutions concerning core functionality, in relation to effectiveness, efficiency and security, provided by the backend service and the database. Discussions in this section will relate to topics introduced in the chapters 3.5 Mobile Backend as a Service and 3.7 Database.

5.5.1 Motivation of Using a Document Store

The NoSQL document store was chosen as database storage along with Parse as an MBaaS to fulfill all requirements for the service and to reduce the development time. With requirements and aspirations of user expansion, there was a need of scalability and a possibility to vary performance limits for the Picwear service in a cost-effective way. Many large MBaaSs, which assist with this scalability and flexibility, come with a NoSQL database [51, 52, 53]. The document store, which is the NoSQL database most similar to relational databases, was a good match for the type of data existing in the Picwear service, where queries needed to be filtered by multiple properties and relationships within the documents.

Comparing other NoSQL database types shows that they might not have been optimal for this project. Key-value stores lack the support of querying on other properties than the key, which is a necessity for building the feed in the Picwear service. Wide-column stores, such as Cassandra, fulfill all of the requirements but may be over-qualified and are more complex to use. Graph stores, which are built for applications focused on relationships, should be good for social networks. However, when performing operations not based on the relationships in the graph, the database is not as efficient. Furthermore, it is less scalable than the other mentioned NoSQL databases and is not as commonly used in the broad market [37].

Using relational databases was also an option, and since the stored model contains much related data, it would probably not have been a bad choice. The data would then have been more normalized, meaning it would be spread over a larger amount of tables, and more joins across tables would have been used. This is due to relationships using aggregations in a document store cannot be represented in the same way in relational databases. Using aggregations might require less computational work than joining tables, since the data to be appended already exists in place [54].

Additionally, the data model structure in the Picwear service has been loose and changing rapidly. In a document store, where the schema is unconstrained, changing the structure of data can be easier than in a relational database that requires more time consuming migration efforts. Unfortunately, the possibility of making errors during migrations is much easier in a loose NoSQL document database than in a traditional relational database.

5.5.2 Mobile Backend as a Service

Using an MBaaS seems to shorten the development time since it serves predefined implementations for specific functions and abstracts the database setup. The service is responsible for uptime, hardware performance and the possibility to scale for a larger amount of traffic. Scaling a platform is as simple as selecting the preferred price range which clearly requires less knowledge than hosting a platform. However, of course the data model and database queries must be able to scale as well.

There are also disadvantages using an MBaaS. To simplify the development of services using the MBaaS, some features are limited. For Parse, there are limitations with queries that cannot be written in Parse but can be written in MongoDB, the database Parse uses. For instance, Parse cannot match equality of arrays or combine multiple conditions within the same query with the OR operation. The OR operation only works with a combination of two or more fully defined queries. Other features, such as *map-reduce* [55] for performing more advanced calculations, are also absent. Consequently, applications with higher computational requirements or more advanced query support could benefit from building their own backend, but for simpler applications or for companies with fewer resources, MBaaSs such as Parse assist well.

5.5.3 Media Storage and Picture Sizes

The service does not only need a database to store data objects, it also needs a file storage from which the clients can download images. Currently, Parse is used for both database storage and file storage. However, the storage to which images are uploaded can be replaced at any time. The motivation for keeping this storage unconstrained is that the storage provider can be changed due to differences in pricing and data limits.

Uploaded images are downloaded and shown on a broad variety of devices, with different screen resolutions. To save bandwidth and device memory, all pictures are saved in multiple sizes. There are currently five different sizes, ranging from a width of 270 pixels up to 2000 pixels. Depending on the device resolution, one of the smaller sizes is downloaded for the feed and the full-sized image is visible when the user navigates to the product details page.

5.5.4 Querying Performance

Fetching users' feeds of pictures is central to the application and needs to be handled as efficient as possible. There are many calculations involved in deciding which pictures that should be included in a feed, and these will be more time consuming as more pictures are added to the service. The calculations include finding relevant items and filtering them by geographical location, price intervals and gender. Therefore, it is important to reflect over how the queries are built and how the data is modeled.

A core concept within the service is the categorization of all products. Each product belongs to a category, and the category structure is represented as a tree. This means that if a user searches for shoes, they will get all products with shoes as its category, but also all products of any subcategory of shoes. As an additional level of complexity, each category can be gender-neutral or exclusive for either men or women.

As described in the section 3.7.3 Modeling Tree Structures, two normal methods for representing a tree is for each node in the tree to include either its parent or its children as references in an additional field. In the Picwear service, these two solutions are not applicable. A search query including subcategories will become inefficient, because the NoSQL document stores do not support recursive queries [40] – especially in this case where the category tree has an arbitrary depth. To find the results, one query for each child's subtree would have had to be called independently, which would decrease the querying performance rapidly.

Two other, more suitable, approaches for categorization are *array of ancestors* and *materialized paths*. These methods require more computation whenever a category is moved within the tree structure, where all its children's paths need to be updated as well. However, because client search queries are prioritized over infrequently performed

category modifications, any of these methods would be more suitable than using parent or children references as described earlier.

The same concept of fast reads and slower updates is also applied on likes and followers. Since it would be inefficient to count the amount of likes of products and followers of stores each time details of a product is fetched, these values are instead stored as an integer as a separate property. Whenever a user adds or removes a like for a product, the field holding the amount of likes is incremented or decremented. Separating the amount of likes with the actual likes may increase the risk of inconsistency issues, which instead relies on the application implementation.

Representing relationships correctly in the NoSQL database will also give a positive impact of efficiency. Because the Parse database is a document store, the differences between aggregations and references, described in 3.7.2 Comparing Aggregation and References, are applied. Aggregating a relationship can lower the amount of requests the database have to make in order to fetch all data, since the data is already included. In the feed example discussed in this section, each picture contains multiple sources of different image sizes. These sources only exist in a single picture, which means that the relationship as an aggregation will benefit the query performance.

5.5.5 Security, Authentication and Permissions

Privacy is an important aspect of a social networking application. Content uploaded by users is sometimes only visible for groups of people and can often just be modified by the uploader. In the Picwear applications, focus has been to secure access to the backend service. Unauthorized users should not be able to fetch data, and authorized users should only be able to see their own and public data.

Searchable objects such as categories, brands and stores are publicly readable information as well as products uploaded by stores. The data that instead have restricted read access are likes, followed searches and user information. In these cases, only the user that the data belongs to can read and modify it. The restrictions are implemented using class-level permissions and access control lists in Parse, described in the section 3.5.2 Security Using Access Control.

Modification of stores must be available for all store users and this permission is therefore implemented using roles. Each store gets a role created during signup, which contains the store users who can edit store information and uploaded products and pictures.

5.5.6 Ordering and Relevancy

A problem may occur when pictures in the feed are ordered by the date they were uploaded. Some stores will upload several pictures at the same time, which will therefore

be shown one after the other on the client. If one store uploads many pictures, there will be less variation in the discover feed, which should be used to find new stores.

No advanced algorithm for sorting the pictures in the feed by relevance was implemented. Instead, the feed uses a simple randomization to rearrange the order so that two pictures uploaded at the same time will most probably not be shown one after the other. Still, pictures that are more recent should be placed higher in the feed, so therefore the order is originally based on the creation timestamp. To this timestamp, a positive or negative random value is added which will sum up to a *sort order* property saved to the picture data. A larger range for the random number will give more spread in the feed. Depending on the upload frequency and the number of active stores, this range can be adjusted to give a satisfying result. The property can be recalculated repeatedly to change the order of items from day to day.

Later, this sort value can easily be adjusted even further based on amount of likes or followers. The difficulty is instead how to weigh the factors against each other in order to give a good mixture. The long-term goal is that relevance of pictures should be individual for each user based on their likes, personal information and other interests.

The current solution does not solve the issue with one store uploading large amounts of pictures while other stores do not. In that case, that store will get more space in the application and thus more attention from the users. This is undesirable but will not be solved in the first version of the service. In this case, the sort order could be lower if the store has uploaded many products during the last month.

5.5.7 Feed Pagination

Pagination of feeds, used for supporting infinite scrolling, introduced some problems. The client application fetches a block of products from the feed on launch, and when the user has scrolled to the bottom of the feed, the next block of products is fetched. The database query can limit the number of products to match the block size and offset the number of products that has already been shown in the feed, but a problem occurs when the data has changed since the last fetch. If new products are uploaded, duplicate pictures will appear in the feed.

To solve this problem, the backend introduced sessions for feeds, which are reestablished when the user refreshes it. This session is represented by the time the feed was initially fetched and no session information is stored on the backend. As long as the client passes the session timestamp to the backend while retrieving feed pages, no duplicate products will occur. The backend logic is as simple as ignoring all products and followed searches created after the session timestamp.

5.6 Project Management

In order to build the Picwear service, project management methods as the ones mentioned in chapter 2 Method have been used. The following sections will discuss the development process, how the project was planned and the advantages and disadvantages of working with Picwear.

5.6.1 Development Process

Picwear's initial vision and plan was to develop an application for iOS, and if successful, branch out to other platforms at a later point in time. However, it was decided at an early stage that the development effort should instead be split into two separate teams, working in parallel on both an iOS and an Android version. The reasoning behind that decision was that certain tools, which are required for iOS development, were not available to all project members. Having too many developers working on the same codebase would also complicate the process and lead to more time spent on distributing work, instead of doing actual coding or engaging in high-level discussions. Since most of the service logic was platform independent and located in the backend, doubling the amount of supported platforms did not redouble the actual workload.

5.6.2 Planning

The planning outline created in the beginning of the project has generally been followed and all the features that were planned have been developed, but the order in which they were developed changed slightly during the project. The reason is mainly that Picwear changed the prioritization of features.

5.6.3 Collaboration with Picwear

The collaboration with Picwear had both advantages and disadvantages. Since Picwear was responsible for design and vision of the service, this project could focus on researching technical issues and developing software. This speeded up the project and meant that a larger part of the service had time to be developed. However, the collaboration also introduced some problems. One example is that the design was mostly optimized for iOS, which increased development time for certain Android features since standard components could not always be utilized.

To make the service easier to expand in the future, the project team made a few decisions that were not explicitly stated in the functionality description, given by Picwear. For example, each product in the functional description was intended to have a single picture. However, the two client applications and the model are built for handling products with more than one picture.

5.7 Evaluation of the Project

The answers in the project evaluation survey in Appendix B shows that Picwear is overall satisfied with both the process and the results. According to the survey response, Picwear valued collaboration and results as the most important project work aspects, for which they also gave the highest grades. The lowest grade was given for time efficiency, which they referred to as the amount of results per the number of calendar months worked rather than amount of actual work hours. Considering this, many hours were spent on research during the development process, which might have been lower if an established consultancy firm was hired.

6 Conclusion

While building multi-platform services, there is a choice to be made between developing standard native applications or applications built on top of a web-based framework. Using such a web framework would probably result in a product in less time than building multiple native applications, since the code only needs to be written once. However, performance issues and limitations of the web frameworks might result in native applications having a better user experience.

Using an MBaaS can make it easier for developers to create a backend in a time efficient manner that has sufficient performance and security for a service such as Picwear. However, an MBaaS may not be desirable for all types of applications, since those backend services often come with limitations.

This project has successfully developed a first version of the Picwear service including an Android application, iOS application, web application and accompanying backend. It includes all of the features that the project team originally planned and most of the features that Picwear had envisioned. The evaluation shows that Picwear is satisfied with the work performed in by the project team, but for a more complete service, there are still features to be implemented.

7 Future Work

Picwear has envisioned many features that did not fit the time schedule for this project. These included features such as adding products to a shopping list and receiving push notifications. It also included building an administration site for managing records of brands, categories and stores. Additionally, the store admin application might get features that add value for the stores, such as statistics and analyses.

Since the Picwear service will probably be released soon after this project has ended, the first published version will not be fully featured. As a part of Picwear, the development team will continue working with the service to implement remaining features of Picwear's vision. The team will also continue polishing existing features to improve the user experience further.

8 References

[1] J. Sternö and T. Nielsén, "Modebranschen i Sverige – Statistik och analys 2015," Näringsdepartementet, Ed., ed. Stockholm: Volante Research, 2015.

[2] M. Secund, D. Lundgren and A. Noort, "Business Plan – Speegl," 2014.

[3] C. Ladas, Scrumban - Essays on Kanban Systems for Lean Software Development, Modus Cooperandi, Inc, 2008.

[4] L. Rising and N. S. Janoff, The Scrum Software Development Process for Small Teams, vol. 17, IEEE Software, 2000.

[5] Google Inc., 2015. [Online]. Available: http://developer.android.com/. [Accessed 8 May 2015].

[6] Apple Inc, 2015. [Online]. Available: https://developer.apple.com/. [Accessed 8 May 2015].

[7] Atlassian Confluence, "Bitbucket," 2015. [Online]. Available: https://bitbucket.org/. [Accessed 19 May 2015].

[8] Chalmers Teknologkonsulter AB, 2015. [Online]. Available: http://www.ctk.se.

[9] Google Inc., "Android Studio Overview," 2015. [Online]. Available: http://developer.android.com/tools/studio/index.html. [Accessed 19 May 2015].

[10] Google Inc., "Activities," 2015. [Online]. Available: http://developer.android.com/guide/components/activities.html. [Accessed 11 May 2015].

[11] Google Inc., "Parcelable," 13 May 2015. [Online]. Available: http://developer.android.com/reference/android/os/Parcelable.html. [Accessed 13 May 2015].

[12] "google-gson - A Java library to convert JSON to Java objects and vice-versa," 2015. [Online]. Available: https://code.google.com/p/google-gson/. [Accessed 11 May 2015].

[13] Apple Inc, "About Objective-C," 17 September 2014. [Online]. Available: https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html. [Accessed 3 April 2015].

[14] Apple Inc, "Why Objective-C?," 15 November 2010. [Online]. Available: https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/OOP_ObjC/Articles/ooWhy.html. [Accessed 3 April 2015].

[15] Apple Inc, "The Swift Programming Language: About Swift," 8 April 2015. [Online]. Available: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/. [Accessed 10 April 2015].

[16] C. Eidhof, F. Kugler and W. Swierstra, Functional Programming in Swift, Florian Kugler, 2014.

[17] Apple Inc, "The Swift Programming Language: The Basics," 8 April 2015. [Online]. Available: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html. [Accessed 10 April 2015].

[18] Apple Inc, "Storyboard," 18 September 2013. [Online]. Available: https://developer.apple.com/library/ios/documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html. [Accessed 10 April 2015].

[19] Google Inc., "Processes and Threads," 2015. [Online]. Available: http://developer.android.com/guide/components/processes-and-threads.html. [Accessed 5 May 2015].

[20] Apple Inc, "About Threaded Programming," 2015. [Online]. Available: https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/Multithreading/AboutThreads/AboutThreads.html. [Accessed 2 May 2015].

[21] I. Seppälä, "Design pattern samples in Java," GitHub repository, 2015. [Online]. Available: https://github.com/iluwatar/java-design-patterns/blob/c1b09cbf6146912640bbc019415d6577e0894a6d/README.md. [Accessed 15 May 2015].

[22] Promises/A+ organization, "Promises/A+," 2015. [Online]. Available: https://promisesaplus.com/. [Accessed 5 May 2015].

[23] BoltsFramework, "BoltsFramework/Bolts-Android," 2015. [Online]. Available: https://github.com/BoltsFramework/Bolts-Android#tasks. [Accessed 2 May 2015].

[24] BoltsFramework, "BoltsFramework/Bolts-iOS," 2015. [Online]. Available: https://github.com/BoltsFramework/Bolts-iOS#tasks. [Accessed 2 May 2015].

[25] Microsoft, "Mobile App Service," 2015. [Online]. Available: http://azure.microsoft.com/en-us/services/app-service/mobile/. [Accessed 25 April 2015].

[26] Firebase, "Firebase - Build Realtime Apps," 2015. [Online]. Available: https://www.firebase.com/. [Accessed 25 April 2015].

[27] "Parse," Parse, 2015. [Online]. Available: http://www.parse.com. [Accessed 25 April 2015].

[28] Kinvey, "Mobile Backend as a Service (MBaaS) for the Enterprise," 2015. [Online]. Available: http://www.kinvey.com/. [Accessed 25 April 2015].

[29] Parse, "REST API Developers Guide," Parse, 2015. [Online]. Available: https://parse.com/docs. [Accessed 20 April 2015].

[30] Parse, "Rest API Developers Guide," 2015. [Online]. Available: https://parse.com/docs/rest/guide. [Accessed 20 April 2015].

[31] S. Yu, C. Wang, K. Ren and W. Lou, "Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing," INFOCOM, 2010 Proceedings IEEE, 2010.

[32] Google Inc., "WebView," 13 May 2015. [Online]. Available: http://developer.android.com/reference/android/webkit/WebView.html. [Accessed 13 May 2015].

[33] Apple Inc, "UIWebView Class Reference," 2015. [Online]. Available: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIWebView_Class/. [Accessed 13 May 2015].

[34] Google Inc., 2015. [Online]. Available: https://angularjs.org/. [Accessed 25 April 2015].

[35] P. J. Sadalage and M. Fowler, NoSQL Distilled: A brief guide to the emerging world of polyglot persistence, Pearson Education, Inc, 2013.

[36] MongoDB, Inc., "NoSQL Databases Explained," 2015. [Online]. Available: http://www.mongodb.com/nosql-explained. [Accessed 11 April 2015].

[37] DB-Engines, "DB-Engines Ranking - popularity ranking of database management systems," May 2015. [Online]. Available: http://db-engines.com/en/ranking. [Accessed 17 May 2015].

[38] J. Bryden, "Guide to Wide-Column Stores – NoSQL Explained," 2014. [Online]. Available: http://nosqlguide.com/column-store/nosql-databases-explained-wide-column-stores/. [Accessed 11 April 2015].

[39] "Data Model Design," MongoDB, Inc, 2015. [Online]. Available: http://docs.mongodb.org/manual/core/data-model-design/#data-modeling-referencing. [Accessed 15 April 2015].

[40] MongoDB, Inc., "Model Tree Structures in MongoDB," 12 May 2015. [Online]. Available: http://docs.mongodb.org/manual/tutorial/model-tree-structures/. [Accessed 14 May 2015].

[41] C. S. Mullins, "An Introduction to Recursive SQL," 8 May 2014. [Online]. Available: http://www.dbta.com/Columns/DBA-Corner/An-Introduction-to-Recursive-SQL-96878.aspx. [Accessed 22 April 2015].

[42] I. Katsov, "NoSQL Data Modeling Techniques," 1 March 2012. [Online]. Available: https://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/. [Accessed 22 April 2015].

[43] Git, "About - Git," 18 January 2015. [Online]. Available: http://git-scm.com/about. [Accessed 2 May 2015].

[44] M. E. Maximilien, H. Wilkinson, N. Desai and S. Tai, "A Domain-Specific Language for Web APIs and Services Mashups" in Service-Oriented Computing – ICSOC 2007, Springer Berlin Heidelberg, 2007, pp. 13-26.

[45] APPLINKS.ORG, "About," 2015. [Online]. Available: http://applinks.org/about/. [Accessed 1 June 2015].

[46] I. Grigorik, "Mobile Networks" in High Performance Browser Networking, O'Reilly Media, 2013, ch. 7.

[47] StrongLoop, "Express - Node.js web application framework," 15 February 2015. [Online]. Available: http://expressjs.com/. [Accessed 17 April 2015].

[48] C. Tossell, P. Kortum, A. Rahmati, C. Shepard and L. Zhong, "Characterizing web use on smartphones," ACM, New York, 2012.

[49] Adobe, "Apps Created with PhoneGap," 2 06 2015. [Online]. Available: http://phonegap.com/app/feature/. [Accessed 2 06 2015].

[50] Drifty, "Ionic Showcase," 02 06 2015. [Online]. Available: http://showcase.ionicframework.com/. [Accessed 02 06 2015].

[51] Microsoft Corporation, "DocumentDB - NoSQL data management," 2015. [Online]. Available: http://azure.microsoft.com/en-us/services/documentdb/. [Accessed 17 May 2015].

[52] Google Inc., "Cloud Datastore - NoSQL Database for Cloud Data Storage," [Online]. Available: https://cloud.google.com/datastore/. [Accessed 17 May 2015].

[53] Amazon Web Services, Inc., "Running NoSQL Databases on AWS," 2015. [Online]. Available: http://aws.amazon.com/nosql/. [Accessed 17 May 2015].

[54] P. Mishra and M. H. Eich, "Join Processing in Relational Databases," March 1992. [Online]. Available: http://www.diku.dk/hjemmesider/ansatte/henglein/papers/mishra1992.pdf. [Accessed 17 May 2015].

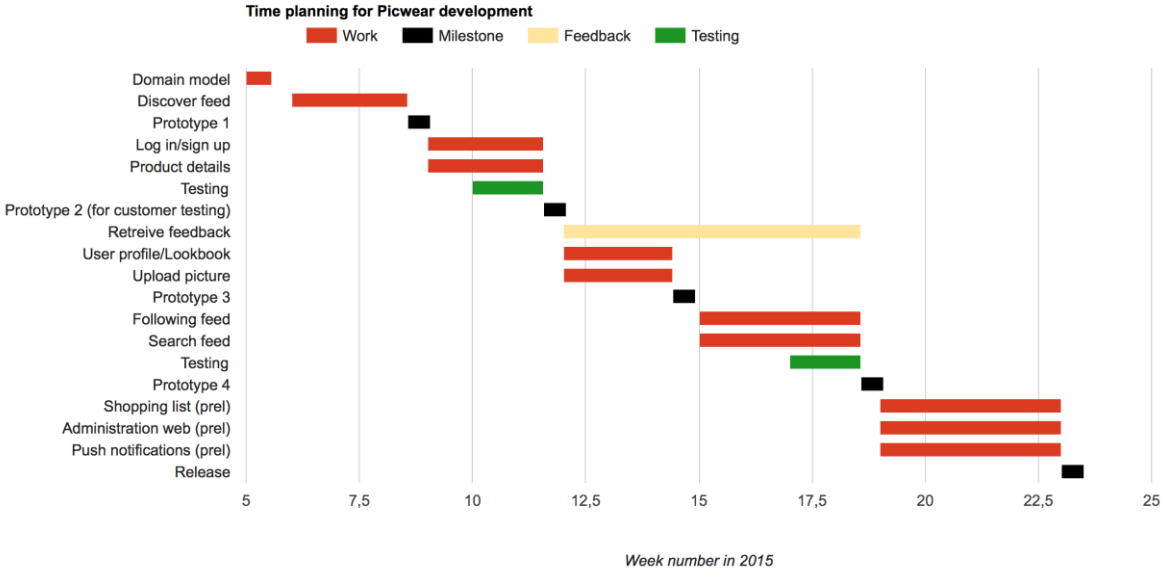
[55] MongoDB, Inc., "Map-Reduce," 4 April 2014. [Online]. Available: http://docs.mongodb.org/manual/core/map-reduce/. [Accessed 14 May 2015].

[56] J. Siracusa, "Mac OS X 10.7 Lion: the Ars Technica review," 20 July 2011. [Online]. Available: http://arstechnica.com/apple/2011/07/mac-os-x-10-7/10/. [Accessed 3 April 2015].

[57] Apple Inc, "Transitioning to ARC Release Notes," 8 August 2013. [Online]. Available: https://developer.apple.com/library/mac/releasenotes/ObjectiveC/RN-TransitioningToARC/Introduction/Introduction.html. [Accessed 3 April 2015].

[58] Joyent, Inc, "About Node.js," 2015. [Online]. Available: https://nodejs.org/about/. [Accessed 2 May 2015].

Appendix A Development Time Planning



Appendix B Project Evaluation by Picwear

What expectations did you have of us before the project started?
To have a Minimum Viable Product (MVP) for one operating system ready for launch during the spring. To have frequent, open and informal communication.

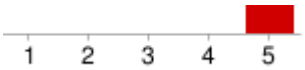
What other expectations (if any) would you have had from a consultancy firm?
More formal approach. Faster delivery of an MVP. After sales support.

How will did the project match your expectations?

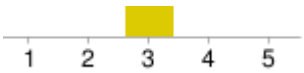
How satisfied are you with...

the ongoing communication during the project?

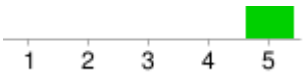
the overall collaboration?



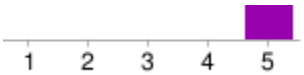
the time efficiency?



the results of the project?

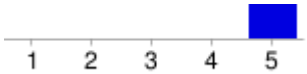


the project as a whole?



Anything else that you were either especially satisfied or dissatisfied with during the project?
Really satisfied with the guys, they have been working independently and knew quickly what and how we want the app to be.

How likely is it that you would recommend the project team?



Rank what you think is the most important of the following. 1 for most important and 3 for the least important.

1. Collaboration
2. Results
3. Time efficiency

Appendix C Functionality Description by Picwear

Authors: M. Secund, D. Lundgren, A. Noort

End-User V1.0

Screen 0.0: Login/register screen (only first time or when logged out):
Register with Facebook button <- Big, blue and most attractive looking
Register with Email button
Log in button -> leads to screen with username and password prompt

Screen 1.0: follow (homescreen)
In this screen all the items from the stores, brands and items the user follows will be shown.

Top bar
In the top left corner there is a menu button that leads to screen 8. In the top right corner there is a shopping list button that leads to screen 4.0.

The feed
When browsing the feed should **only show the pictures**, no logos, text or buttons should be visible on the pictures. When a user clicks on an item, it pops up to screen 1.1.: the enlarged photo + more info screen. We thought max. two pictures side by side is a good format (like Wish, Pinterest) and that the pictures automatically fill the whole width and height. Pictures in landscape mode will be reserved for customers with a premium account in the future.

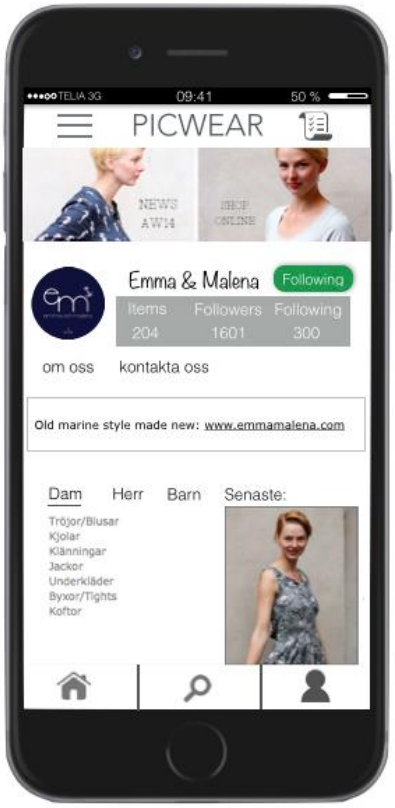
Bottom bar
On the bottom of the browse screen there needs to be a bar with 3 buttons that are permanent (sort of like Instagram). The left one is the homescreen button (screen 1.0), the middle one is the search/browse button which lead to screen 5.0, and the right one a button that leads to the user's personal account (screen 3.0). Lastly, there should be a button that leads to the location and time filters. This can be a drop down in screen 1.0. Here users can choose a city or choose distance from their current location. They also need to be able to filter photos on when they were uploaded.

When a user scrolls down in the feed, both the top and bottom bar dissapear so it only shows the pictures. When the user scrolls up they re-appear again.

Screen 1.1. Enlarged photo + more info pop up screen.
This screen shows a larger version of the photo and all the information we have on that item provided by the store: brand, item (pants, sweater, etc.), price, color, fabric, style (formal, sport, etc.).

On top of the picture the store name/logo needs to be shown and an option to follow this store. The logo needs to be clickable and lead to Screen 2.0: the store profile screen. On the bottom of the picture there needs to be two buttons: a "like" button (or similar) and a add to shopping list button. Clicking the like/love button will add the item to the user's personal lookbook in screen 3.0. Clicking the add to shopping list button will add it to their shopping list in screen 4.0. There is also a share button where users can share this item to social media etc. In the corner, a little flag should be present where users can report it if an item is not available in the store.

Screen 2.0: store/profile pages
In this screen the profile of a store will be visible. It should be possible to see how many followers they have and how many pictures they have uploaded (like Instagram) and a description of the store. Also a "follow" button should exist, if the user already follows the store it should be a "pressed" button "following".



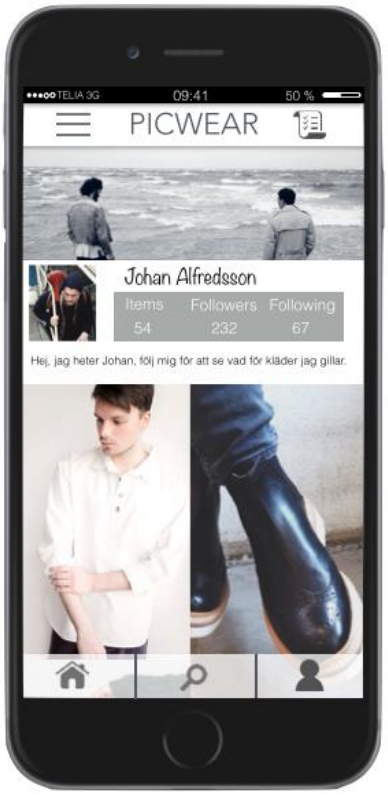
Here all info about the store should be collected. It should have three categories: "about us", "contact us", and "find us". The find us button will lead to map screen 7.0 and show where the store is. There needs to be an arrow (back button) in the upper corner which leads to the last page visit.

It should be very easy to see what pictures the store uploaded. This can be done by clicking a button which leads to the search/browse screen 5 with a filter for that specific store already active.

It would be a nice function if the latest upload is visible in this screen too but might be something for the next version.

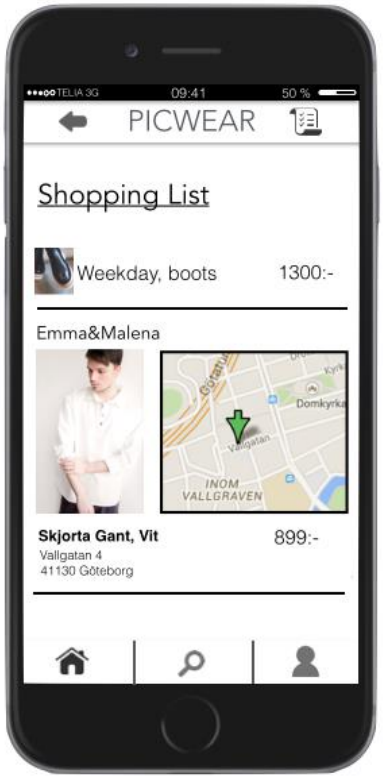
Screen 3.0: lookbook

The lookbook is the user's personal page. Here all the pictures that the user has liked will be stored, also how many followers the user has and how many the user follows. There will be a short "about" text (optional, 140 characters) under the profile picture. The "menu button" should be directed to screen 8.0 menu. Clicking an item will bring the user to screen 1.1 again for the enlarged photo and more info for that item. Add a button to change profile picture and text etc. (see how Facebook is doing it).



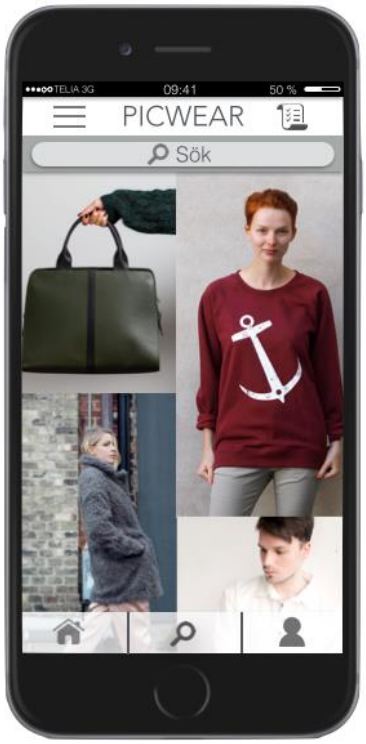
Screen 4.0: shopping list

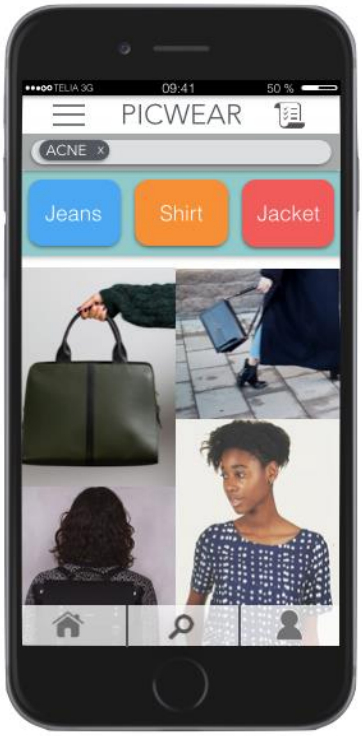
This page collects the items that the user adds to the shopping list. There should be a button here that leads to the suggested shopping route, screen 9.0. Also, buttons to delete items from the list should be visible. Clicking the item would obviously lead to the screen 1.1 for the item again.



Screen 5.0 browse/search

The search button should lead to a screen that leads to a browse page that shows recently uploaded items from all stores (like Instagram). When a user starts searching, the search bar should give suggestions. When one search filter is active, the most popular sub-categories should be suggested at the top of the page (brands, colors, etc.) (like Pinterest). When the user decides to search manually the most relevant items should show up in the results.



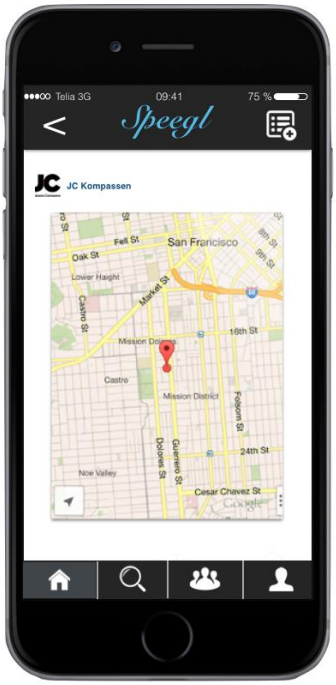


It should be possible to choose if you want to search for stores or items, like Instagram (user or hashtag). The difference is that you could follow the hashtags in our app.

New: it should be possible to follow items, and brand, not only stores. For example the user maybe want to follow 2hand+shirt, to see all shirt uploaded by the vintage stores. Still to discuss, should the following feed be separated in two pages, one with user accounts and one with item or brand.

Screen 7.0 location

This will be a map in the app where you see where you are and where the location of the store is.



Screen 8.0 menu

Here the most essential options are listed. The structure should look a bit like the one from Wish. The most important function here is to set the size of the radius to filter items that are closest to the users location. The minimum 1km, then 5, then 10 and maximum 50.

Screen 8.1 settings

- Username
    - Change username
- Password
    - change password (write two times)
- Email
    - Change Email
- About (optional)
    - short description of the store/user
- Phone number (optional)
- Gender: Male, Female ->
- Contact Picwear
    - fill in form and send button (to info@picwear.com)
- Secrecy
    - Our policy

Other functions:

- Push notifications - send notifications when user gets feedback request, when two weeks inactive, when new features come out, etc.
- Ask users whether they actually bought the items on their shopping lists -> create database.
- Have a “rate this app” popup for users that come back often
- Invite your friends function

Problems right now:

- finding/following friends
    - separate feed?
- Lookbook: more like boards as with Pinterest? Possible to create categories?

Stores

The stores will have a special store account which allows uploading pictures. When clicking the camera button:

Screen 1: take/upload a photo

Screen 1.1: crop photo so that it fits the feed perfectly
Screen 1.2: add tags: man/woman/unisex, item (dress, jeans, etc.), brand, price, style (fancy, business, casual, sport, etc.).

The store should also have access to the partner portal on the website.

Screen 1: login
Screen 2: overview of all uploaded photos. It should have 2 tabs: active and inactive photos. The active photos should have an X button to deactivate it and the inactive photos the opposite.