Benza: A Carpooling Mobile Application for CS3528 Software Engineering and Professional Practice

Duncan van der Wielen, Magni Hansen, Loredana Cozzone, Greta Šimonytė

April 2019

Abstract

This report presents the progress in development of an Android mobile application, Benza, which aims to connect users based on geographic factors and ultimately enable them to carpool in a structured manner. There will be a summary of the project and a review of the current competition, followed by a discussion of the project's requirements and design (as laid out in the initial White Paper). While the White Paper described early working prototypes of functional requirements, these features have been developed further to perform in a cohesive manner and deliver the initial requirements. The code behind the project will be explored with a focus on key implementation decisions before examining the team's testing strategy. An evaluation of limitations to the project will be based on the results of testing and will pay particular attention to fulfilment of the requirements. A video demonstration of Benza is available on Youtube.

Contents

1 Acknowledgements

The team would like to thank:

Nicolò Mazzucato, for invaluable contributions to the mobile application and Dr. Martin Kollingbaum, for guidance throughout the project.

2 Introduction

2.1 Overview

Background. The carpooling concept is altering society's definition of personal transport. Everyday commuters, poor students and adventurous tourists are all part of the 21st century revolution that says your vehicle is a shared resource rather than a mutually exclusive possession [REFERENCE!]. The benefits are clear to see: vehicle owners pay less per litre of fuel with a passenger contribution, roads are less congested, leading to shorter journey times and pollution from private vehicles is reduced, benefiting our environment.

Motivation. Although many people might offer a seat in their car or would be interested in finding one, there is still a lack of suitable middleman services that would bring these two parties together. The obvious difficulty in providing such a service is that users on the platform need to trust one another before interacting in a real world situation. This issue has been the main focus of industry leading companies like BlaBlaCar (see Section 3) and is something that needs to be prioritised in any newcomer.

Rationale. Our innovative carpooling service enables its users to plan shared car journeys and split the costs of their journey, while reducing traffic congestion and pollution as well as meeting new people. To use the app, a user only needs to create an account, join a group and connect with other users in their group to organise a trip. Unlike other ride-sharing services, in Benza there is no clear distinction between user types, so anyone with an account can be either a driver or a passenger.

2.2 Team Organisation

The team consists of four members: Duncan van der Wielen is the team leader, Magni Hansen is his deputy and Greta Šimonytė is a programmer who shares her responsibilities with Loredana Cozzone, Benza's marketing lead.

After the departure of a valuable fifth member, who had previous experience in software development and knowledge about cutting-edge technologies and web development tools, roles within the team were updated to include extra responsibilities.

2.3 Project Management Strategy - GRETA

At the start of the project the Agile approach was chosen to be used for the development of software application. During the first half of the project development, team had fortnightly meetings with the academic course guide, Dr Martin Kollingbaum. These meetings were considered to be the start of the new iteration of the cycle which consisted of planning the next steps to be done, splitting them into small components, tasks and working on them during the next two weeks. The work done was presented to the guide and the team during the meeting instantly receiving the feedback. The team also held weekly informal meetings as well as worked in pairs when needed.

In the second half of the development cycle, weekly meetings with the team's academic course guide were held to discuss the progress and steps for the next stage in the development cycle of the software. The structure of the meetings was similar to those in the first half of the project development, however,the length of the sprints was adapted to fit between weekly meetings.

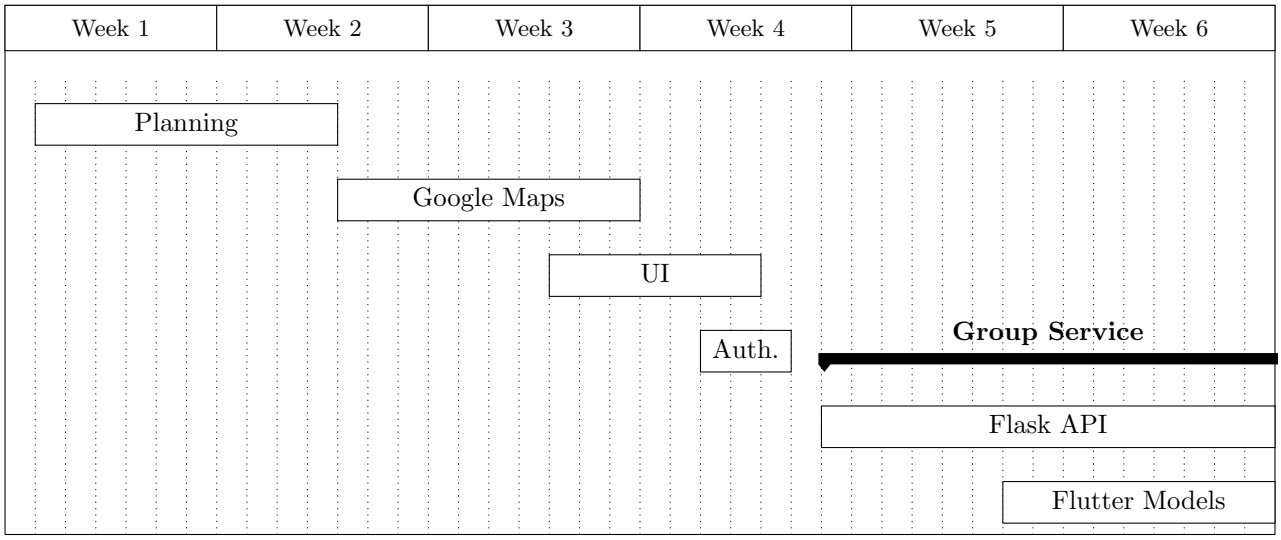


Figure 1: Development schedule from 14th January to 24th February 2019 (The remaining segments are available in Appendix D)

The rest of the implementation schedule can be found in Appendix D.

Unfortunately, team members soon found themselves struggling to find convenient time for the informal meetings amid their various commitments and schedules. To counter the slow progress, the team decided to adopt a loosely defined somewhat agile strategy, where members would contribute to the project in a time of their choice and to work in pairs whenever possible. However, as the spring break approached, the team found it easier to meet more often and primarily focused on finishing the project on the set deadlines.

Multiple tools were used in order to keep everyone on the team well informed about the progress made in development of the project as well what needs to be done. Google Drive was one of the tools used for general organisation and sharing of material. As all the materials were held in the same place, everyone within the team were to able to access important information such as logs from the meetings or contribute to their part of the report. Software development platform GitHub was used for tracking the implementation progress of the application. Team members could see the changes made to the code, work on the issues found and move to the next stage of implementation.

3 State of the Art

The recent surge in attention to the carpooling sector presents challenges to a startup like Benza: various functionalities of our application are already delivered by Lyft, BlaBlaCar, Uber and Waze Carpool - the most popular applications currently in deployment. For Benza to be marketable, it will differentiate itself from competitors by offering carpooling to persistent and geographically-based groups of users, resulting in a user experience akin to group chats in a social networking application.

3.1 Competitors

BlaBlaCar is the application that is most similar to Benza. The key difference is that Benza does not make a clear distinction between *driver* and *passenger*. In contrast, when a user interacts with BlaBlaCar, they have to specify if they are offering a ride, as a driver, or looking through the current available journeys and requesting to join a ride, as a passenger.

BlaBlaCar has seen exponential growth over the last 12 years, gaining 15 million users at the peak of its take-off during 2014 [13] and totalling over 70 million users at the start of this year [14].

The app started as French ride-sharing service Covoiturage.fr. As the service gained popularity, the company fuelled domestic growth by providing carpooling infrastructure to several high profile companies[1] before releasing BlaBlaCar on Apple's UK App Store in 2011. Since its debut, BlaBlaCar's parent company has re-branded to suit their unicorn and continues to acquire and rebrand ride-sharing services in all of their countries of operation [1, 2, 3, 4] as well as push its service to new users through existing channels like Google Maps (Figure 2).

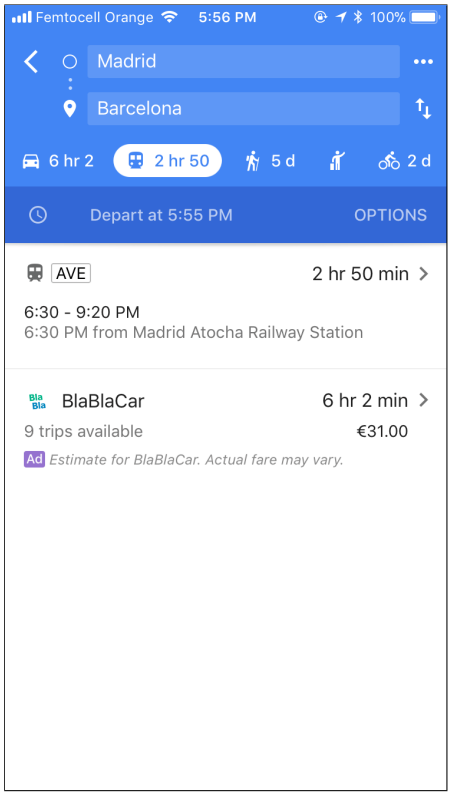
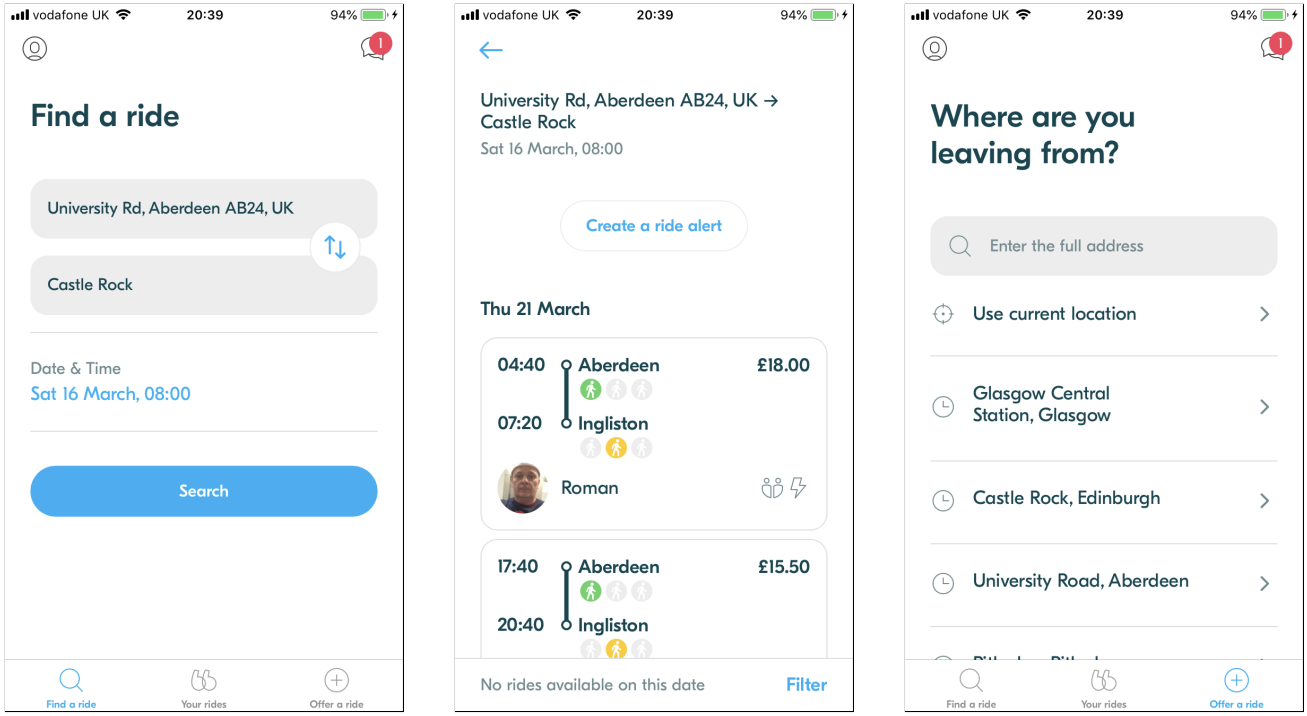


Figure 2: BlaBlaCar recommendations within Google Maps using ads [15]

BlaBlaCar is an industry leader with a long history, so we expect that their software is engineered to high standards. However, our team have found that the UX of their app can often lack intuition as well as a sense of community. We aim to exploit these shortcomings to the benefit of Benza. For example, where BlaBlaCar forces their users to complete boring text fields (Figure 3a), Benza will instead show a list of nearby carpooling groups without the hassle of searching for a starting location. This example also illustrates how Benza interprets carpooling as communities of people, instead of discrete transactions. More details about the value that Benza brings to its users can be found in Section 3.2.

[1] IKEA [29], Vinci Parking [28, page 70] and Carrefour [5]

Uber is a spectacularly successful company that provides features resembling those seen in our application, so one might assume that it falls into the scope of Benza's most serious competitors. Indeed, we listed it as one of our main competitors in the initial White Paper. However, Uber is focused on on-demand transportation by designated drivers, not crowd sourced ride-sharing. The service provided by Uber profits its drivers (who are paid by Uber), contrasting with Benza's payment system, which was initially designed to accommodate in-app payments but has now been reduced to a simple recommendation system (see Section 5.5).



(a) Finding a ride (b) List of rides (c) Start of an offer

Figure 3: Mobile interface for BlaBlaCar (iOS)

Much like BlaBlaCar, Lyft was initially released as a proof of concept service by its parent company Zimride after some years spent developing and fund raising. Shortly after its initial release in 2012, Lyft raised $15 million in funding [11]. Following this successful series B funding round, the service's founders promptly sold parent company to Enterprise Holdings (owner of Enterprise Rent-a-car) so they could focus on Lyft. The sale came immediately after a further round of funding worth $60 million and also led to Lyft moving into new domestic markets, encroaching on Uber's market space. Years of steady growth and a sequence of successful funding rounds have culminated in its recent IPO, which was one of the most successful tech IPOs in history [32]. In mid-2018, Lyft claimed to have 35% of the USA ride-sharing market [7] - a measure of how mature the company has become.

When comparing the Lyft and Uber experiences, it is clear that Lyft places more emphasis on empathy. It accomplishes this through marketing strategy rather than app functionality [27] and UI, both of which remain very similar to Uber's app. Although this similarity might seem contradictory for an app that is trying to disrupt a market, the familiarity serves as a pull factor for Uber's existing users. When Uber entered the market, it was subjected to a slew of bad

press for its controversial and aggressive targeting of the taxi industry. Years later, it still bears the burden of negative coverage that accompanies its relentless attitude toward trailblazing in various other markets [24]. Lyft has been able to capitalise on this strategy, operating within the larger company's wake to distance itself from any association with Uber. So, while Lyft does appear to value its user community on a more personal level than Uber does, Benza aims to go above and beyond by providing a sociable experience to users before even entering a car (Section 3.2).

Although Lyft is certainly a competitor to be wary of, it also caters more for the on-demand ride-hailing industry, which is not the market space that Benza wants to occupy. In addition to this difference in market space, Lyft is currently still limited to the USA market, although it briefly partnered with Chinese ride-hailing service DiDi Chuxing to provide its customers with a seamless experience in some Asian countries [6]. It may be the case that the EU's GDPR discourages non-European unicorn startups from prioritising the European market. This hypothesis could become very relevant if the UK were to leave the EU, abandoning the GDPR and making itself more accessible to tech startups like Lyft.

Waze Carpool is Google's carpooling service, primarily catering for coworkers that commute using common routes on a daily basis. It is especially geared towards employees of large companies, as seen by Google's collaboration with Amazon warehouses across the USA [25]. Designed primarily as an extension of the original Waze navigation app, which specialises in avoiding high-traffic and otherwise time-consuming routes, Waze carpool is Benza's closest competitor after BlaBlaCar. The app's ability to gamify the ordinary process of navigation, coupled with its use of saved "routes" where Benza's would use groups, makes it the most community-focused of the competitors discussed in this report.

Taking these similarities into account, it is advantageous that Benza will be released on the UK market: Google's product is currently only available in the USA, Mexico, Israel and Brazil. As Waze continues to gain traction with an audience of a country that often imparts its cultural trends to the UK (particularly the British youth), Benza could be marketed as a British substitute for something that is already on the fringes of common knowledge. Waze Carpool's audience is composed largely of millenials [8]; another positive indicator for Benza, which will be released for beta testing in a university environment (Section 7.1), where the majority of the student population falls within the millenial cohort.

3.2 Innovation

Constructing a vision for Benza has been an important part of defining the aims and values of the application. These values (Figure 4) led to the prioritisation of certain design features and changes to the requirements analysis during this first development cycle (see Sections 4, 5.5). This section will explore how Benza will realise the team's vision and the defining features that Benza users can expect from the app.

Community: Benza aims to connect people in a manner that encourages organic growth of carpooling groups as opposed to the transaction-like features that characterise the apps of our competitors. Benza users can create their own groups with unique names, locations and member lists (Figure 11a) to better reflect the way people already coordinate shared travel arrangements. By centring our app on a list of nearby groups (Figure 10c), new users are greeted by familiar locations and more meaningful group names. The familiarity of location is designed to make

the new user feel as if they already have a stake in these groups - that they belong to and are welcome in them. If any user wishes to create a new group, they can do so at the tap of one button, followed by an intuitive four-step process where they provide details for their group. As more groups are created, the app's home page will become more dynamic, changing its content depending on the user's proximity to group locations to deliver an engaging experience to users that remains relevant to their own location.

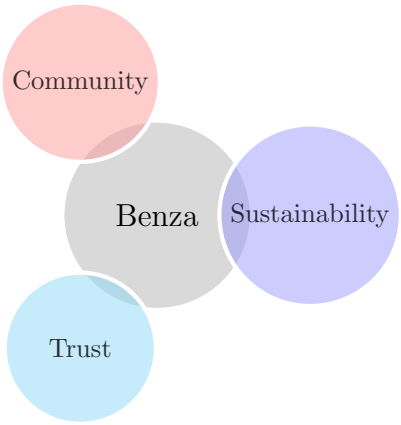


Figure 4: Benza's values

Sustainability. Benza facilitates environmentally friendly travel for all its users by maximising the person to vehicle ratio, fulfilling the contemporary meaning attached to "sustainability". However, the development team aims for Benza to be an application relevant to more than ride-sharing with personal vehicles. Taking university sports away games as a potential use case, the Benza application could be adapted to pair teams that are travelling to the same university. After pairing, the application could interface with the teams' student union (or relevant governing body) to book transport, thereby eliminating paperwork, minimising fuel and vehicle hire costs and bringing university clubs together to strengthen ties within the student body.

The demand for geographically anchored team coordination is increasing as traditional centres for work and education develop remote-access capabilities. If the carpooling-specific features of the application were to be removed, it could be viewed as a location-dependent instant messenger. Different modules could be created to offer different types of functionality within the app. For example, if a routing feature was replaced with a structure representing a curriculum's course and content, the application could be used by academic institutions to create groups of students for group projects. This pluggable behaviour is encouraged by the application's core language, Flutter, which treats every piece of the mobile app as an interchangeable widget [20].

Trust. The development team aim to complete a slow and steady initial release, conducting extended beta testing at the University of Aberdeen before expanding to other UK markets. This extended testing period will provide the feedback necessary to improve Benza's UX, which is difficult to gauge without a diverse pool of users. The limited initial release is an introduction technique that the team aim to take advantage of when moving into new markets. While rapid and open expansion brings undoubted benefits to startups, solidifying Benza's reputation and curating an audience that delivers constructive criticism will take priority over market impact during initial release.

4 Requirements

Requirements were specified during CS3028 and were informed by the guidance of in-course practical material. Adopting an agile process has resulted in numerous reiterations of the requirements throughout our cycle of analysis, development, testing and evaluation. The main cause for reiteration came from evaluating use cases, which informed the team about how effective the requirements were /in capturing our vision for the application. Further reiteration arose during implementation, where new issues were highlighted, especially in terms of technical feasibility. These changes were what led to the version of the requirements included in this report.

This section will guide the reader through an in-depth analysis of the pre-release version of Benza, focusing on the reasons why each requirement was added and the impact of the requirement on the application's User Experience, but also on details about why some requirements were prioritised and others were dismissed or deemed unfeasible . For more information on implementation barriers and obstacles found in the process, see Section 6.2. The dismissed features will be retained for potential development in any future releases of Benza.

4.1 Functional Requirements

The requirements in Table 1 pertain to the main use case of the system: how users will be able to sign up, log in, view the list of groups and join them to get to the end-goal of coordinating with other users for a carpooling trip. The outlier requirement for *rating another user in a group* might not appear to be a core part of the application, however trust is a vital aspect of all community-based platforms, especially so for platforms where users eventually meet in real life [13].

Base User Requirement	After joining a Group
Sign-up with unique User details	View list of Users in a Group
Log-in with unique User details	Chat with other Users in a Group
View User profile details/rating	View profile of other Users in a Group
Change User details	Rate another User in a Group
View list of Groups	Leave Group
Join a Group	

Table 1: Requirements for Base Use Functionality

Sign-up & log-in: A user will be able to sign up and consequently log in with their details. The required details are an e-mail that hasn't yet been registered (validated to match $x@y.z$), a password with a minimum of six characters and a name. If a user forgets their password, they will be able to reset their password from the login page using e-mail verification.

User details: Once logged in, a user will able to view and personalise their profile, changing details such as their name, profile picture and bio (maximum 50 characters). These details will

also be visible to other users within groups and a thumbs-up or thumbs-down widget will be available to these group members so that every user can be assigned a rating.

View & join groups: A user will be able to view a list of all existing groups, and view details such as the group's name and a Google Maps preview of the group location. From this list, a user will be able to explore, join and leave groups.

Group creation: A user will be able to create a group using a dedicated page. A unique group name and a valid location are the only prerequisites for creating a new group. A location is the name of the city and country, but can be further specified by adding a street name and postcode. A preview of the location will be displayed to ensure that the user has entered their desired location before letting the user to confirm the creation of the group.

Group details & chat: Once in a group, a user will be able to the view the details of the group together with a list of the users who have joined the group. Users in a group can also access and use the group chat to communicate within the group, as well as select a specific user to view their profile.

Trip User Requirement	After offering a Trip
View Trip details	Change Trip details
View list of offered Trips	Trip Moderation
Offer a Trip within a Group	Take Payment for Trip
Pay for Trip	Complete Trip
Leave Trip	
View list of Completed Trips	

Table 2: Requirements for Trips Functionality

Table 2 lists the extended requirements, complete with a payment and trip management system that can be customised to suit different dates, times and start/end locations. This data can be used to generate routes and schedules for any given trip.

Another important requirement worth noting is *Trip Moderation*, which entails the ability of a user to further manage the 'small' community they have created. This was meant to be applied to both groups and trips, but was later restricted to allow the trip offeror to have more control over trip participants and trip details.

These features were originally part of the functional requirements (Table 1) but have been abstracted to this second level of functionality as iterative analysis, implementation and evaluation prioritised other requirements and made some less feasible.

4.2 Non-functional Requirements

The non-functional requirements are essential to the quality of the UX. So, to structure and categorise the different requirements the team decided to use the FURPS+ model of classifying software quality attributes [10]. A thorough explanation of the non-functional requirements

(Table 3) will follow.

Category	System Requirement
Functional	See Tables 1 and 2.
Usability	Provide offline access to FAQs and T&Cs. Rank groups according to geographical proximity. Notify users of significant in-app events
Reliability	Dynamically adapt to volume of user activity[2].
Performance	Execute Group and Trip queries within 4 seconds of user request. Have the capability to serve at least 1,000 users simultaneously.
Supportability	Support modular load balancing. Run without fault on Android 6.0 – 9.0 and iOS 10.0 – 12.0. Provide a secure method of payment and credential storage[3].

Table 3: Non-functional requirements, based on FURPS+ Model

Usability. Making the user comfortable with using our application and ensuring that helpful information is readily accessible at any time is essential to success after the initial release. Hence why Benza's frequently asked questions and terms and conditions are made available to users even when the they do not have an internet connection.

To make the process of finding an appropriate group more streamlined, the requirements include a point about ranking groups according to geographical proximity. For this purpose, an algorithm was planned during the analysis phase, to compute a ranking value for each group based on the user's request. The groups best suited to the user would have then be displayed in decreasing order for the user to join.

Notifications are an expected feature from current mobile applications and would be a useful tool in the capabilities of any application containing a chat. In our application, a user would receive a notification whenever relevant changes happen, such as a new message in the groups the user is part of, or when details are updated. These could be the location or the time and date of a group, or a user joining or leaving a group.

Reliability. Startups can be required to scale incredibly quickly if their application gains significant traction after initial release. This rapid growth in traffic can cause unforeseen strain for application infrastructure and can lead to problems if the management and the product is unprepared [31], including: stalled growth, damaged reputation and even loss of user data. To safeguard Benza against such unfortunate circumstances, the application uses the containerisation platform, Docker. Docker will enhance Benza's reliability by adjusting to the volume of traffic through built in load balancing features, where any strained services can be virtually replicated to share the burden of increased traffic.

Performance. Today's mobile applications provide users with on-demand access to a massive range of resources in seconds. As the tech industry continues to boost users' expectations with

[2]Using load balancing with service replication
[3]In accordance with Payment Card Industry Data Security Standard (PCI DSS)

regards to convenience and speed, it is very important that Benza's users get responses to their queries within a reasonable time limit. Erring on the side of caution, the Benza mobile app will have a maximum wait time of 4 seconds for any operation. Although this is a long time to wait by today's standards, multiple test queries have proven that the application is usually well within this upper bound and never in excess of it (Section 7).

For Benza to reach its full potential, it will need a large and distributed user base. Working in tandem with Docker, Firebase's Cloud Firestore (Benza's storage solution for user data), boosts performance by providing support for more than 1000 simultaneous reads (Section 5.2). However, in this pre-release version of Benza there are restrictions placed on the daily number of reads and writes that can be performed in application's Firebase space. This performance ceiling is a direct result of the restrictive lack of funding available to the development team.

Supportability. To take full advantage of Docker's load balancing feature, Benza's architecture will follow the microservices pattern. In accordance with the pattern, Benza's various components will be split into distinct services, which will in turn allow Docker to provide load balancing specifically for those services that are under strain. The modular characteristic will minimise the system bloat created by whole-system replication when only one service is under strain.

Further to the requirement specifying compatibility with mobile operating systems (Table 3: Supportability), Benza eventually aims to run smoothly on the latest versions of both Android and iOS. This non-functional requirement was updated following the release of iOS 12.0, while the Android version compatibility remains unchanged. Currently, Benza runs without fault on Android 8.0 Oreo (Section 7) and the development team aims to port the application to iOS immediately after the Android beta testing phase.

The initial application proposal included a payment method so that users could contribute to the cost of a journey's fuel. The development team has since realised that this functionality is not essential to the application and could be reserved as a feature to be potentially added at a later stage. Moreover, the implementation of an in-app payment feature raised concerns about potential security issues handling the users' financial details. Another possible issue is that including payments in the software itself could wrongly encourage users to consider the ride as a job opportunity, which is not in harmony with the social nature of the application. Lastly, there are risks of running into regulatory obstacles, since Benza could then be lawfully considered as a taxi service. The industry has strict legislation that protects the traditional ride-hailing companies such as the local Rainbow City Taxis [REFERENCE].

4.3 Feasibility Study - MAGNI/LORI

In order to analyse whether the project can be fully developed in the time-frame given and if it is worth developing, our team carried out a feasibility study. This considers different aspects of the project, including the economical and technical side.

4.3.1 Technical Feasibility

A detailed risk analysis was carried out in the first semester and will be summarised here.

4.3.2 Operational Feasibility

4.3.3 Schedule Feasibility

4.3.4 Economic Feasibility

5 Design

5.1 Current System Characterisation

Benza was designed using three different software architectures: 3-Tier, Model-Viewer-Controller (MVC) and microservices. Figure 5 clearly illustrates the aforementioned three tiered structure and the individual modules that adhere to the microservices pattern, while the characteristics belonging to the MVC pattern are obfuscated for brevity.

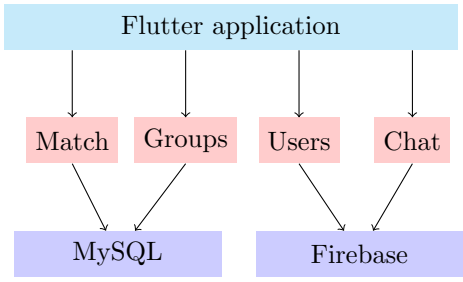


Figure 5: Benza's simplified architecture

The presentation layer (top) contains the Flutter application, which has an internal structure resembling the MVC pattern (Appendix C) and handles the user's interaction with the system. The microservices are shown in red in the application layer (middle); these services work independently and are called upon individually depending on user actions. The databases (bottom), are accessed by methods located in the microservices above and are the only locations where user data is stored in the system.

3-Tier: Benza uses the 3-Tier software architecture because it is an adaptable pattern that can accommodate various other architectures, while also providing a tried and tested high level system architecture that ensures minimal coupling between storage and presentation layers. Although this pattern provides a good base template for assigning responsibility within a system, incorporating microservices led to some unforeseen alterations. Since each microservice contains its own Data Access Object (DAO), the DAOs that would traditionally appear in the storage layer are bundled into microservices that are better suited to the application layer. These DAOs access data held in the inferred database layer. In the case of the *Groups* service (Figure 5), the Data Access Object (DAO) will access data that is held in a MySQL database (Section 6.2 discusses how this is a microservice in itself). The structure and organisation of Benza's data is discussed further in Sections 5.2 and 5.3.

Microservices: By incorporating the microservices pattern into Benza's business logic, the system's modularity is improved and its ability to deliver consistent service in hand with Docker is maximised. The sfubmodules contained within each microservice are highly cohesive, performing operations that are pertinent only to the purpose of the parent microservice. (submodules, modularity, minimising external dependencies, high scalability, keeping it cohesive)

MVC: The Model-Viewer-Controller pattern was chosen for its ability to define the flow of control within an application. By dividing responsibility for certain operations between distinct classes and files within the Flutter app, the system benefits from improved modularity and abides by the "principle of least privilege"[9]. The inherent modularity of MVC increases implementation speed because programmers are able to locate program features in the file structure.

5.2 Data Organisation

Benza's microservices architecture allowed the team to choose the most fitting storage solution for the implementation of each module. The decision to distribute the data into two different types of databases: a MySQL relational database for storing the data relating to groups and a cloud-hosted NoSQL database for storing user data.

The NoSQL database is hosted by Google's Firebase platform and was chosen for being easy to set up and maintain, as well as offering additional useful features such as secure user authentication [17] and real time data synchronisation. These additional features made it the ideal solution for storing sensitive user data and implementing Benza's group chat functionality. Because the data in Cloud Firestore is synchronised in real time, all connected clients are informed about or able to see the changes to the data that they have access to. By executing one-time fetch queries, Firestore allows to see the result almost instantly [16]. Security rules are set by the database host (Firebase) for datatype validation and user access restrictions. For example, only users who have joined group x are able to see offers and messages posted in group x.

However, implementing Benza's matching and group management services required a different approach for storage and retrieval of data. This part of the software is what makes Benza unique, ensuring complete control of these microservices and their data is essential and cannot be achieved with a cloud hosted database. Instead, a relational MySql database was the solution of choice because it is open source, scalable (capable of handling almost any amount of data, up to as much as 50 million rows), secure [23] and widely used within the software engineering community [30].

Distributing data between two different hosts rather than relying entirely on one prevents the system from having a single point of failure, enhancing the reliability of the system. In the case of MySQL database not responding, certain functionalities of the app would still be working as usual, such as users' ability to register, log in and see their profile. This is due to all the necessary data to provide these services being stored in the Cloud Firestore database. Only the functions that require access to group data would not be functional, as the data necessary for those operations would not be available. In the case of a failure within the Firebase-hosted Cloud Firestore, users would not be able to log into or sign up for the app, but the important group data residing in the MySQL database would be kept safe.

Using two databases also reduces the cost of maintenance and supports scalability, which is very important for a startup like Benza. Firebase is taking a lot of responsibility for the data management, allowing the development team to primarily focus on managing the MySQl database which stores the data needed to provide the core service of the app and is less flexible in terms of change. Storing all data in the MySQL database would require specific expertise and constant maintenance, which would result in increased costs of running the application. With regards to scalability, both databases are capable of expanding according to the needs and growing amount of users of the application. Firestore is capable of running up to 1 million concurrent connections, while the MySQL database can store up ... of data depending on...[REFERENCE]. However, in order to keep the cost of running the application as low as possible and comply with a tight startup budget, the amount of daily read, write and delete operations in the pre-release version of Benza is limited to 50 thousand reads, 20 thousand writes and 20 000 deletes.

5.3 File & Database Structure

Having two distinct hosts responsible for the storage of Benza's data, result in data being structured in a specific fashion and requiring different querying approach as each host possesses its own data model. The data in the Cloud Firestore database is organised into collections and subcollections consisting of records in the form JSON data format. While the traditional MySQL database consists of tables, columns representing the attributes (different types of information stored) and rows being the data stored.

The Cloud Firestore database is responsible for storing user data and messages sent within the groups. In the database, data fields are stored in artefacts called documents, which belong to a collection [18]. There are two distinct collections: Users and Chats. Each document within the Users collection represents a single user and consists of five fields: bio, email, name and UID (Figure 6). Users' profile photos are stored in a separate storage bucket provided by Firebase and referenced in the Users collection by means of a URI. The document can be retrieved by its name which is the UID. The Chats collection is structured in a similar manner. The difference being that within each group in the collection, there is another subcollection for the list of messages, where each message is a document. Every message contains five fields: message content, UID of the sender, name of the sender, timestamp of the moment of sending and message type.

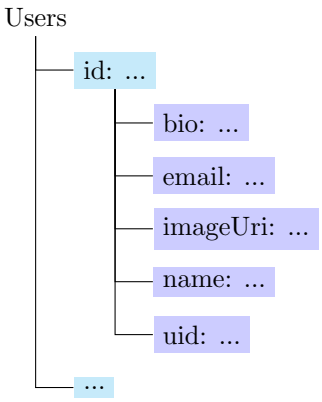


Figure 6: The Users collection in the Cloud Firestore database

The data relating to groups is held in a single table in the MySQL database. Every record in the group table has columns for the group ID, group name, group location and the users that have joined the group. The data types of these fields are set by the Flask API when the database is initialised and the table is created (during Docker startup). It is important to note that once the application is released, the Docker container will be available 24/7 and by extension, so will the database. The majority of data stored in this database is typed as SQL VARCHAR, with unspecified field lengths (with the exception of group IDs, which are stored as SQL INTEGER). This implementation decision is explored further in Section 6.2.

The file structure of the system (Figure 12) reflects the microservices architecture that was proposed in Benza's initial white paper. The Group Management Service, Notification Service and Matching Service are all structured as distinct collections of files, a characteristic that enhances Docker's ability to replicate services that are under significant strain and thereby increase load-balancing efficiency.

5.4 User Interface - MAGNI

The application is comprised of six separate views: a log-in/sign-up form, a user profile page, a group creation form, a page containing the list of groups and lastly the group and trip view pages. The log-in/sign-up form serves as the landing page of the application, and after logging

Figure 7: The bottom navigation bar

in or signing up, the rest of the pages become available through the navigation bar consistently displayed at the bottom of each view except for groups.

The navigation bar has a minimalistic blue and white design, containing four buttons, one for each page. Navigating to a page will alter the bar to display which view the user is in (for example, in Figure 7 the user is in the list of groups page).

The main interface of the application is built using a scaffold of three containers: the header containing the title and a sign-out button, the content containing the relevant widget (depending on which view the user is in), and the footer containing the navigation bar. This is the layout used both in the landing page (excluding the navigation bar), and the pages accessible in the navigation bar.

Once a group is selected or created by a user, a new view is displayed using a , the group's vie. It consists of a map preview in the header, with the group details underneath (name and location), and a group chat at the bottom. It does not include a navigation bar, as it would clutter the view and hinder the user's ability to use the chat. This comes from the KISS principle (Keep It Simple Stupid), only allowing the bare necessities to be displayed for the user to view and interact with, making the application optimal and easier to use.

Even the colour scheme of the interface followed this same idea, as only three colours, blue, white and black, were used in the interface. This keeps the design consistent with minimal changes between the views, except for the chat, that will have users with customisable profile icons.

5.5 Modifications to Design - LORI/MAGNI

Don't discuss things that we *aren't* doing here - rather discuss things that we set out to do one way but have ended up doing differently. This is a good place to draw on material from the requirements.

There's also lots of opportunity for linking back to the feasibility study, pointing out whether the risks that we anticipated there actually caused us to change the design (or not).

6 Coding and Integration

6.1 Planning

Planning how the components would be implemented took a multitude of factors into account. The coupling of some back end system components determined the order that parts of the project could be implemented in. For example, the app's login page (Figure 8a) depends upon several other features being functional: the device running the app needs to have internet access, there must be an online Firebase space with the Firebase Authentication tool activated and there must be a corresponding record in the Authentication tool's storage for the login data to be validated against.

Priority according to the requirements analysis was another factor: the software should be implemented in an order that fulfilled the base use requirements (Table 1) before the trips feature (Table 2) and the remaining non-functional requirements (Table 3). This structured approach is clearly illustrated by the Gantt charts that the team used to plan and record the order of implementation (Figures 1, 14, 15).

Like the coupled back end components, there were elements of the Flutter mobile application that needed to be implemented in a certain order to produce meaningful test outcomes and the desired system behaviour. These front end components follow what one might call a *flow of possibility*, where features that implement a specific functionality build on each other to fulfil a requirement. For example, to fulfil the *user should be able to create a group* requirement, a model that represents the internal structure of a group needed to be implemented. Modelling custom objects that have attributes of various data types presented its own challenges further into the implementation phase, when the objects needed to be passed to an API in JSON format and decoded from that format upon receiving a response.

Creating a template for the system that adhered to the architectural patterns decided in the design phase (Section 5) was an important first step in planning implementation. The Flutter application was initialised with a comprehensive "quick start" structure, providing a logical space for the team to implement the MVC pattern. Each of the microservices used boilerplate code from the Flask RESTPlus documentation [21] to provide a generic API with simple CRUD capabilities. Each microservice's Data Access Object borrowed elements from the Dataset micro-framework documentation [26] to provide a base functionality that complimented its Flask API.

6.2 Key Implementation Decisions

Trips. To not implement trips before releasing Benza was a turning point in the programming process. To understand why, let us briefly recount what a trip is. Every trip starts as an object with several attributes of different data types (similar to the groups discussed in Section 5.2). Creating nested fields for group objects in the API was an arduous task that overflowed its allocated programming time and eventually resulted in using flattened lists to represent a single field containing a list of values. This hack produced the application behaviour that was necessary to fulfil the base use requirements (Table 1) but greatly complicated the task of integrating the trips feature in the process.

The decision to discontinue programming efforts for the trip feature was made after significant time was lost to troubleshooting and the time allocated to other features began to shrink dangerously. Fortunately, the architecture template produced as part of the implementation plan has dedicated structures intended for this feature and should enable the team to resume work post-release.

Notifications. The decision to not fully develop the notifications feature was taken after the team experienced setbacks late in the implementation phase. This feature was listed as a non-functional requirement and as such, was allocated programming time in the later stages of implementation. Despite following the planning strategy set out in Section 6.1, this has led to Benza being released for beta testing without a notifications feature. However, as well as being the main cause of failure to meet this requirement, the planning strategy is also what will allow the development team to return to future development of the feature with tempo. This is due to the complete system template being laid in accordance with the chosen architectures before implementation of the business logic even began, providing a boilerplate Flask API for each of the microservices (Figure 12).

```
1     group_ns = api.namespace('group', description='CRUD for groups')
2     @group_ns.route('/') # The base route of the web application
3     class Group(Resource):
4         @group_ns.doc('Gets all groups')
5         def get(self):
6             return DAO.get_all(), 200 # Accessing the DAO
```

Listing 1: The Flask API's getter for the list of all groups from `/benza/group_service/app.py`

The decision to structure the microservices as RESTful APIs with web application interfaces provided the team with a simple and intuitive interface for use during the testing phase (Section 7). The decision to abstract the DAO class out of the flask API file improved the modularity of the microservices by separating each service's business logic from its Data Access Object (DAO). The handover in responsibility from the Python application to its Data Access Object can be seen in Listing 1:6. In the listing, a hint of the structure for the web application can also be gleaned: each of the getters defined in the Python file are referenced by a relevant namespace (route). The method to get a list of all the groups in the database is defined in the base namespace (Listing 1:2), so it will be one of the methods that are executable without the client sending any plain text data in the HTTP URL request.

```
1     class GroupDAO(object):
2         def __init__(self):
3             # Specifying database and table that the DAO should connect to
4             self.db = dataset.connect('mysql://root:not_so_secret@mysql_db/group_db')
5             self.group_table = self.db['group']
6
7         def get_all(self):
8             result = list(self.group_table.all()) # Returns a JSON list of all groups in MySQL db
9             return result
```

Listing 2: The group service's DAO structure from `/benza/group_service/GroupDAO.py`

Once responsibility is passed to the DAO class, it needs to do several things: firstly, initialise a connection to the MySQL database (Listing 2:4). This locates the database, which is hosted as a service by Docker, before selecting the singular table in the database (Listing 2:5). If this table is not yet present in the database (the Docker service has been completely restarted and the tables dropped), then Dataset[2] will automatically create a new table named 'group'. After the GroupDAO class object has been initialised, the system can access its internal methods. Listing 2:8 shows how the GroupDAO uses the `.all()` function that Dataset provides to operate on all the rows in the 'group' table.

```
1    services:
2        group_service:
3            build: ./group_service
4            depends_on: - mysql_db
5            ports: - "4100:4000"
6            volumes: - ./group_service:/group_service
7            restart: on-failure:5
8        ...
9        mysql_db:
10           image: mysql:5.7
11           ports: - "32000:3306"
12           volumes: - ./db/mysql:/docker-entrypoint-initdb.d/:ro
13           environment: MYSQL_ROOT_PASSWORD: not_so_secret
```

Listing 3: The application-level `docker-compose.yml` file structure

To provide access to the MySQL database and avoid the hassle of having to manage a server, the team decided to provide it as a service using Docker. The most tangible benefits of this decision were the ease of testing the database-dependent features (Section 7.1) and the fine-grained control and feedback provided by the Docker console (Section 6.4).

To demystify the process by which Docker provides containerised services, snippets from the application-level `docker-compose.yml` file and the service-level `Dockerfile` are included (Figures 3, 4). The `docker-compose.yml` file is essentially a recipe that Docker follows to provide all the services as individual containers. Listing 3:4 instructs Docker that the group service container needs the database service declared in line 9, so that the service does not attempt to start up before the databases are online.

This is reflected in the group microservice's `Dockerfile` (Listing 4:2), where the tools to manage and build the database are initialised. Following initialisation of the database, lines 7 and 10 launch the service as an API, which is the interface that the Flutter mobile app will use to interact with group data in the MySQL database.

By specifying dependencies for each service in their own `Dockerfile`, the application avoids the risks associated with outdated versioning and ensures that all instances of Benza are using an identical back-end.

[2]Python library that Benza uses to connect to the MySQL database

```
...
# Initialise database dependencies
RUN apk add py-mysqldb
RUN apk add --no-cache mariadb-dev build-base
...
# Make port 4100 available to the world outside this container
EXPOSE 4100
...
# Run app.py when the container launches
CMD ["python", "app.py"]
```

Listing 4: The group service's `Dockerfile` structure from `/benza/group_service/Dockerfile`

6.3 Libraries - DUNCAN

The following are code libraries that have been used during implementation to avoid reinventing multiple, rather complex, wheels.

Firebase Suite. Continuing with Benza's theme of Google development tools, the team elected to use Firebase as the storage solution for all data relating to the application's users and the group chat feature. Although it is not technically speaking a 'library', Firebase has been too important to Benza's development to be considered an 'other resource'. The Suite supports the application with two useful tools:

- Authentication: Firebase Authentication supplied a mature user management infrastructure to the development team and fit intuitively into the Flutter application. With aesthetic methods to access the authentication API, it also improved readability for all the team members.
- Cloud Firestore: A cloud-hosted NoSQL database, providing data synchronisation and intuitive management through a friendly web-based console.

Google Maps Android SDK. When the team released their initial technical report on Benza at the end of last semester, the application used the free-to-use OpenStreetMap resource to display map tiles relating to latitude-longitude pairs. In December 2018, this was the best solution for integrating maps to Flutter applications. However, the Flutter Development Team announced the Developer's Preview version [19] of the `google-maps-flutter` plugin on March 6th of this year and the Benza development team adapted to take advantage of what was sure to be a rapidly improving new technology. The new plugin allows Benza to use Google's Maps for all its mapping features and anticipates growing capabilities from the plugin in the months and years to come.

Flask RESTPlus. Flask is a widely-used and simple library that facilitates easy setup of APIs. The Flask RESTPlus micro-framework is designed around Flask and focuses on streamlining the process of creating a RESTful interface and encouraging good RESTful practices [22]. Benza uses the framework in conjunction with Docker to create and run its microservices. An added benefit of this particular framework is the Swagger documentation and UI that it generates automatically. The UI is available via the base URL of whichever API is being accessed and

provides a method for testing the methods in the Flask-like code base.

Dataset. To interact with the containerised MySQL database, Benza uses the Python framework called Dataset. Dataset provides a way for Python programs to interface with SQL databases in a way similar to interacting with a NoSQL store or JSON file [26]. As a result, the development team avoided writing complicated SQL queries, saving developer time and creating code that was easier to read and maintain.

Built-in Dart/Flutter libraries. Since Flutter is at heart a framework based on the Dart programming language, the mobile application makes frequent use of built-in Dart libraries. Some of the most prolific imports were:

- `dart:async`

  The Dart language has one thread of execution. To avoid blocking and unwanted program freezes mid-execution it lets developers declare resource intensive and time consuming operations as asynchronous. This means that developers can use an `await` statement to instruct the program to wait for the operation to terminate.

  Programs can also operate on variables that are declared as the `Future` of a certain asynchronous operation, so execution can continue while the operation is processed. This feature was particularly useful when interacting with the microservice APIs, which had some inherent latency due to internet connection speed.

- `http:http.dart` and `dart:convert`

  Both of these libraries were used to interface with the microservice API's. The main application for `http` was in setting up a persistent client object for an instance of the mobile application. Once the client object has been executed, the connection that it opens to a server will remain open to minimise the frequency of restarting the connection (which would use more mobile data then an open connection with no traffic).

  The data being transmitted over this connection consists mainly of JSON encoded objects, on account of the Dataset framework preferring that format. To transform the Dart objects that are produced by the application into JSON format, Dart provides a `json.encode(x)` method.

6.4 Other Resources - GRETA

The following are resources that have aided the team during implementation.

Docker

Docker is a containerisation platform that serves as a great asset in managing various services that make up the application. Each service is treated as a separate container possessing source code, settings and required libraries - all the things needed for the execution of the service. More importantly Docker runs all the microservices simultaneously ensuring needed co This results in minimising interdependancies within the system. Individual containers can be easily managed and updated without worrying how it will affect the rest of the services.

Moreover, use of Docker led to Benza being a portable application. Not only services are

represented as containers, but also Benza itself is a container, thus can be easily transferred and executed in any environment.

Also high scalability and performance of the application was achieved with the help of these Docker features:

- Load balancer: ensures that no service within the system is put under too much pressure. Struggling services gets virtually replicated in order to take overload preventing failure of the microservice that and the whole application.
- Docker console used as tool for tracking the performance of the microservices.
- Docker compose: executing multiple containers with a single command.
  - multiple isolated environments on a single host
  - preserve volume data when containers are created
  - only recreate containers that have changed

Flutter related resources

At the start of the development cycle team had no previous experience working with the very young application development framework Flutter. To get additional support when fixing issues or having trouble figuring out how certain functionalities should be implemented using Flutter team looked for help within the community of flutter developers.

- Boring Flutter Show: a set of videos that provided useful information on initial development of the app, type fixes and blunders. The basics of Flutter was explained by showing how to build an app from scratch while fixing encountered errors. Thus, team gained desperately needed knowledge on how to start the implementation and approach encountered problems.
- medium.com/flutter-community: website that offers a number of flutter design patterns and packages. Also contains articles posted by fellow developers including tips and advice on how to implement certain features and overcome issues. Gave valuable insights on how the application should be structured.
- flutter samples: an official, curated list of sample flutter projects on GitHub that is regularly updated by actual Flutter developers. Served as a source on new features and examples of Flutter which helped to make well informed implementation decision without reinventing the wheel.

Data related resources

As user data in Benza is stored in the form of JSON objects and information about groups is stored as a set of attributes with assigned values in the traditional MySQL table, a tool able to convert data from one form to another was a crucial resource.

- Quicktype.io: JSON parser generator was used for this purpose. It allowed to create group objects to and from JSON objects so that data could be written into the MySQL database saving huge amount of time for the team.

7 Testing and Evaluation

- Evaluation starts from the result of test data to discuss the quality and the limitations of what you have done so far. In particular, you should discuss how well your software fulfils the requirements (and here is where non-functional ones play a major role).

7.1 Strategy

A loosely defined testing strategy enabled the team to remain adaptable during the series of sprints that define the agile process. The low granularity of the implementation schedule (Figures 1, 14, 15) led to implementation of the small code units (which combine to deliver application features) being completed at the discretion of the team's programmers. As such, the testing for each feature in the schedule was conducted by that feature's assigned programmer during the feature's development.

The risk associated with this strategy was a lack of high level feature testing and a focus on unit testing. Fortunately for the development team, every component of the application has its own dedicated testing infrastructure that encouraged a high-level perspective of its functionality:

- Docker

  The Docker CLI tool provides information about all active containers when the application is launched using a `docker-compose.yml` file. The CLI also has a feature that lets the developer attach to individual containers. This feature is incredibly useful in that it adapts to the content of the container that Docker is attached to. If the container is a MySQL database, the console will feature SQL response messages and warnings.

- Flutter

  The Flutter SDK generates informative error messages in whichever terminal window executes the `flutter run` command. In its default configuration, the error messages display information about the error itself, the probable cause and where the failing widget is located in Flutter's widget tree.

  As a framework that prides itself on creating beautiful mobile applications, Flutter ensures that any errors impacting the mobile app's GUI are indicated to developers by way of graphical warnings in the emulator. Since the team had no experience with Flutter before this project, the graphical warnings would inform agile development and testing of UI features.

  Since Flutter is built on Dart, it inherits compatibility with the Dart VM Observatory. Dart describes its Observatory tool as "A Profiler for Dart Apps"[12].

- Flask RESTPlus

  The Swagger UI testing interface is integrated into Flask RESTPlus and generates a browser-based GUI for the API's exposed methods. This GUI let the development team test the microservices' capabilities without defining tests with harness code, saving time in coding and learning.

  Decorators are an element of Flask RESTPlus that automatically add details to the generated Swagger UI. An example decorator is `@api.expect()`, which tells the method exposed

to the GUI what kind of input data to expect. There is also an optional boolean parameter for validating the input data after it has been submitted: if the validator returns false, an error is thrown and execution halts.

- Dataset

  Dataset is built on top of SQLAlchemy to ensure compatibility across a range of common database types[3] but this design choice also gives error messages that are very informative. Since Dataset is used by the Docker-hosted microservices, an SQLAlchemy traceback became visible once a developer ran the `docker-compose up` command or attached to a container that was already running in the background.

Unit testing. If a feature that was not being worked on interrupted program execution after it was deemed to be complete, steps were taken to follow the flow of the program with regards to that feature, using the in-browser Dart VM Observatory. The Observatory enabled developers to place breakpoints in the Flutter application and jump between points of asynchronous execution once debugging halted at a breakpoint.

The mobile application relies on asynchronous operations to communicate with the Firebase and MySQL databases, so this feature was particularly useful in determining whether the exception was the result of other faulty elements.

Alpha testing. The alpha release of Benza will trialled with a group of 10 students from the University of Aberdeen who have past experience with carpooling, including hitch hiking.

Beta testing. As proposed in Section 3.2, initial beta testing of Benza is to be conducted with the University of Aberdeen student population. The group of fewer than 30 students that preferably have home addresses on the East coast of Scotland between Aberdeen and Dundee will be selected as participants. After a period of time spent collecting feedback from this pilot group of users, an extended beta will be released to people with `*@aberdeen.ac.uk` and `*@abdn.ac.uk` email suffixes. After a minimum period of 2 months spent testing with this extended audience, the first official release of Benza will be available to people with `*.ac.uk` email suffixes.

7.2 Test Data - DUNCAN

Re-imagined for each component.

Mostly passing different data types to other components to see how they react.

7.3 Harnessing Code - DUNCAN

Debug environment generates its own basic test code.

Built-in function calls will fail tests set in by their language if the software breaks.

[3] SQLite, PostgreSQL, MySQL and more [26]

7.4 Results - DUNCAN

Lots of type errors for Flutter app to API (and API to Flutter app)

Returning instances of futures in Dart

Rendering errors for UI

7.5 Evaluation - DUNCAN

Some important stuff goes here!

8 Conclusions and Future Plans

- If original client requirements were not satisfied - explanation of why.

8.1 Outcomes - LORI

The software that this report is released with is not to the standard that the team originally envisaged. However, there are also aspects of Benza that the team didn't believe were possible to attain at the beginning of CS3028.

8.2 The Future - DUNCAN

As with any new application, there are lots of features that the development team is still considering implementing. For some of these features it would be a matter of the team resuming development of from where it was discontinued, as mentioned in Section 7.5.

The Flutter Development Team is working on improving the Google Maps plugin, which will eventually feature fast routing on Google's existing Maps framework. With this capability, the maps already integrated in the Benza mobile application will become more suitable for use with the discontinued 'Trips' functionality.

The possibility of "pluggable" features described in Section 3.2 is an exciting prospect but also one that would be best reserved for when the development team has implemented the discontinued features detailed in Section 6.2. In the future, there may be opportunity for Benza to be opened to other developers as part of an open-source campaign, run in conjunction with the University of Aberdeen's Computing Society. Benza's comparably niche software paradigm makes it an ideal option for developers who would like to broaden their knowledge of mobile application development, a topic that is not included in the University's curriculum.

8.3 Lessons Learnt - GRETA/DUNCAN

The software engineering process has been an illuminating experience for the whole team.

Teamwork plays an important role in the software development cycle: team management was shaky at first and communication suffered as a result [EXPAND].

The team's overall experience of the development process would have been more positive had the members had some foresight about the challenges they would be facing. Since this is the first software development group project of this scale that the members had been involved in, everyone found it difficult to adjust to the dynamics.

Very young technologies can be very rewarding but present their own unique challenges: frequent changes and unfinished features lead to existing code becoming unreliable with little to no warning.

A small developer community and a limited amount of existing material leads to ambiguity when trying to ascertain the "right" way to implement a feature.

Appendices

A User Manual

A.1 Preamble

- Hardware Requirements:
  Any device capable of running the operating systems in the Software Requirements.
  The software was developed and tested on a Google Pixel, in an Intel Atom (x86) Android emulator with a 1080x1920 display, running Oreo (Android 8.1).

- Software Requirements:
  Your device OS must be Android, versions 6.0 – 9.0.

- Installation:
  Benza is coming to the Google Play Store in May 2019.

  If you would like to take part in the open beta, you can download Benza from Team Bravo's Google Drive using the link below. You can download the file to your computer or directly to your compatible Android device.

  Download APK

  If you have downloaded the APK to your computer, connect your Android device via USB and move the file to your device.

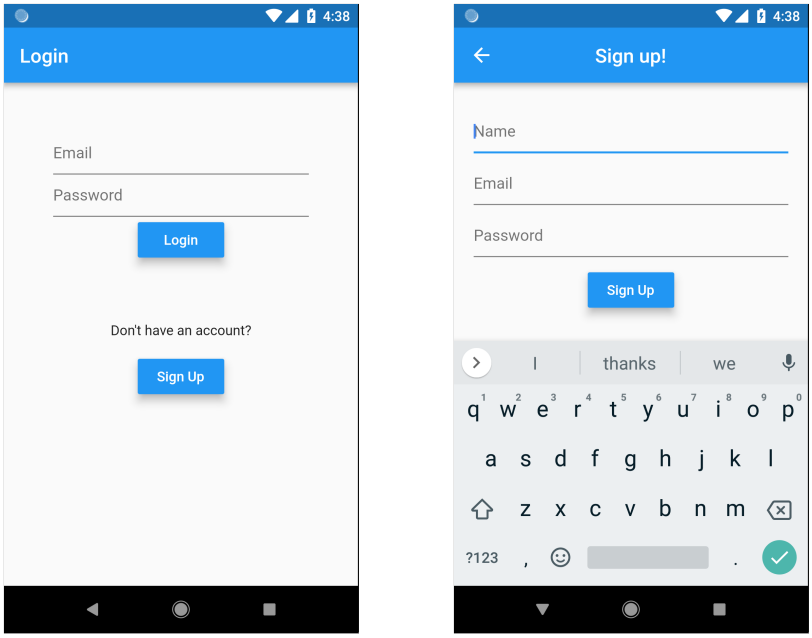
  When the APK appears on your device, you will have successfully installed the beta release of Benza!

  Let's jump right into it by tapping the Benza icon.

A.2 Signing Up & Logging In

You will be greeted by the log in screen (Figure 8a). If this is your first time using Benza, you won't have an account yet. To register an account, tap the *Sign Up* button.

If you have already registered an account, enter the details that you previously registered into the *Email* and *Password* fields before tapping *Login*.



(a) Log in screen

(b) Sign up screen

Figure 8: Initial sign up process

Now you can see the sign up screen (Figure 8b). To register an account, complete the *Name*, *Email* and *Password* fields with your own information.

If you ever forget your password, you can contact the development team [CONTACT] to update it to a new one. After notifying USA that you've forgotten your password, we'll send an email to the address you provided when you signed up, which will have link that lets you change your password (Benza will *never* store your password in plain text and it will *never* be readable by our team).

A.3 Your Profile

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean eget faucibus justo. Fusce bibendum commodo suscipit. Integer nisi ante, pulvinar eu condimentum non, iaculis et ante. Integer venenatis imperdiet lacus, quis eleifend augue hendrerit at. Etiam nec scelerisque ex, nec sodales eros. Sed eget orci dolor. Cras ex nibh, iaculis id odio dapibus, lobortis pharetra arcu.

Sed nibh risus, iaculis ut varius sit amet, commodo vel risus. Maecenas felis felis, congue ac posuere id, fermentum eget mi. Maecenas euismod, odio nec mollis lobortis, sem sapien ornare ligula, sed semper mauris neque non lectus. Nullam id arcu risus. Fusce eros felis, dictum eu bibendum at, faucibus eget urna. Nullam iaculis accumsan nunc. Aliquam tempus nulla eget luctus fringilla. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin consequat sagittis dui, sed convallis leo consequat ac. Aenean mattis placerat finibus. Phasellus aliquet dictum turpis at blandit. Suspendisse tincidunt, ipsum vel condimentum gravida, nisl elit venenatis neque, venenatis imperdiet erat sem ut turpis. Donec tempor ex leo, id rhoncus erat placerat a. Aenean posuere turpis vitae metus fermentum dapibus. Sed non fermentum orci.

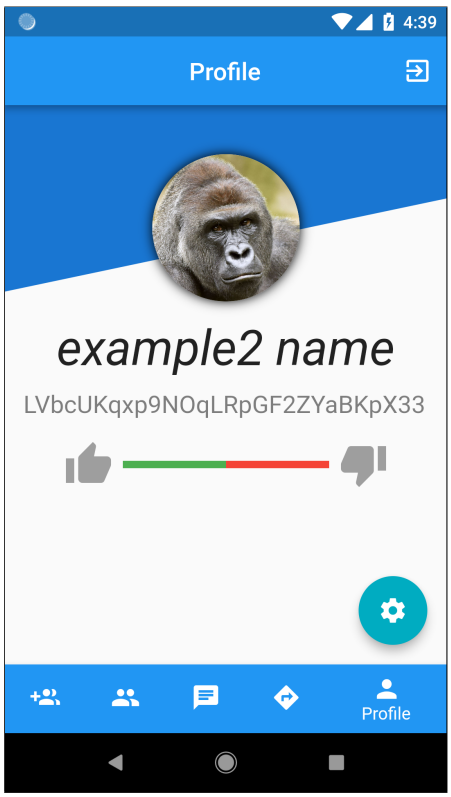


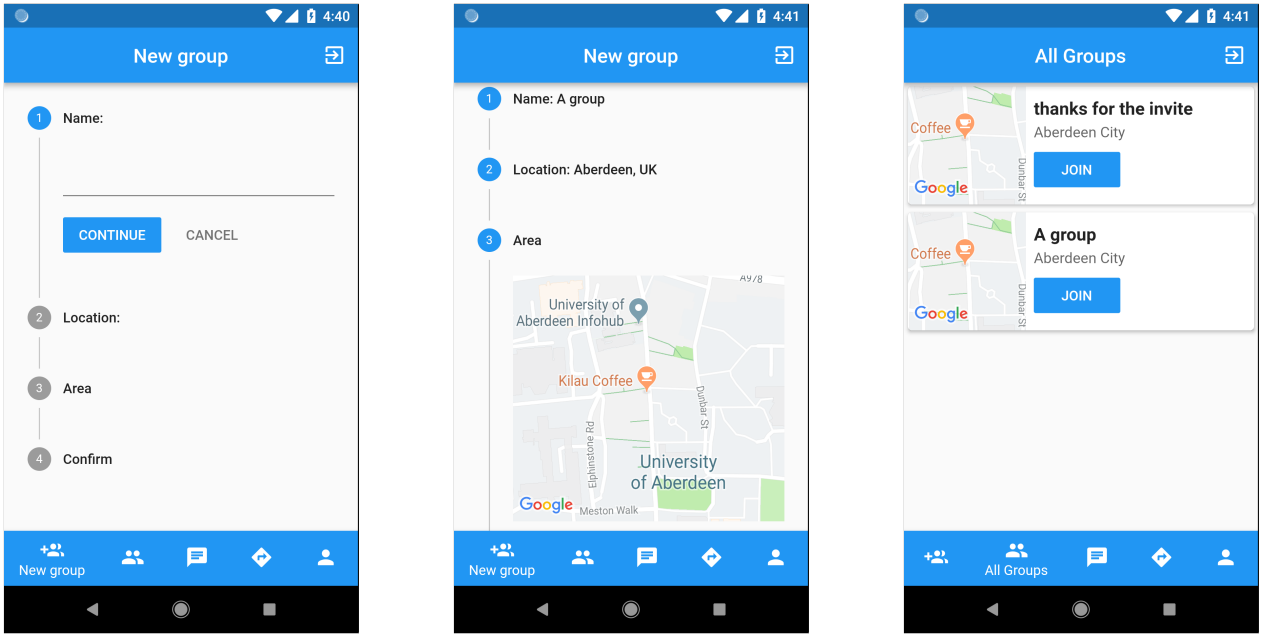
Figure 9: User's personal details

Sed nibh risus, iaculis ut varius sit amet, commodo vel risus. Maecenas felis felis, congue ac posuere id, fermentum eget mi. Maecenas euismod, odio nec mollis lobortis, sem sapien ornare ligula, sed semper mauris neque non lectus. Nullam id arcu risus.

A.4 Creating a Group

To start carpooling with Benza, you need to join a group! To start with, consider joining one of the groups that already exist close to you. To see a list of all these groups, tap the second leftmost icon in the bottom navigation bar.

If none of the groups in the list look like something you want to join, tap on the leftmost icon in the bottom navigation bar to start the process of creating your own group.

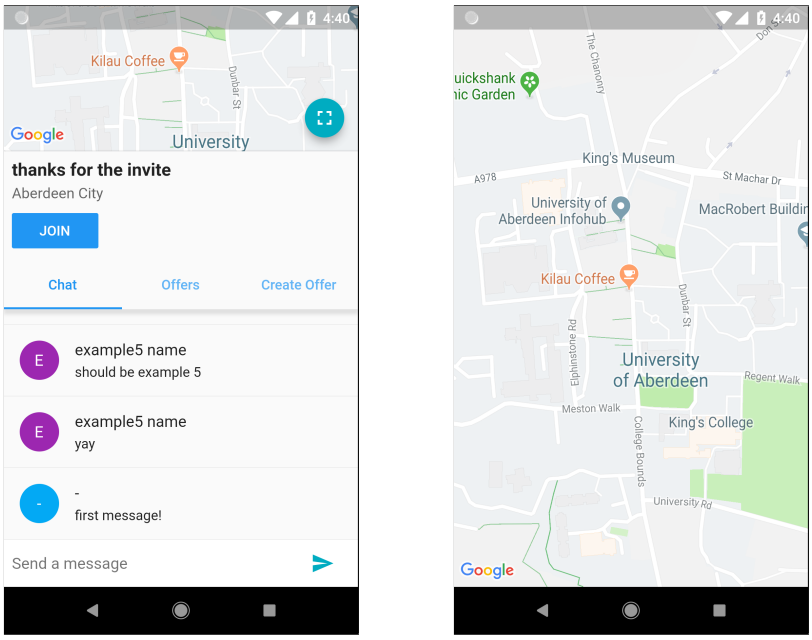


(a) Entering group details (b) Confirming group's location (c) List of all groups

Figure 10: Creating a group

Upon tapping the leftmost icon, you will see a screen setting out the steps required to create a new group, starting with the group's name (Figure 10a). Enter a name for your group and tap *Continue*.

A.5 Inside a Group



(a) Expanded group view.

(b) Full-screen location of group.

Figure 11: Communicating with users inside a group.

B Maintenance Manual

B.1 Getting Started

To get started, fork the latest version of Benza from the GitHub repository[4].

If you are unsure about how to fork a repository, read the GitHub article on how to fork a repository[5].

B.2 Minimum Requirements

- Hardware

  Your machine should have at least: 4-core CPU, 2GB RAM, 4GB free hard disk space.

- Software

  1. An IDE will make your development experience smoother. For this project, the team used Visual Studio Code with the Android Emulator.
  2. Install the appropriate Docker developer package for your OS. To complete the Docker setup, you will need to register your own Docker account.
  3. Install the latest stable version of Flutter.
  4. Install Python 3.7.
  5. Update pip on Mac OSX and Linux:

     ```
     pip install -U pip
     ```

     Windows:

     ```
     python -m pip install -U pip
     ```

B.3 Dependencies

1. Install Flask-RESTPlus:

   ```
   pip install flask-restplus
   ```

2. Install Dataset:

   ```
   pip install dataset
   ```

[4] github.com/nicomazz/benza
[5] help.github.com/en/articles/fork-a-repo

B.4 File Structure Overview

The folders (purple) and files (blue) that make up the Benza application (Figure 12) follow a very purposeful structure.

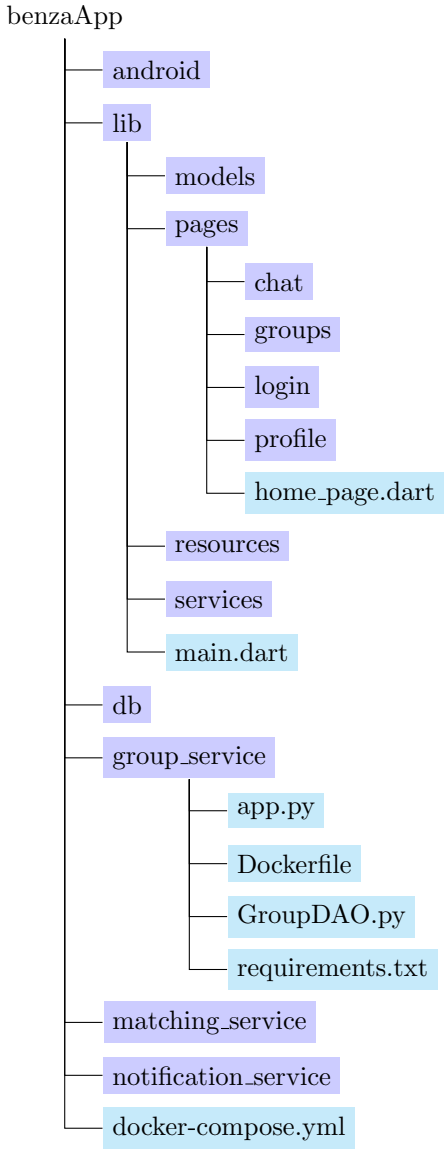


Figure 12: Structure of pertinent files

In the structure, the group service is the only microservice that is expanded, due to it being the most complete service and therefore the most representational of the finished application.

C Full Architecture

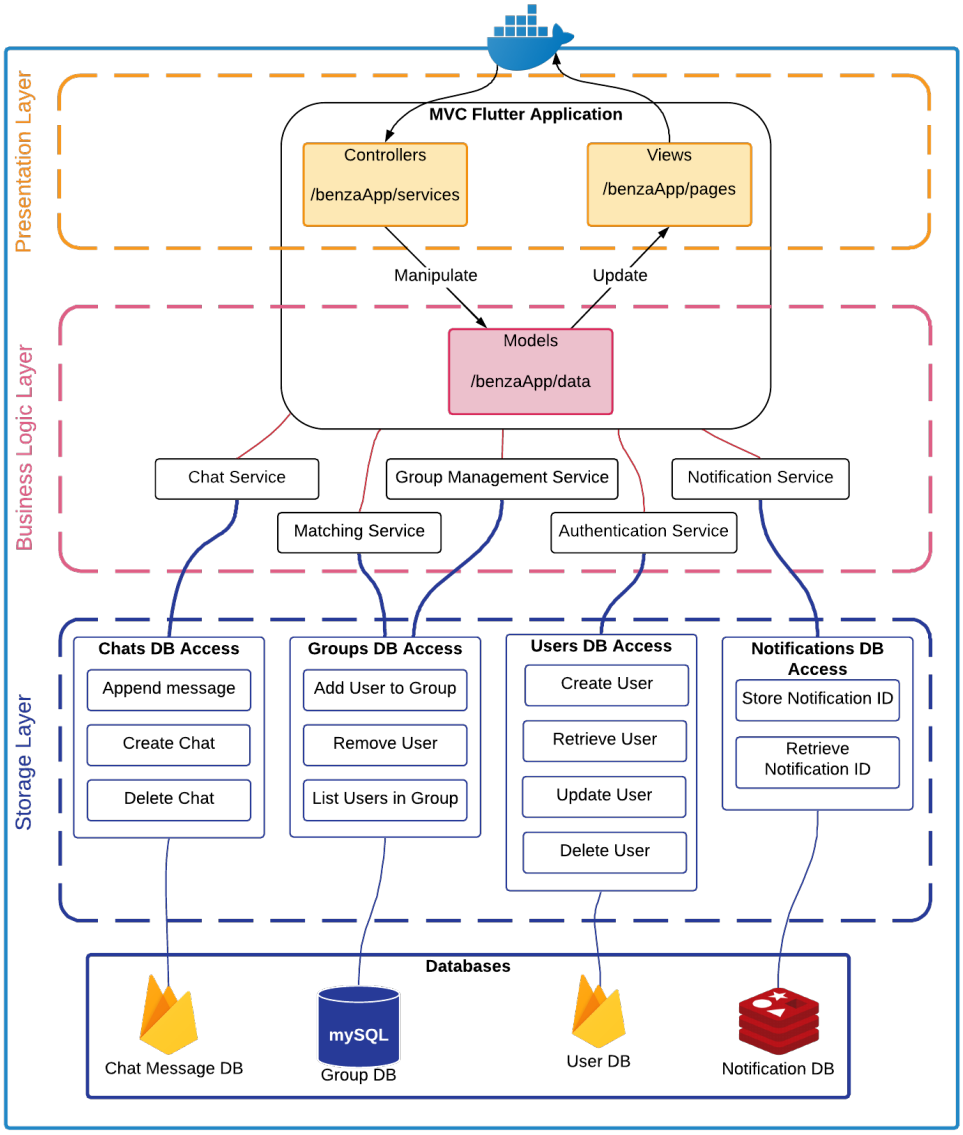


Figure 13: Benza's legacy architecture from CS3028

D Implementation Schedule

In Figure 14, the 'API Interfacing' bar contains milestones for when each of the CRUD features were made usable. You will notice that this means the application is missing the last element of the acronym, *Delete*. For more discussion about why this feature was not fully developed, refer to Section 6.2.

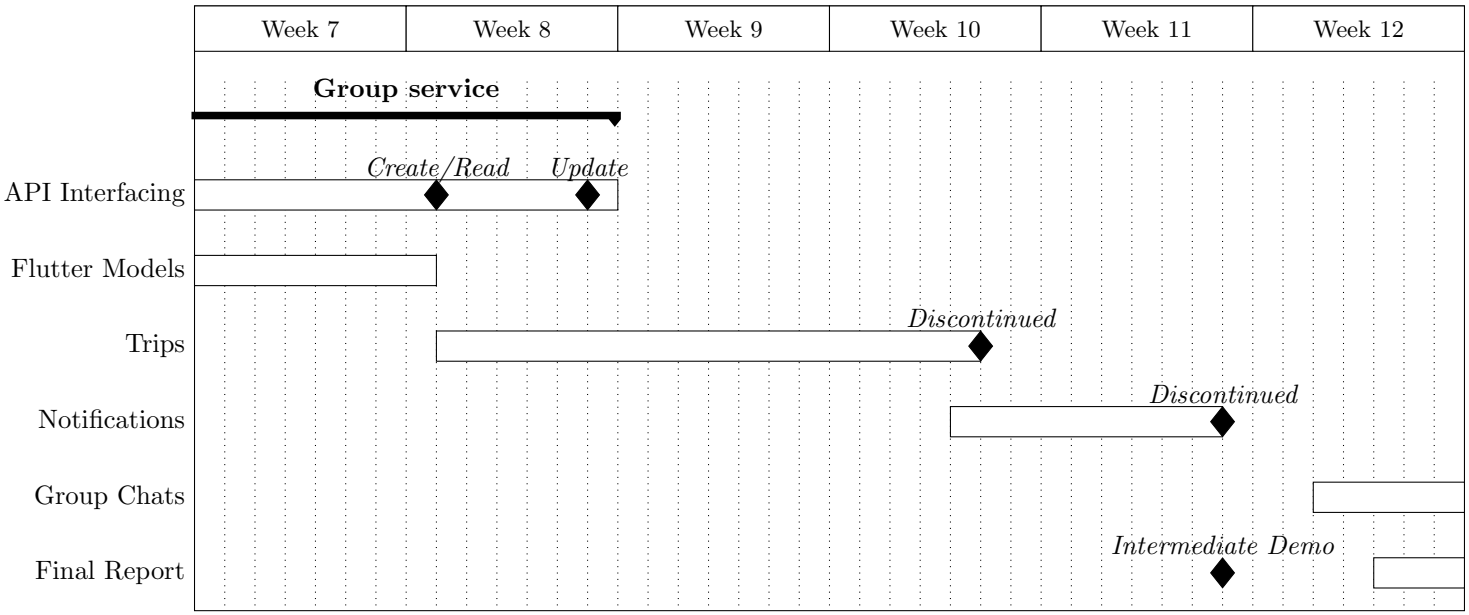


Figure 14: Development schedule from 25th February to 7th April 2019

The 'Notifications' and 'Trips' bars finish with a negative milestones. Due to complications in implementing the services and time pressure starting to mount, the features were discontinued so that work on other features could continue.

The intermediate demo of the application occurred in late March and the resulting evaluation of the current system characteristics was an influencing factor in the decision to discontinue the notifications feature.

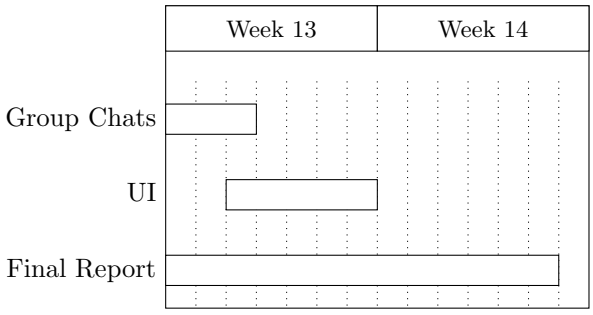


Figure 15: Development schedule from 8th to 21st April 2019

Glossary

3-Tier Software architecture pattern with presentation, application and storage layers. 17

app A mobile phone application. 6, 8, 10, 11, 21, 29

bio Colloquialism for biography - a concise description of a person. 19

boilerplate code that can be reused with minimal changes. 21, 22

bucket A totally flat storage space to hold an unstructured collection of data. 19

gamify Integrate game mechanics to motivate participation, engagement, and loyalty. 10

gantt A charted visualisation of a project's scheduled tasks over time. 21

getter A method to access data held elsewhere in the system. 22

microservice A single-function module with a well-defined interface. 15, 17, 18, 21, 22

pluggable Architecture where elements can be removed or replaced freely. 11, 30

RESTful A web service following the Representational State Transfer architectural pattern. 22, 24

traceback A stack trace from the point of an exception (error). 28

unicorn A privately held startup company valued at over $1 billion. 8, 10

Acronyms

API Application Programming Interface. 19, 21–23, 27, 29

APK Android application Package. 32

CLI Command Line Interface. 27

CRUD Create Read Update Delete. 21, 40

DAO Data Access Object. 17, 21–23

EU European Union. 10

FURPS+ Functional, Usability, Reliability, Performance, Supportability, +. 13, 14

GDPR General Data Protection Regulation. 10

GUI Graphical User Interface. 27, 28

HTTP HyperText Transfer Protocol. 22

ID Identifier. 19

IDE Integrated Development Environment. 37

IPO Initial Public Offering. 9

JSON JavaScript Object Notation. 19, 21, 25, 26

KISS Keep It Simple Stupid. 20

MVC Model-Viewer-Controller. 17, 21

NoSQL Not only SQL. 18, 24, 25

UI User Interface. 9, 27, 29

UID User ID. 19

URI Uniform Resource Identifier. 19

URL Uniform Resource Locator. 22

UX User Experience. 8, 11–13

VM Virtual Machine. 27

References

[1] M. Ahmed. *BlaBlaCar buys ride-sharing competitors to lock in European lead*. 2015. URL: https://www.ft.com/content/fbbd1528-e2ae-11e4-aa1d-00144feab7de. Accessed: 15.03.2019.

[2] BlaBlaCar. *BlaBlaCar acquires Russian carpooling platform Beepcar*. 2018. URL: https://blog.blablacar.com/newsroom/news-list/blablacar-acquires-russian-carpooling-platform-beepcar. Accessed: 15.03.2019.

[3] BlaBlaCar. *BlaBlaCar acquires urban carpooling company Less*. 2019. URL: https://blog.blablacar.com/newsroom/news-list/blablacar-acquires-urban-carpooling-company-less. Accessed: 15.03.2019.

[4] BlaBlaCar. *BlaBlaCar makes an offer to acquire Ouibus and secures €101 million investment*. 2018. URL: https://blog.blablacar.com/newsroom/news-list/blablacar-makes-an-offer-to-acquire-ouibus-and-secures-e101-million-investment. Accessed: 15.03.2019.

[5] BlaBlaCar. *Inside Story 3: Fail. Learn. Succeed.* 2019. URL: https://blog.blablacar.com/blog/inside-story/fail-learn-succeed. Accessed: 15.03.2019.

[6] A. Bogle. *Lyft's 'anti-Uber alliance' was quietly called off*. 2017. URL: https://mashable.com/2017/03/22/didi-chuxing-lyft-roaming-deal/?europe=true#XtpQ9reqXOqf. Accessed: 15.04.2019.

[7] D. Bosa. *Lyft claims it now has more than one-third of the US ride-sharing market*. 2018. URL: https://www.cnbc.com/2018/05/14/lyft-market-share-051418-bosa-sf.html. Accessed: 06.04.2019.

[8] J. Ciolli. *'They're changing the whole way that commerce works' — BlackRock's $1.8 trillion bond chief explains how millennials are spearheading an economic revolution*. 2018. URL: https://www.businessinsider.com/blackrock-bond-chief-rick-rieder-on-millennials-economy-waze-2018-4?r=US&IR=T. Accessed: 15.04.2019.

[9] E. Compatangelo. *L12 - Design Principles*. 2018. Accessed: 15.10.2018.

[10] E. Compatangelo. *L6 - From Inception to Elaboration*. 2018. Accessed: 11.10.2018.

[11] Crunchbase. *Series B Overview*. 2019. URL: https://www.crunchbase.com/funding_round/lyft-series-b--1b41c14e#section-overview. Accessed: 15.04.2019.

[12] Dart. *Observatory: A Profiler for Dart Apps*. 2019. URL: https://dart-lang.github.io/observatory/. Accessed: 19.04.2019.

[13] A. Dent. *BlaBlaCar: How We Built a 25 Million Member Strong Community Based on Trust*. 2016. URL: https://www.slideshare.net/crowdsourcingweek/blablacar-how-we-built-a-25-million-member-strong-community-based-on-trust/5. Accessed: 15.03.2019.

[14] R. Dillet. *BlaBlaCar is optimizing its service for small cities and has a new visual identity*. 2018. URL: https://techcrunch.com/2018/01/30/blablacar-is-optimizing-its-service-for-small-cities-and-has-a-new-visual-identity/. Accessed: 15.03.2019.

[15] R. Dillet. *Google Maps suggests BlaBlaCar for long-distance rides*. 2017. URL: https://techcrunch.com/2017/08/22/google-maps-suggests-blablacar-for-long-distance-rides/. Accessed: 15.03.2019.

[16] Firebase. *Choose a Database: Cloud Firestore or Realtime Database*. 2019. URL: https://firebase.google.com/docs/database/rtdb-vs-firestore. Accessed: 09.04.2019.

[17] Firebase. *Cloud Firestore*. 2019. URL: https://firebase.google.com/docs/firestore/. Accessed: 09.04.2019.

[18] Firebase. *Cloud Firestore Data model*. 2019. URL: https://firebase.google.com/docs/firestore/data-model. Accessed: 10.04.2019.

[19] Flutter. *google_maps_flutter*. 2019. URL: https://pub.dartlang.org/packages/google_maps_flutter#-versions-tab-. Accessed: 17.04.2019.

[20] Flutter. *Technical Overview: Everything's a Widget*. 2019. URL: https://flutter.dev/docs/resources/technical-overview#everythings-a-widget. Accessed: 15.04.2019.

[21] A. Haustant. *Quick Start*. 2014. URL: https://flask-restplus.readthedocs.io/en/stable/quickstart.html. Accessed: 08.10.2018.

[22] A. Haustant. *Welcome to Flask-RESTPlus's documentation!* 2014. URL: https://flask-restplus.readthedocs.io/en/stable/#welcome-to-flask-restplus-s-documentation. Accessed: 08.10.2018.

[23] JavaTpoint. *MySQL Features*. 2018. URL: https://www.javatpoint.com/mysql-features. Accessed: 09.04.2019.

[24] S. Levin. *Uber's scandals, blunders and PR disasters: the full list*. 2017. URL: https://www.theguardian.com/technology/2017/jun/18/uber-travis-kalanick-scandal-pr-disaster-timeline. Accessed: 15.04.2019.

[25] N. Levy. *Waze launches carpool service at 50 Amazon warehouses as part of nationwide rollout*. 2018. URL: https://www.geekwire.com/2018/waze-launches-carpool-service-50-amazon-warehouses-part-nationwide-rollout/. Accessed: 06.04.2019.

[26] F. Lindenberg, G. Aisch, and S. Wehrmeyer. *Quickstart*. 2018. URL: https://dataset.readthedocs.io/en/latest/quickstart.html#storing-data. Accessed: 08.10.2018.

[27] A. Ruhland. *The Lyft Formula: 5 Steps to Getting Noticed in a Dominated Market*. 2014. URL: https://www.leadpages.net/blog/the-lyft-formula/. Accessed: 11.04.2019.

[28] P. Salanne. *VINCI Autoroutes - Rapport d'activité VINCI Autoroutes 2010*. 2010. URL: https://publi.vinci.com/vinci_autoroutes/vinci-autoroutes-rapport-activite-2010.pdf. Accessed: 15.03.2019.

[29] A. Schwartz. *IKEA's Leko Is a Carpooling Service, Not a Flatpack Car*. 2009. URL: https://www.fastcompany.com/1249386/ikeas-leko-carpooling-service-not-flatpack-car. Accessed: 15.03.2019.

[30] T. Shay. *Most popular databases in 2018 according to StackOverflow survey*. 2018. URL: https://www.eversql.com/most-popular-databases-in-2018-according-to-stackoverflow-survey/. Accessed: 09.04.2019.

[31] B. Sutton. *Scaling Up Excellence*. 2014. URL: https://ecorner.stanford.edu/video/scaling-up-excellence-entire-talk/. Accessed: 15.04.2019.

[32] A. Tabatabai, K. Korosec, and K. Clark. *Dissecting what Lyft's IPO means for Uber and the future of mobility*. 2019. URL: https://techcrunch.com/2019/04/06/dissecting-lyfts-ipo-uber-scooters-autonomous-vehicles-and-the-future-of-mobility/. Accessed: 06.04.2019.