Nearme: mobile application to help people to meet up with friends and acquaintances

Judit Ramo Solé

Director: Eloi Puertas Prats
Realitzat a: Departament de Matemàtiques i Informàtica
Barcelona, 24 de gener de 2022
# Table of Contents

Abstract ......................................................................................................................... 3
Resum ............................................................................................................................ 3
Resumen ....................................................................................................................... 3
Introduction ............................................................................................................... 4
  Motivation ............................................................................................................... 5
  Objectives .............................................................................................................. 5
  Structure of document ......................................................................................... 7
Analysis ...................................................................................................................... 8
  Context .................................................................................................................... 8
  The Idea .................................................................................................................. 8
  Functionalities ....................................................................................................... 10
  Branding .................................................................................................................. 14
Planning ..................................................................................................................... 18
  Product Backlog ..................................................................................................... 19
  Sprints .................................................................................................................... 19
  Final Stage ............................................................................................................. 20
Architecture Design .................................................................................................. 22
  Frontend .................................................................................................................. 22
  Backend ................................................................................................................ 23
UX Design .................................................................................................................. 27
Final Design ............................................................................................................... 28
  Phone Sign In ........................................................................................................ 29
  Sign Up .................................................................................................................. 30
  Login ....................................................................................................................... 31
  User Details .......................................................................................................... 32
  Contacts ............................................................................................................... 34
  Find Friends and Friendship Requests ............................................................... 36
  Location and Match System ............................................................................. 37
Implementation .................................................................40
Backend Architecture ..........................................................40
Security, tokens, encryption .................................................44
WebSockets .........................................................................44
Testing the Results ..............................................................45
Tasks to test the product .......................................................45
Detailed Results ....................................................................46
Conclusions of the test .........................................................47
Conclusions ..........................................................................48
Future Work ........................................................................49
References ............................................................................50
Annexes ...............................................................................52
Annex I: Manual ..................................................................52
Annex II: Sprint Details .........................................................53
Abstract
In an ever-evolving society, sometimes it is hard to not fall behind. We have implemented in our lives all kinds of new technologies. We live surrounded by screens and social media and sometimes we forget that we also live in a physical world. This project aims to use the very same technologies that most applications use to keep people looking at the screen with a different purpose. We intend to build a mobile app to bring people closer and encourage them to meet up face to face.

Resum
En una societat en constant evolució, de vegades és difícil no quedar-se enrere. Hem implantat a les nostres vides tota mena de noves tecnologies. Vivim envoltats de pantalles i xarxes socials i de vegades oblidem que també vivim en un món físic. Aquest projecte pretén utilitzar les mateixes tecnologies que utilitzen la majoria de les aplicacions per mantenir la gent mirant la pantalla amb un propòsit diferent. Tenim la intenció de crear una aplicació mòbil per apropar la gent i animar-la a trobar-se en persona.

Resumen
En una sociedad en constante evolución, a veces es difícil no quedarse atrás. Hemos implementado en nuestras vidas todo tipo de nuevas tecnologías. Vivimos rodeados de pantallas y redes sociales y a veces nos olvidamos que también vivimos en un mundo físico. Este proyecto tiene como objetivo utilizar las mismas tecnologías que utilizan la mayoría de las aplicaciones para mantener a las personas mirando la pantalla, pero con un propósito diferente. Tenemos la intención de construir una aplicación móvil para acercar a las personas y alentarlas a encontrarse cara a cara.
Introduction

If there is one thing that differentiates humans from the rest of the species on Earth, that is our continuous search for progress, innovation, and the use of new technologies we ourselves invent. That is what made us able to rule the world ever since history began. We have been involved in this cycle of artificial evolution for dozens of centuries and we have learnt and evolved with each and every technological innovation. As humanity grew, so did our societies.

As we learnt and kept learning, the time between discoveries decreased. Until the twentieth century, societies could keep up with the speed of innovation. When we started to use computers, general-purpose machines, only a small part of the population started using them. When we invented the internet, not too many people got into it, at the beginning at least. Although later, the general public did discover the greatness of world-scale connectivity.

Some years after the internet was created, we came up with online platforms where we could interact with each other and share different content. Nowadays we call them social media. Still, in the beginning, there was not a huge response to it. Although more and more people were using the internet every year.

Then something happened. There was a new platform that made it their mission to connect all the people in the world. When Facebook appeared, it outpaced any competitor very rapidly. Suddenly, everyone was on Facebook.

We were still learning how to integrate social media into our lives. How we could use it to get all the advantages from it, without giving up our entire lives. But then the smartphone came up. And we ended up having unlimited access to all these social media platforms just in the palm of our hands. Anytime. Everywhere.

While we are still trying to figure out how to let social media in our lives in a healthy way, the world keeps getting excited about the next new thing. Many components of social media make us not want to miss anything, so we spend more and more time on it every day.

As developers, we create software to improve people’s lives. We give them entertainment, tools to work with, to learn with, to play with, and even tools to become better in multiple ways. But we also create software that intrudes on their lives, that makes them addictive and even that generate depression.

In the end, all we can do as individuals is give what we can on our part and hope we did enough. With this project, we intend to encourage people to spend more time with each other without a screen in between.
Motivation

I have always looked at the world with different eyes to those around me. I often wonder why things are the way they are and not the way they should be. I heard once a conversation that went something like: “Nobody can change the world” and the other person responded: “Everybody changes the world!”. Big or small, I do believe that we can make this world a better place if we try to do good.

The reason I started this degree was that I wanted to improve the way we communicate with technology and the way we use it. I wanted to use this project as an opportunity to learn and create something that I believe people would use and like using.

Additionally, during the degree, I had the feeling that we never finished any project. The subjects I enjoyed the most were all those related to pure software, so it made sense to build a project like this and try to get as close as a real-production working-state project as possible.

Objectives

The idea of the project is to tell people when they have friends nearby and then let them interact with one another so that they can meet up.

The main objective of this project is to build an entire distributed mobile application for both Android and iOS platforms, build a secure server and deploy it online so that everyone who downloads the app can use it. The correct integration of the different technologies and the design decisions are key to the final result and the development of the project.

An application like this is never finished, and it can always be improved. New functionalities can be added and old ones can be updated. That is why one of the important goals of the project is to reach a working state for the app and its main functionalities. Then iterate on it adding new features, but having always a working application at the end of each iteration.

We will be combining the knowledge acquired in different subjects of the degree, such as:

- Disseny de Software
- Factors Humans
- Enginyeria del Software
- Software Distribuít
- Projecte Integrat de Software
- Computació Orientada al Web
- Bases de Dades
To be able to evaluate the work done at the end of the project, we define a set of more specific goals. We will divide them into two categories:

**App Objectives:**

1. Build a mobile application that satisfies the requirements of the project. Those are to be able to manage your account, manage contacts, get notifications and be able to start an interaction with the other user.

2. Build a non-intrusive app: it works in the background, and the user only needs to open it when there is a match or to add new contacts.

3. Doesn’t take a long time to use or learn to use and has an intuitive interface.

4. Build a brand around the app, so that we can customise the look and feel of the interface according to it.

5. We treat delicate information very carefully. Passwords must be encrypted all the time, especially when stored, and session tokens must provide a secure authentication mode. As this application uses phone numbers to manage contacts, we need to verify the user is the owner of the phone number on their account.

**Project Objectives:**

1. Build a mobile application for both iOS and Android platforms. We will have to study the different options to do so and make the pertinent decisions.

2. Implement the application in a develop-production environment. Set up the project on GitHub with automated test and deployment to achieve a correct Continuous Integration and Continuous Deployment (CI/CD).

3. Perform a usability test to make sure we are building an intuitive interface and the distribution of the element on the app makes sense. Then take our conclusions on what the app needs to improve.

4. Build an efficient application. During the development, we will face different obstacles. We must work to keep the project light and efficient.

5. Build a secure backend system. Make sure the information is safely stored and that access to the server is restricted to authorised users.

6. Design a good model for the data in our system and investigate what technologies we should use for the database and the different requirements.
Structure of document

This is a cross-disciplinary project. Thus there is a variety of issues regarding different parts of the project that we encountered and focused on. We will try to summarise all the work done during the semester. The structure of this document will be:

1. Analysis of the problem we are trying to solve and description of the solution.
2. Functionalities we want to implement on the app.
4. Planning of the work to do during the semester.
5. Architecture Design and description of the technologies used and the decisions made along the project.
6. Interface Design, including the initial prototype and the final implemented design.
7. Implementation of the most interesting parts of the program.
8. Results and UX testing of the final version of the project.
10. Future work.
Analysis

Context

There are multiple platforms online to schedule meetings with people that could be of our interest. Whether we look to meet new potential lovers, people to debate with, new friends or other professionals in our field, we have our option. However, the vast majority of these apps are meant only for meeting new people.

If we search for platforms where we can interact with people who are already our friends, we will probably end up on social media. There we can chat and share information with people we know, and people we don’t know. The issue with social media is that is completely online, and in a world that is becoming ever more digital, we tend to spend more time on our cellphones than we probably should.

Far from wanting to create yet another social media or another platform to meet new people, we wanted to build a tool to help us meet with people we already know without too much work for our part. We also wanted to encourage people to meet each other in real life and not just through a screen.

The Idea

The initial idea is to tell people when they have friends nearby so that they can casually meet or get together to do some activity. We identified some cases that happen regularly and we want to fix:

1. Two friends are near each other and they don’t know.

Imagine two friends talking about what they did last weekend. One says: “I was going to stay at home, but a colleague told me about this festival near the city and I went on Saturday night”. And the other answers: “Really? I was there on Saturday! What a shame, I wanted to stay but my friends didn’t, I could’ve stayed with you…”

In this particular situation, our app would have told them they were very close to each other and they could have met if they wanted to.

2. Two old friends live or work very closely and they don’t know.

- “Oh, hey! Is it you? How are you doing? What are you doing here?”
- “Hey, I didn't recognise you! I live right there, I’m going to work.”
- “You live here? Since when? I live about two blocks from here.”
- “Well, it’s been more than 4 years, actually…”
- “No way… I’ve been living here for 2 years… How come we haven’t met before?”

Instead of eventually meeting, these people could have known they live near each other if they were contacts on our app.
3. Two people who know each other are in a foreign country, far from home, but relatively close to each other.

They say there is no place like home. When we go abroad it can feel a bit overwhelming, especially if we go alone. If we could meet someone we know, even if they are not our friends, wouldn’t we want to meet with them? Well, in case we didn’t know, the app would tell us if there is someone near. In foreign countries, it would make sense to automatically increase all ranges for our contacts or work with special parameters.

How it would work:

- You probably already have a list of people you know or have known in your life in your pocket. The idea would be to use this phone book on your smartphone to search for contacts. Users should then sign in with their phones.

- Once you have added your contacts, the app can simply just start working. It notifies you when you have one of those contacts near you.

- You could manage when to get notified, contact by contact or group by group.

- You can create groups to manage some contacts all at once. That would be useful to manage a group of coworkers or neighbours.

- You could set a region or area to not get notified. This area could be assigned to a contact or a group, so that for instance, when you are at home you would be notified if a neighbour is at their home as well.

- The user could turn off and on all their notifications (‘busy’ mode). If they don’t get notified, neither do their contacts.

Things to consider:

- Contacts on the app should have permission to be contacts from both users.

- You cannot have the location of a contact on your phone without their explicit permission. (This implies all the calculations regarding locations must be on the server.

- When the app is running, in order to work properly, it must track the geolocation periodically. This might result in a higher consumption of power.
Functionalities

The main functionality of the app is to send a notification when a user is close to a friend. For that, we will have to build a whole system able to manage users, make them friends, manage their configurations and ultimately send notifications and allow interaction after matches.

In order to develop the application, we will focus on what the user will do with the interface, how the interface will manage that and how it will respond. To do so, we use a Use Case Diagram, shown in figure 1, that later will result in a list of User Stories.

Use Cases

As said above, we need first to manage users. We will have to build an authentication system to give the users a private space where they can do their interactions with the app and other users. Regarding the diagram (fig 1), this would involve the cases mark in blue.

![Use Case Diagram](image)

Fig 1. Use Case Diagram
Then, we allow users to add friends as contacts. For this, we will have to build a request system to let users control whom they want to have as friends. To give them more control, we can also let them manage when they want to receive a notification: They could establish safe zones, where they receive no notifications, make these safe zones only safe from a group of users, or simply change how close must a contact be to get a notification.

As we can see in diagram 1, we would allow users to customise their preferences for any contact. In order to manage the preferences for more than one user at once, we would let them manage contacts in lists. This would be extremely helpful when having a group of contacts that are, for instance, coworkers or neighbours, for the user wouldn’t have to change their configuration one by one.

Last but most important, when we have built a system that connects users, we will be able to compare their locations. Based on the preferences they have for each other, the app will determine whether to send a notification or not.

After both users involved in a match are notified, we will need a system for them to get in touch, have a conversation and even share their locations so that they can meet. In addition, we have to think about a system to manage rejections when a user doesn’t want to meet or when they just don’t see the notifications.

**Epics**

We can separate all the identified use cases into different User Stories so that we can implement the basic functionalities one by one. This will help us later follow an agile methodology and reach a working stage of the app at every iteration of the code.

First, we will take these use cases and turn them into Epics (figure 2). An epic is a piece of work that can be broken down into specific tasks (user stories) based on the needs and requests of the end-users.

<table>
<thead>
<tr>
<th>Epic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>A user needs to access their account and manage its information</td>
</tr>
<tr>
<td>Contacts</td>
<td>A user can add and manage contacts</td>
</tr>
<tr>
<td>Lists</td>
<td>A user can add and manage lists of contacts</td>
</tr>
<tr>
<td>Location</td>
<td>The app tracks the location of users and send tells them when contacts are nearby</td>
</tr>
</tbody>
</table>

*Figure 2. Epics*
User Stories

Now we are going to extract the particular tasks we need to implement. We will do so for every epic that we have. As user stories are short descriptions of a feature told from the perspective of a user or customer, our stories will follow the commonly used template by Marc Cohn:

\[ \text{As a } \text{< type of user >}, \text{ I want } \text{< some goal > so that } \text{< some reason >} \]

Epic User:

<table>
<thead>
<tr>
<th>US</th>
<th>User Story</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>As an unregistered user I want to register to the app so that I can start using it</td>
</tr>
<tr>
<td>23</td>
<td>As a user I want to login in my phone so that I can access to my account and use the app</td>
</tr>
<tr>
<td>24</td>
<td>As a user I want to see my personal information and be able to change it so that I can control what information the app has</td>
</tr>
<tr>
<td>26</td>
<td>As a user I want to logout of the app so that I can make sure the session is closed.</td>
</tr>
<tr>
<td>27</td>
<td>As a user I want to change my profile picture so that people can recognise me</td>
</tr>
</tbody>
</table>

Figure 3. User Stories of Epic User

Epic Contacts:

<table>
<thead>
<tr>
<th>US</th>
<th>User Story</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>As a user I want to see my contacts so that I can change their configurations</td>
</tr>
<tr>
<td>2</td>
<td>As a user I want to add a contact to my account so that I can be notified when he/she is nearby</td>
</tr>
<tr>
<td>12</td>
<td>As a user I want to block a contact so that I can stop all interaction with them</td>
</tr>
<tr>
<td>14</td>
<td>As a user I want to see the configuration for a given user so that I can know when I'll get notified</td>
</tr>
<tr>
<td>25</td>
<td>As a user I want to deny a contact request from another user so that I can control who can have me as contact.</td>
</tr>
<tr>
<td>28</td>
<td>As a user I want to add a friend that is not on my contact list so that I can connect with people I meet</td>
</tr>
</tbody>
</table>

Figure 4. User Stories of Epic Contacts

---

1 Mike Cohn is one of the founders of the Scrum Alliance and one of the contributors to the Scrum software development method
### Epic Lists:

<table>
<thead>
<tr>
<th>US</th>
<th>User Story</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>As a user I want to add a contact to a list so that I can manage my contacts better (by groups, all at once)</td>
</tr>
<tr>
<td>4</td>
<td>As a user I want to create a new group of contacts so that I can add contacts to the list</td>
</tr>
<tr>
<td>10</td>
<td>As a user I want to change the preferences for a given group so that I can decide when to get notified</td>
</tr>
<tr>
<td>11</td>
<td>As a user I want to change the preferences for a given contact so that I can decide when to get notified</td>
</tr>
<tr>
<td>15</td>
<td>As a user I want to see the configuration for a given group so that I can know when I'll get notified</td>
</tr>
<tr>
<td>16</td>
<td>As a user I want to see a list of my groups so that I can see/change the configurations for it</td>
</tr>
<tr>
<td>17</td>
<td>As a user I want to see a list of contacts for a given group so that I can manage which contacts are in...</td>
</tr>
</tbody>
</table>

Figure 5. User Stories of Epic Lists

### Epic Location:

<table>
<thead>
<tr>
<th>US</th>
<th>User Story</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>As a user I want to connect with a contact when they are nearby so that I can meet with them</td>
</tr>
<tr>
<td>6</td>
<td>As a user I want to answer to a connexion so that I can accept it</td>
</tr>
<tr>
<td>7</td>
<td>As a user I want to reject a connection so that I can keep doing what I’m doing</td>
</tr>
<tr>
<td>8</td>
<td>As a user I want to send a message when I reject a connexion so that the other person doesn’t get angry with me</td>
</tr>
<tr>
<td>9</td>
<td>As a user I want to have some predefined answers to send when I accept/reject a connection so that I don’t have to think about what to answer</td>
</tr>
<tr>
<td>18</td>
<td>As a user I want to share my location with a given contact so that I can show them where I am</td>
</tr>
<tr>
<td>19</td>
<td>As a user I want to see a contact’s location (shared with me) so that I can know where they are</td>
</tr>
<tr>
<td>20</td>
<td>As a user I want to have a chat with all the contacts that I want so that I can keep a conversation with them within the app</td>
</tr>
<tr>
<td>21</td>
<td>As a user I want to view my latest “matches” with any contact so that I can see what the app has done for me</td>
</tr>
</tbody>
</table>

Figure 6. User Stories of Epic Location
Branding

While the app has the mission to bring people together, and it intends not to be intrusive, we still want to build a brand that will differentiate us from other applications. We pretend to inspire trust and encourage people to meet and have fun so we will present our brand as energetic, modern and vibrant.

Name

We wanted the name of the app to be easy to pronounce in different languages. We also wanted to describe what the app is about and create a strong and easy-to-get bond between the name and the mission of the app. As that is to encourage people who are near each other to meet, we will try to inspire nearness and proximity.

There were some candidates. First, we thought about Nearby, but then we discovered that Android has a relatively new function called that very same way. In addition, there was an app called Nearby on the Play Store. So we dismissed that option.

The next option considered was Nearme. Unlike the previous option, this is a composed name instead of just a word. That makes it harder to find other projects using it and it can give the brand the sense we were looking for. If we look for Nearme in the Play Store we can find a lot of applications sharing the name. Most of them are from outside of Europe and act locally in other countries, so it should not be a problem for the users to find our app easily. Besides, we pretend to rely on people recommending the app to their friends.

In conclusion, the app will be called Nearme.

Colours

As we said before, we are looking for the app to be modern and look energetic. We won’t be using a lot of colour in the app interface but will difference the important buttons and links with it. We will also use the brand colours for the app icon and the logo.

We decided to use a colour range of yellows and oranges (figure 7). This also sets us apart from the commonly used blue colour inside the app and it will add more consistency between elements.

![Figure 7. Colour Range for Nearme](image)
**App colours:**

Yellow is a dangerous colour we have to be careful with, especially on top of a white background. If we are adding buttons, they are most likely going to be visible for a great majority of users. But if we use this colour for text such as links, we will need to darken it.

In order to maintain the same look and feel throughout the app, we predefine the colours we will use. We will need to use them in different situations and elements, so we are choosing colours that combine with each other, and have different effects on the background. We can see all the colours chosen in figure 8. We will use three colours ordered by an order of preference.

Since we will be applying these colours on clickable elements, we define a shade and a tint tone for each colour. This way when an element is clicked it will be displayed in a darker colour. We will be using HTML elements. They have more effects than just ‘click’, but we can cover them all with these variations.

For these colours, with orange tones, we decided that the possible text on top of them would be black. We don’t use white because of the Primary colour, which is lighter, and it could decrease the visibility of the text. As for the other colours, we rather maintain the same colour for the sake of cohesion and coherence.

Regarding the colour of the text, we most likely will use the secondary colour to increase visibility and improve the experience.
Preparing for the app, we will define two more colours. One for success and one for ‘danger’ actions (figure 9). They normally mean ‘accept’ and ‘cancel’ / ‘delete’ respectively. We will also use them to indicate whether a user is connected to the app or not.

As we can see the colour for ‘success’ actions is green, with its tones of shade and tint. Same for the danger colour, being a tone of red in this case. We do not intend to use these colours for big elements or very often, but we predict they will be used indeed.

It is common that together with these two colours we find the warning colour. This would have an amber tone. However, it is too similar to the colours we are using as app colours, so there is no need to add it. Besides, we do not expect to need it in our app.

**Typography**

Another important element to build a brand is the typography we use for all the content. We said above that we wanted our brand to look modern. Since we are not creating it ourselves, we would need to find a typography font that we can use. We could look for a commonly used but preferred to differentiate ourselves in that as well. For all that we searched for a sans serif typography with a free license that wasn’t used too often.

We found some options online and we finally decided to use one called PT Root UI (see figure 10). It has a free license for both personal and commercial use². We downloaded the font family which contains the typography in Regular, Light, Medium and Bold versions.

We will be using different versions in the font family. We will apply PT Root UI Regular to most texts. When we need to show bigger letters or emphasise some words we will use PT Root UI Medium or Bold as needed.

---

² PT Root UI is licensed under the [SIL Open Font License (OFL)](https://scripts.sil.org/OFL)
Logo

We need a logo to use in the app launcher, the different stores, and for people to identify the app more easily. We will use the colours mentioned before. We want to give the app a modern and light look so we will use a degradation between orange and yellow (figure 12).

We wanted to make the icon suggest the app had something to do with location, so we tried out different shapes. In the end, we chose a simple icon with two white circumferences on the aforementioned background. We also did a version that inverts the colours, so that we could use it with a white background (figure 11).
Planning

The agreement for the development of this project was made on September 14th. Since the deadline is on January 24th, this leaves us with 19 weeks to work on the project. We wanted to work on the code for as long as possible, so we planned first an initial couple of weeks to research the best way to approach the project, and a month, in the end, to finish it for delivery. The latter consisted of a UX Testing session to test the product and the production of all the necessary documentation. The rest of the time in between was dedicated to the development of the project itself. We can see the timing for each part in figure 13.

<table>
<thead>
<tr>
<th>Week</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UX testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 13. Initial Planning

Regarding the planning of the code development, this project has been developed following the Scrum model. This is an agile methodology designed for teams of ten or fewer members that work on a project through a series of time-boxed iterations called sprints. There are two important figures in the Scrum methodology: the Product Owner, who advocates for the interests of the business and users, and the ScrumMaster, who helps the team perform at their highest level.

At the beginning of the project the development team, including the ScrumMaster, and the Product Owner agree to a list of features the product needs to have. This list is called the Product Backlog. At each sprint, the development team will commit to completing a particular set of tasks. To know the importance of every feature every step of the way, each item on the Product Backlog is assigned a business value. The team also estimate the work weight for the items.

During the sprint, the members of the team gather every day in a meeting no longer than 15 minutes (called daily scrum) to discuss what has been done and what will be done the next day. At the end of the sprint, the team demonstrates the new functionalities to the Product Owner. Then conducts a sprint retrospective to reflect on the work done and identify opportunities to improve.

As Scrum is a framework to manage teamwork and this particular project has been developed by just one member, some of the scrum's characteristics weren’t implemented. Both the Product Owner and the ScrumMaster roles fall on the developer, and the sprint reviews are built as a meeting with the tutor of the project.
Product Backlog

The most important part of the initial planning is the creation of the Product Backlog. In each sprint, we will be creating a new Sprint Backlog based on tasks on the Product Backlog. Therefore, it must be highly detailed and contain specific acceptance criteria to make sure we end the sprint with a working version of the code. A task is done, if the code works and does everything it was agreed it should do. It should also be fully tested and documented.

The functionalities that we need to include in our Backlog are strictly related to the use cases and user stories studied above. All user stories will be converted into tasks, with their specific acceptance criteria. We can check the extended Backlog in appendix X. After completing the list we must assign a business value and Scrum points to each element on the list.

Sprints

We will divide the development work into a series of sprints that will last from 15 to 20 days each. In annex II we can find the full report for every sprint. At the beginning of the sprint, we will decide which US will be developed. Then we will evaluate each one and estimate the approximate number of hours that it will take to get it done. With that and the tracking of our work day by day, we can construct what is called a Burndown Chart (see figure 14). This shows the remaining effort for each day. It compares the ideal burndown with the actual work done. Updating the information every day, we can know if we need to speed up or if we are doing the work we expected.

---

**Accumulated Burndown Chart of all Sprints**

![Accumulated Burndown Chart of all Sprints](image)

Figure 14. Sprints Burn Down Chart Accumulated
In Figure 14 we can see the evolution of each sprint. To sum up, the work done on each sprint and understand how the project evolved through time we can go to Figure 15.

The first sprints were dedicated to the creation of the app, the design of the interface, and the authentication system. Later on, the system to manage contacts was done. Once we had those systems working, we could start with the location functionalities. This includes the location tracking on the background for the frontend and the matching system, that needs to communicate with the front app asynchronously.

In this last figure, we can also see when we needed to do some research about the new technologies to be implemented and when we needed to design more screens as we added functionalities.

### Final Stage

Some work areas took longer to implement than we first expected. This is due to the need to add these new technologies to the project and the complexity of some implementations. In Figure 16 we have the summarised stage of the Backlog at the end of sprint 6.

We can see that there are three issues for which we had to spend a lot more hours than for the rest. The setting up of the project included learning the new technologies, preparing the security layer and setting the Continuous Deployment of the backend. US number 22 involves all the design and implementation of all the screens on the sign-in flux that we will see later. Finally, the US05 includes geolocation background tracking and web sockets. Both had a long learning curve.
We can also see some issues marked as “Working on”. The app is in a working state, but these user stories didn’t pass all the acceptance criteria. Basically, that is because the communication and chat system is currently implemented via WhatsApp.

<table>
<thead>
<tr>
<th>US</th>
<th>Short Description</th>
<th>BP</th>
<th>SP</th>
<th>Hours worked</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>US000</td>
<td>Set up</td>
<td>-</td>
<td>20</td>
<td>62h</td>
<td>-</td>
</tr>
<tr>
<td>US001</td>
<td>See Contacts</td>
<td>20</td>
<td>5</td>
<td>6h</td>
<td>Done</td>
</tr>
<tr>
<td>US002</td>
<td>Add Contact (send request)</td>
<td>20</td>
<td>10</td>
<td>12h</td>
<td>Done</td>
</tr>
<tr>
<td>US003</td>
<td>Add Contact to List</td>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US004</td>
<td>Add List</td>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US005</td>
<td>Be notified when near a contact</td>
<td>20</td>
<td>20</td>
<td>99h</td>
<td>Done</td>
</tr>
<tr>
<td>US006</td>
<td>Answer connection</td>
<td>20</td>
<td>20</td>
<td>21h</td>
<td>Working on</td>
</tr>
<tr>
<td>US007</td>
<td>Reject connection</td>
<td>20</td>
<td>10</td>
<td>19h</td>
<td>Working on</td>
</tr>
<tr>
<td>US008</td>
<td>Send message when rejection</td>
<td>2</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US009</td>
<td>Predefined answers</td>
<td>1</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US010</td>
<td>Change group preferences</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US011</td>
<td>Change contact preferences</td>
<td>10</td>
<td>20</td>
<td>3,5h</td>
<td>Done</td>
</tr>
<tr>
<td>US012</td>
<td>Block a contact</td>
<td>1</td>
<td>10</td>
<td>4,5h</td>
<td>Done</td>
</tr>
<tr>
<td>US013</td>
<td>Search near me now</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US014</td>
<td>See contact details</td>
<td>10</td>
<td>10</td>
<td>4h</td>
<td>Done</td>
</tr>
<tr>
<td>US015</td>
<td>See List details</td>
<td>10</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US016</td>
<td>See Lists</td>
<td>10</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US017</td>
<td>See List's Contacts</td>
<td>10</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US018</td>
<td>Share Location</td>
<td>2</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US019</td>
<td>See Contact's Location (shared)</td>
<td>2</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US020</td>
<td>Chat</td>
<td>5</td>
<td>20</td>
<td>6h</td>
<td>Working on</td>
</tr>
<tr>
<td>US021</td>
<td>History with Contact</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US022</td>
<td>Register</td>
<td>20</td>
<td>20</td>
<td>71h</td>
<td>Done</td>
</tr>
<tr>
<td>US023</td>
<td>Login</td>
<td>20</td>
<td>20</td>
<td>16h</td>
<td>Done</td>
</tr>
<tr>
<td>US024</td>
<td>See and change my user details</td>
<td>20</td>
<td>20</td>
<td>15h</td>
<td>Done</td>
</tr>
<tr>
<td>US025</td>
<td>Accept or Deny Contact Request</td>
<td>20</td>
<td>10</td>
<td>10h</td>
<td>Done</td>
</tr>
<tr>
<td>US026</td>
<td>Logout</td>
<td>20</td>
<td>5</td>
<td>2h</td>
<td>Done</td>
</tr>
<tr>
<td>US027</td>
<td>Change Profile Picture</td>
<td>2</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US028</td>
<td>Add Contact without phone</td>
<td>2</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 16. Backlog at Final Stage of the Project
Architecture Design

In the early stage of the project, various decisions had to be made. The major ones were about the technologies used in the development and the architecture of both the frontend and the backend. (To follow better the explanation see figure 18).

Frontend

One of the objectives of the project was to build an application that worked both on iOS and android platforms. There are two options to achieve that. On one hand, we can build two different applications, one for each platform. On the other, we can build what is called a hybrid application. Both ways have their pros and cons that we have summarised in figure 17.

<table>
<thead>
<tr>
<th>iOS app + Android app</th>
<th>Hybrid Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Having to write the same functionalities twice.</td>
<td>Only having to write the code once</td>
</tr>
<tr>
<td>Working directly with the operation system, easily accessing the built-in capabilities of the user’s device (GPS, address book, etc.).</td>
<td>Needing plugins to interact with the different systems, and depending on how those plugins work and if they are updated.</td>
</tr>
<tr>
<td>Using functionalities of both Xcode and Swift and Android Studio and Kotlin</td>
<td>Using web technologies that are highly documented</td>
</tr>
<tr>
<td>Able to debug the app directly</td>
<td>Debugging only available for web (and some tasks cannot be performed in the browser)</td>
</tr>
<tr>
<td>Different systems to build the GUI</td>
<td>Relies on web design frameworks</td>
</tr>
<tr>
<td>It allows to have two separate teams that work on the two different frontend platforms</td>
<td>If there is only one team developing the frontend, it is easier and keeps things simple.</td>
</tr>
</tbody>
</table>

Figure 17. Characteristics of Hybrid and Native Mobile Applications

Building two different native applications is more than a valid option, especially if it is meant to be put on production and maintained for a long time. However, it will take a lot more time to set up the projects and develop every function for, not only two projects, but for two different languages.

Hybrid applications can be equally laborious to set up but much faster to develop, as you don’t have to do it twice. Although you may end up depending on unofficial software you do not control, you can always build your own libraries and plugins.

In conclusion, both systems are valid, and there is still an open discussion on the internet about it. On this project thought, we have chosen to build a hybrid app. This will simplify the work and will keep all the frontend code in just one place. This way, we also avoid the inconsistencies between versions.
Framework

In order to develop the GUI for a hybrid application, we can use different frameworks. The most commonly used is Ionic, which offers sufficient native functionalities and several UI components. There are other alternatives such as React Native, made by Facebook, that claims to be very easy to develop with, but it is said to have a high learning curve.

Since Ionic is completely based on pure HTML, CSS and javascript, it makes a most compelling case. In addition, it can be used with Angular, which uses typescript instead of javascript. There is also more extensive documentation about ionic than about any other similar framework, so we chose to use it for this project.

Although Ionic was built to work with Angularjs and the release of Ionic 4 added the possibility to use ionic with React and Vue.js, the technology best integrated with it since the second version of the framework is Angular 2. Because of that and the fact that it uses typescript instead of javascript, added to the high volume of documentation for Angular, we have chosen it to develop this project.

Ionic 4 also gave birth to Capacitor, a new alternative for Apache Cordova. Both serve as a converter from the coded web app to the native mobile applications. They also provide libraries and plugins to connect with native functionalities. In this project, we will use Capacitor when we can, as it is recommended by Ionic, and Cordova or other open-source projects for functionalities not implemented in Capacitor yet.

To sum up, and as we can see in figure 18, the frontend is implemented in Ionic 4 for the graphic components, in Angular for the logic of the app, and it uses Capacitor to build native projects on both iOS and Android platforms. These last projects consist of native applications that contain the web app inside. To build and run them we will use Xcode and Android Studio.

Backend

Our backend system will serve as the permanency of data and interactions with other users for the app. It will also contain the main logic of the program and will decide what actions can be performed. To communicate both frontend app and backend server we will build a REST API. All Client apps will consume information from the Server through API calls.

When there is a match between two users, the server needs to communicate with these two clients, but it cannot do so through the API system, which is unidirectional. We will need a way to establish a bidirectional connection or at least a way for the backend to send information to a specific client on demand.

The ideal way to do this would be to take advantage of the push notifications system both iOS and Android have. However, we encounter a problem with Apple. We need to have an Apple Developer Program Plan to be able to implement push notifications on iOS. After some research, we decided to implement the notification system using web sockets and local notifications.
Framework

To implement the backend server, two options were considered: Flask, and Java Spring. Flask is a lightweight and simple framework that works with python. Spring is a framework that uses java, and it can be built as an API or for both front and backend.

Both options could be a good solution but since most of the logic of the application will be on the backend, Java was considered a better option. Especially for its great approach on Object-Oriented Programming and its extended usage for big projects that will need thousands of lines of code.

Inside the Spring world, we can find different projects that we can add to our code. Following an Agile methodology, we incorporated the diverse spring technologies as we needed them to implement the tasks assigned in the specific sprint. The most important that we used are:

- **Spring Boot**: Used to build and start the project. It simplifies the entire development of the project and automatically sets most of the configuration the Spring framework needs. We can always overwrite the configuration we need to change.

- **Spring Security**: As they put it on their website: “Spring Security is a powerful and highly customizable authentication and access-control framework”. It is indeed customisable through all its configurations, although sometimes these configurations can get a bit tedious. We used it in our server to create the whole security layer and create a filter for all those endpoints that needed to be secured.

- **Spring Data JPA**: to create the database dynamically. It sets all the characteristics of each table and attributes through a system of annotations. The query system to modify the database is implemented with Repository JPA classes, making it much easier to work with the database. It accepts multiple relational database systems.

Java Spring is prepared to run as a server, so we will have the backend system always running. It will be connected to the frontend applications via HTTPS protocol and Web Sockets. The functionality for HTTPS is already implemented in Spring so we will just have to use it. For the addition of Web Sockets, we will use a library called SockJS.

Deployment

This is a distributed application. We have lots of clients that will connect and interact with the same server. We will need to maintain the server always online to give service to the users. So we needed to find a hosting service that could build, run and operate our server.

With all the new technologies implemented on this project we wanted to use a platform that we already knew, that could manage a relational database and that could be easily connected with GitHub to implement a Continuous Deployment. To do all that, we decided to use Heroku, which supports several programming languages, including java.
Additionally, Heroku has multiple services that you can install to your project as add-ons. We decided to use Heroku Postgres to build the database. We had to configure the project to be able to run on Heroku, and consume the database connected to it, so we achieved that with java Spring’s configurations.

We considered using other types of databases, but the priority was to implement the different functionalities. If by the right time we thought we needed to use a nonSQL database to make the app more efficient or work better, we could always add that option.

GitHub

One thing we mentioned earlier was Continuous Deployment (CD). Working with GitHub, we discovered the newly added functionality of GitHub Actions. This allows us easily automate all our software workflows. We used it to maintain Continuous Integration along with CD, to make sure every integration of new code passed the implemented tests. When the server passes all tests, GitHub Actions directly deploys it to Heroku.

Figure 18. Frontend and Backend Technology Architecture.
Firebase

There were some functionalities we could not implement in Java Spring. We needed to use other services to be able to, for instance, verify phone numbers through SMS. We cannot send verification codes through SMS directly from Spring. We need another service that provides this functionality. There were various options we could choose from. To use most of them, we had to connect their services to the Java Spring project and manage all the communication from the server.

There was another more interesting option. We could implement this functionality through Google Firebase, a platform we have already used in the past multiple times. The interesting thing about Firebase is that it has a lot of functionalities you can add to your project with little effort. Besides, it was built to work for mobile applications, which is exactly what we are building.

In this case, we would not have to go through the server to verify a phone number. We would directly connect the mobile app to Firebase, verify the number and then send it to the server. To make sure the number passed the verification in the frontend, instead of sending the number itself, we can send the AuthUID returned by Firebase to the server. Then we can connect the backend to Firebase through Firebase Admin, and retrieve the phone number with the AuthUID.

The possibilities that Firebase bring were an added motivation to choose it. Going ahead with the project, we might want to implement other authentication systems, and with Firebase we have the possibility to add providers such as Google, Facebook, Twitter or Apple.

We could also implement a Push Notifications system through Firebase Cloud Messaging. This would be managed in the same Firebase project. This makes Firebase an excellent addition to the project.
UX Design

Apart from the architectural design of the application, there is a big important area of design that we had to work in: the user interface. In the end, that is the only thing that the user will actually see. To do an initial mockup of the app we used a program called Balsamiq (see figure 19).

Figure 19. App Initial Mockup made with Balsamiq
Final Design

In this section, we will explain the final state of the product. When the app starts, it shows a splash screen (figure 20), to allow time to load all the content. Normally, if it waits too long is because the server is sleeping, and it has to wake up (due to using Heroku’s free plan). If the user is already logged in the app will redirect them from this screen to the main application.

If the user is not logged in, the Wellcome Screen will appear (figure 21). It has only one action: to enter the sign-in flux.

Fig 20. Splash Screen  
Fig 21. Wellcome Screen
Phone Sign In

Contacts are currently added by phone, so we are interested in users signing up with their phone numbers. That's why the phone input screen (figure 22) is the sign-in method by default. Here the user needs to select their country and just write their phone number. We need to know the country to format the number correctly.

In case the user doesn’t want to sign up with their phone number they can directly go to the next page. If they have already an account and they want to log in via password, they can go to the login page as well.

Another consequence of using the phone number to find new contacts is that we have to make sure the user is the owner of this number. For that, we need to verify it by sending a code by SMS. The user then has to introduce the code on the app (figure 23).

After verifying the phone number, if the user is already registered, this will work as a login system and they will access the main application without further actions. If they are not registered, the signup form will appear instead.
**Sign Up**

In the signup form (figure 24), we will ask the user for a unique username, a unique email and a secure enough password. There were two options to implement the password input. We could either put two fields that had to match and be secured or put a single input field with the possibility to see what you are typing. We chose the latter because this way you can see what you are writing and make sure you wrote exactly what you intended to without having to rewrite the password again. This screen has also a link to go to the login page, for people who already have an account.

Before submitting the information, if the text written in the input fields is not valid, the app will show a message (see figure 25). This message in red will explain what the input needs to be acceptable, and it will disappear when it is. If the username or the email pass the format filters but are already used in the server, pressing the sign-in button will trigger a message saying so in the form of a toast. Finally, if everything is okay, and the app registers the new user, the app will show the main application.
Login

Users can log in to the application in two ways. One was to log in with a phone number, verifying they are the owners of the number. The other is to log in introducing the password (figure 26). To identify the account the user wants to login into, they can use the username, email or phone number. That is because all those elements are unique in the database, and we make sure no one signs up with an existing value.

The password field is shown as we saw on the signup page, with a single input that can hide the characters or show them as the user wants.

Messages of error from the server are shown as toasts (figure 27). This could be either that the username does not exist or that the password is wrong. As we can see, there is the option to let the app remember your credentials. It saves locally the id and the encrypted password on the device. To finish with the sign-in system, this page has a link to go back to the signup screen and when the user is logged in correctly it enters the main application.
**User Details**

The main application is distributed in tabs. The user tab shows the profile of the logged user and allows them to change their information (figure 28). From this page, they can also log out of the application. In this screen, we plan to put all the user configurations that can eventually be added.

If we click the change password configuration the app opens a new screen (figure 29). To change the password we only need to verify the old password and enter a secure new one.

![Fig. 28. User Detail Screen](image1)

![Fig. 29. Change Password Screen](image2)
If we click ‘Edit’ in the User Detail screen the app will allow us to edit our ‘name’ and ‘about’ (figure 30). This is information that our contacts will be able to see, and the user can choose what they want to put. When the user has finished editing can press ‘OK’ and the information will be saved.

The user can also choose what profile picture they want to use. In the future, the user may be able to upload their own picture, but for the moment we give them some options to choose from (figure 31). To change the profile avatar they only have to click on the picture, and they will be shown all the options.
Contacts

Once we have added contacts to our account, the tab of contacts will show the list of our contacts in the app (figure 32). Each item in the list represents a contact and then show their names, their personal information ('about') and their profile picture. We can also see at a glance which contacts are connected at the moment. Online contacts are shown first on the list, and with a green badge on the profile picture. Offline contacts have a red badge. We can search for a specific contact by typing on the search box. This will filter the list.

If we click at any contact on the list, we will go to the contact detail page (figure 33). We can observe this screen is similar to the User Profile page, but with changes to what the user can do. In this case, we see a badge showing whether the user is connected or not (green and red respectively). In addition, we can change our preferences for the contact.
We will be notified when the contact is closest to as than a certain distance. We can choose this distance by changing the notification range. If we click that option we will be prompted with a wheel of options (figure 35).

We can also block a contact if we want to stop receiving notifications from them. To do that, we click on the Block Contact option. It is displayed in red colour because it is considered an action the user would not do often and it can lead to unwanted behaviour when misclicked. To make sure the user actually wanted to do this action a confirmation alert is shown (figure 34).
Find Friends and Friendship Requests

In order for the app to be useful, the user will need to add some contacts. To do so we can go to the find contacts screen by clicking on the ‘Add’ link on the top of the Contacts page (figure 32). In this new screen (figure 36) all the contacts we have in our device phone book and that are on the app will appear. From here we can directly send a friendship request to the users we want. To be able to see our contacts on this screen, we must give the app permissions to read the contacts on the device phone book.

Once we have sent a friendship request, this will appear on the requests tab. If we click on it, the requests screen will appear (figure 37). From there we can see the pending requests separated as sent and received. The actions here are very intuitive. We can cancel a petition sent, and accept or reject a petition received.

In this screen, we use again the danger colour to indicate cancel or rejection and the success colour to indicate acceptance. Once we accept a request, the other will be added to the contact list.
Location and Match System

The user must give permission to the app to get the device location. If they want to receive notifications that someone is near without updating the location manually, they will have to give permission to the app to enable background geolocation tracking when prompted.

If the geolocation tracking is enabled, the user’s location will be sent to the server regularly. With this and the web socket established, the user will eventually receive a notification for a match. All active matches will be shown on the matches screen (figures 38 & 39). From here we will

At the bottom of this page, we can find a button that says ‘Search for friends near me’. And that is exactly what it does. It updates the location on the server and triggers the matching system.
The list of active matches will have different elements. The item itself will say what type it is. First, there are the new notifications, that are displayed the same way to the two users involved in a match.

When one of these users wants to contact the other, they can just click on the match and the action options in figure 40 will appear. If the user clicks ‘Contact’, a petition to meet will be sent. The match in question will show another message saying that a petition was sent. If we click on it again different actions are shown (figure 41). In this case, we can ‘Cancel’ the petition, which would reset the match to its initial state.

If we click ‘Edit Configuration’ we go to the Contact Detail page.
In case of the user receiving a petition to meet (figure 42), we would have the option to ‘Open Chat’, which currently opens the app WhatsApp in the conversation with our match contact. Or we could also just press ‘Interested’ and carry on the interaction however the user sees fit.

When a match is accepted for both users is closed and disappears from the match screen. We can find it in Recent Matches. If we click on the link at the top of the page that says ‘See Older’ we can see all the matches we have had, in which both users said they wanted to meet (figure 43).
Implementation

Backend Architecture

One of the challenges of this project was to learn how to use Java Spring and make the most of it. We will explain the most interesting parts of the code inside the Spring project, on ‘codi/backend/ServerAPI’.

Inversion of Control

Spring’s code slightly defers from regular Java projects. It uses Inversion of Control (IoC). That means that instead of our code calling a set of libraries, it is the Spring framework that will take control of the program flow and make calls to our custom code. The problem with this is that it can be difficult to get used to it, especially in the beginning. Even so, it simplifies the code and reduces the necessary effort to build the system.

Annotation System

The IoC is implemented through the IoC container, represented in Sprint as the interface called ApplicationContext. In our case, we will use WebApplicationContext. Its responsibility is to instantiate, assemble and manage objects called beans. Beans are defined with the annotation @Bean which can only be applied to methods. These methods are automatically called by the IoC container and will define a particular behaviour or configuration in our program.

The other big important annotation is @Component which is applied to types. There are different types of components with a different purposes each. The ones that we will use are:

- @Configuration: a class is decorated with this annotation to indicate that it contains @Bean definition methods. We use it to define certain behaviours from our configuration classes. In our project we have a file for general configuration that manages the connection to the database, initialises the connection to Firebase and creates some other useful beans. Apart from that, we have two more configuration classes, one to create the beans necessary to manage the security layer, and another to control the web sockets connections.

- @RestController: this is a specialised version of the controller (annotated with @Controller) that additionally implements functionalities from the @ResponseBody annotation. The classes decorated with this annotation will build the first layer of our server. They are typically used to handle server requests. To do that, we will use @RequestMapping to map the HTTP requests to the handler methods implemented that are additionally decorated. A method becomes a handler when annotated with one of the following: @GetMapping, @PostMapping, @PutMapping or @DeleteMapping. There are other types we could use, but these four are the ones we implemented in our program. We use these annotations to map the different endpoints to the methods. The functionalities added from @ResponseBody allow the request handling methods to automatically serialise return objects into HttpResponses.
- **@Service**: this particular annotation will only make the classes annotated with it available to the application context. We use this for the classes that contain the business logic.

- **@Repository**: annotates classes at the persistence layer, which will act as database repositories. In these classes, we will write all the queries we want to do over the database.

Apart from the basic classes we find in Spring Boot, we needed to implement other Spring services. One of those, which is actually very important, is Spring Data JPA. From all the options in the Spring Data Project, we decided to choose JPA for its simplicity to program with.

Spring Data JPA barely needs configuration. We used the annotation `@EnableJpaRepositories` on the Configuration class and we could start using JPA. We will use it to create the classes that will represent our tables on the database and to configure our repositories.

First, the annotation `@Entity` will mark a class to indicate we will use it as a table. Then we make our repositories decorated with `@Repository` to extend the JpaRepository interface. We will need to create a repository for every table/entity because JpaRepository needs our repository to specify which entity it will be using.

**Dependency Injection**

It is used when implementing IoC to set object’s dependencies. In our classes, initialized by the IoC container, we set the dependencies on the constructor. The WebApplicationContext then instantiate objects of all our annotated classes and pass those objects to the new classes that need them. We must be careful here, because we cannot have cyclic dependencies.

**Architecture**

Implementing all what we have talked about, we will end up with the following structure (see figure 44): At the lower level, we have the database, represented in our code as Entities. We have a set of Repositories, one for each Entity, that control access to the database. These Repositories have queries implemented that will be used from the Services, that contain most of the logic of our application. These Services sometimes call one another but are mostly used by the Controllers, which receive the HTTP requests through methods mapped to the different endpoints.

We can consult all these classes on the generated javadoc, placed on codi/backend/javadoc. In our project, we can see the different entities and repositories in the ‘restapi.crud’ folder inside the Spring project. Controllers and services have their own folders inside the ‘restapi’ package.
**API Rest Responses**

To control the exchange of information with the front app, we encapsulated the content of the responses in Data Transfer Objects (DTO). We can find these classes in the ‘json’ folder. We always return an `HttpResponse<RestResponse>`. This way we can control how the response is sent. `RestResponse` is an interface whose implementations will later be transformed into JSON text.

We have some interesting classes in this folder, implementing `RestResponse`. `RestList` and `RestDictionary` extend `ArrayList` and `HashMap` respectively. This allows us to send not only DTO objects but lists of DTOs or maps like `{string: DTO}`.

The DTOs serve as a filter for the Entity classes so that we can control which information we send and which we keep private in our server.

**Data Architecture**

We can see a summary of our database entities in figure 45. The first entity we designed was User. All the other entities use the user’s id to refer to which user belongs the information to. Another reference is `MatchRequest` having the `matchId` to indicate which `Match` is the information for.
One particular interesting implementation is the way we manage the user’s preferences for other contacts. Both users have their own preferences in the table `ContactPreferences`, but the preference we use to calculate if they are near each other are the preferences saved in the `Contact` row that relates them. These are the most restrictive preferences, that we update every time one of them makes any change on their configuration for each other.
Security, tokens, encryption

One of the greatest challenges of this project was preparing the security layer ready. As we can see in figure 44, it is applied to the petitions received from outside the server. All the security classes are inside the ‘restapi.utils.security’ package. The ‘config’ folder contains all the security settings.

In big terms, we are filtering all the server HTTP requests. We get the ‘Authorization’ Header, which will be a JWT Token, test if it is still valid and extract the username encoded in it. Then we compare this token to the encrypted token saved on the database, in the table User (searching with the username extracted). If everything is correct we let the petition through to the controllers. The controllers receive the original request with an AuthenticationPrincipal in the form of LoggedUser (we find this class in the security package). This makes it unnecessary for the client to send an id of the user, since we are already identifying it with the Token.

There is a set of endpoints that are not secured: all the sign-in methods (inside the class UsersControllerPublic). When a user logs in, through one of these endpoints, the server returns the JWT Token.

WebSockets

Another endpoint that is not secured is the one used to start a WebSocket connection. WebSockets are implemented with the Stomp library, using SockJS. This type of connection is meant to be bidirectional, but we will only use it to send information from the server to the client.

We use the security token to make sure the user is whom they say they are. However, we are not in the security layer, so we will only use it for authentication. We will send the username along with the token from the client.

We have already mentioned, that we were using WebSockets because we could not use Push Notifications. So, if in the future we change protocols and stop using Websockets, we will not have to change any other implementation.
Testing the Results

The app has been developed through the addition of various functionalities. In every sprint, there was a clear idea of what the front should look like at the end. However, the complexity of the application has grown and therefore we must make sure it is usable without a lot of explanation.

To measure the usability of the app and its ability to meet its objectives we will conduct a User Experience Testing session. During the test, we will expose any usability defect and discover any improvement users might suggest.

Tasks to test the product

All the functionalities of the app can be grouped into four blocks:

1. **Create and Edit User.** This includes the sign-in system and the customisation of the user’s profile.

2. **Add Contacts.** Here we have the request system to add new friends, including the search for contacts’ phones in the user's device.

3. **Edit Contact Information.** This is essential for the main functionality of the app. Users must be able to edit their configuration for each contact so that they can customise their experience, and allow the app to serve them as they need. It includes blocking and unblocking a specific contact and changing the minimum distance that contact must be from the user for the app to send a notification.

4. **Make Match and Meet with Friends.** Once received a notification from the app saying that a friend is near, we can start an interaction with them. For now, we can send a petition to meet and answer the petition to start a chat and share locations. This last part is in an early stage of development but the test will help decide which is the best direction to go.

In order to test these four blocks, we will ask the users to perform a particular set of tasks for each block. Considering the app needs interaction with other users, we will use another phone to send friendship requests, become friends, and let the application match us with the user.

<table>
<thead>
<tr>
<th>Block</th>
<th>Tasks</th>
</tr>
</thead>
</table>
| Create and Edit User   | 1. Open the app and create user with your phone, email, and a custom username.  
                            2. Without saving password.  
                            3. Customise personal profile: change name, personal info and profile picture.  
                            4. Log out from the app.  
                            5. Turn off app, turn on app and log in saving password. |
| Add Contacts           | 1. (User has received a friendship request) Reject request received.  
                            2. Send a friend request to any user. Then cancel the request.  
                            3. (User has received a friendship request) See and accept friend request. |
Detailed Results

After the tests with our three app testers, we arrived at some conclusions on things that needed fixing or improvement. This is a recompilation of what the users did on the test and their impressions while they were doing it:

<table>
<thead>
<tr>
<th>Block</th>
<th>Tasks</th>
</tr>
</thead>
</table>
| Edit Contacts | 1. Block a contact  
2. Change the distance within which you want the app to tell you when a particular friend is near.  
3. Unblock Contact previously blocked. |
| Make Match    | 1. (User has received a new match) Send a petition to meet to friend.  
2. Cancel the petition sent.  
3. (User has received a request to meet) Accept the petition received and open a conversation. |

<table>
<thead>
<tr>
<th>Block</th>
<th>User 1</th>
<th>User 2</th>
<th>User 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create User</td>
<td>🟢 Hesitated at the beginning but followed the steps very well.</td>
<td>🟢 Was confused while waiting for SMS message. Found log in confusing as well.</td>
<td>🔴 Unable to complete task. Sign in flux was not intuitive. User went to login instead of signing up. Send SMS button is confusing. When editing profile, clicked on Edit but then confused about what to do.</td>
</tr>
<tr>
<td>Add Contacts</td>
<td>🟢 Had to navigate through the tabs to find how to add a contact. Needed hint to find the Add button.</td>
<td>🟢 Add contact not intuitive. Search bar on Contacts was confusing. 🟡 Suggested number notification badge on tabs</td>
<td>🟢 Looked for new contact on the search bar. Got lost on tabs searching for contacts and requests.</td>
</tr>
<tr>
<td>Edit Contacts</td>
<td>🟢 Found Contact details very easily.</td>
<td>🟢 Found Contact details very easily.</td>
<td>🟢 Found Contact details very easily.</td>
</tr>
<tr>
<td>Make Match</td>
<td>🟢 When the notification was received, found out how to interact with it easily.</td>
<td>🟢 Interface was intuitive but some code errors were experienced.</td>
<td>🟡 Location tab not intuitive. User needed direction. Needed hint to know that element ‘match’ on list was clickable.</td>
</tr>
</tbody>
</table>
## Conclusions of the test

<table>
<thead>
<tr>
<th>Identified Problem</th>
<th>Type</th>
<th>Next Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>First page is confusing, specially if the user is already registered.</td>
<td>UX</td>
<td>Delete first page, make it last 2 seconds or make the button say ok, next, access…</td>
</tr>
</tbody>
</table>
| Phone sign in is confusing because it’s both sign up and login, and there are too many options on the page | UX   | Make two separate flows.  
  - Sign in: phone + extra (username, email, password)  
  - Login: Login with password (link to login with phone + sms) |
| Links below the sign in button (with phone) are distracting and confusing         | UX   | Make them look like the ‘go to sign in’ link in login page.                                                                             |
| Sending SMS takes too long. It is confusing for the user because there is no immediate feedback. | UX   | Add a loading page after introducing phone number, till the verification code is sent.                                                 |
| ‘No user with this phone number’ appears when login with a username with numbers. | Bug  | Change configurations on the login page.                                                                                              |
| Edit Personal info is not intuitive for some users.                               | UX   | When ‘Edit’ button is clicked, edit options should be more visible, to let the user know what they can edit.                             |
| Search bar is confusing. Users try to use it to find new contacts.                | UX   | Search bar could be hidden at loading and appear only when you want to search (with a button or going up). Or the text could simply say ‘filter’. |
| Tabs are not intuitive (specially the location tab)                               | UX   | Requests could be inside the contacts tab as a sub-tab. Location tab should be renamed and its icon could change as well. Some users suggested a loupe. |
| Add tag on Contacts was hard to find                                              | UX   | Put a bigger button on the bottom of the screen with a ‘+’ symbol. (Probably a Floating Action Button).                                |
| User expects to click on tab just one time and go to the main page for that tab. Now it is necessary to click a second time. | UX   | On Angular router clear page history when changing tabs or when exiting non-main pages.                                               |
| The distance on the location match shows only the range on the parameter.         | UX   | This was surprising for some users as they would expect the actual distance, exact or approximated.                                   |
| In some cases the user received a notification that a user was interested to meet, but they were actually the senders of that petition. | Bug  | Probably a minor change of code in the server would do, but definitely necessary to solve.                                           |
| ‘Open chat’ does not act like the ‘interested’ option in match requests.          | Bug  | Minor change on frontend app code. Open chat should both open chat and mark match as ‘fulfilled’.                                    |
| One user left the location tab expecting to find match requests on the requests tabs. | UX   | Make tabs more clear and somehow show that the matches on the location tab are clickable.                                             |
Conclusions

This project was ambitious from the beginning. I wanted to build an application ready for production and that implies taking care of a lot of different matters. The backend had to be secured enough and the app had to be easy to use and intuitive. The communication from the client to the server had to be secured as well and the server had to be able to communicate with the client asynchronously. Additionally, to add coherence to the app I wanted to build a brand around the app and use it to decide how it would look. We can say these objectives were achieved, up to a point.

I did build the app to run on Android and iOS and I did it following an agile methodology, that allowed me to implement CI/CD through GitHub Actions. I built a security layer on the server to control endpoint access and the most delicate information is encrypted on the DB. Even if someone could access the database, they would not be able to use users’ passwords. I conducted some research on the most used technologies, explored different options and finally made the decisions that built the project. I built a brand and decided how the look and feel would be. At the end, I conducted a UX testing session in which we could extract a lot of information useful for the next steps of the project.

The built application satisfies the basic requirements of the project. There were some more functionalities mentioned in this document that we did not implement, but we ended with a working version of the product. The addition of more functionalities is only a matter of doing more iterations on the project.

Other objectives we had were that the app had to be efficient and non-intrusive. As for the efficiency I used a plugin to track the location that cannot be controlled as much as it would be necessary. So, although it works, this would need a revision. The notification system should also be improved in further iterations with the more than probable decision to implement Push Notifications before going to production.

In conclusion, the app is not ready to go on full production, but it is in a working state. It could easily go on beta testing in a few more iterations.

Personally, I learned a lot during this project. I had the chance to work on technologies such as Java Spring or Ionic that I had never worked with before. It was interesting to build an app for iOS and see the process to develop it.

Doing a project like this on my own made me realise all the advantages of working in a team and the little advantages of working alone. Although it was interesting having to make this many decisions by myself and having the chance to work on all the layers of the same project.
Future Work

After the UX testing, we identified some issues of the interface that needed to be improved. Some minor problems with the code were also identified but fixed for delivery. In the next iterations of the project, we should include applying these improvements and work to take care of all the little details of the app. That would include interface flows, texts and the look and feel.

As we concluded above, the geolocation tracking should also be improved as well as the notification system. Functionalities such as the implementations of lists or the differentiation of different regions of the map would be implemented in future iterations.

Finally, if the app is to be actually put on production, we would need a financial plan and start spending money on deployment. Heroku is a good platform to deploy the app, but we would need a paid plan to host the server for production. We would also need to create an Apple Developer Account and license our app, in order to offer it on the App Store. To add the app to the Play Store we would also have to pay Google's fees.
References

Angular
Angular. Communicating with backend services using HTTP. <https://angular.io/guide/http>

Ionic
Ionic. UI Components. <https://ionicframework.com/docs/components> [ September 17, 2021 ]

Rest API

Spring
Nguyen Nam Thai. What is a Spring Bean?. Baeldung <https://www.baeldung.com/spring-bean> [ September 17, 2021 ]
Loredana Crusoveanu. Intro to Inversion of Control and Dependency Injection with Spring.

Spring Data

Spring Security


**Web Sockets**


**Deployment**

Annexes

Annex I: Manual

Server
The server is currently running on Heroku. To open the Swagger Documentation go to the next link (it might take some time if the server is sleeping, as Heroku’s free plan puts the app to sleep after some time of inactivity):


To try any commands on the backend you can use swagger or connect to the endpoint directly at:

https://nearby-apirest-deploy.herokuapp.com/<endpoint>

To run the server on local host we have two ways:

- IntelliJ IDEA: this is the IDE we developed the java spring project with. We just have to open the project ( folder: src/backend/ServerAPI ) and run the main class restapi.BackEndApplication

- Command Line:
  1. If maven is not installed, install it following the instructions on: Install Maven. Alternatively, if you have Homebrew you can install maven with the command:

     brew install maven

  2. Run the following comand:

     mvn spring-boot:run

The server normally opens on the port 8080, if it is available. Open Swagger Documentation with:


App Client
The front application is conﬁgured to the backend deployed on Heroku. It is already converted into Android and iOS and I don’t recommend changing any configuration on the base web project because it would have to be updated in the mobile versions with capacitor and we would have to manually change some configurations on Android Studio.

To run the application you will need the native IDE for each platform. Both app versions can be run on a simulator or a real device:

- iOS: you will need AppCode or Xcode to run the app (Mac needed). From both, open project folder: codi/frontend/front-app/ios

- Android: you will need Android Studio to run the app. Open project folder: codi/frontend/front-app/android. This can also be run on IntelliJ installing the Android SDK.
Annex II: Sprint Details

Sprint 1

During this first Sprint, we need to create the project, both for the frontend and the backend. We want to have our server online as soon as possible, so we will deploy it in this sprint as well. Apart from setting up the project, we thought the first tasks we should take from the backlog should be the ones related to the authentication system. The reason why is that we will need the users to be logged in for the next tasks.

So, at the beginning of the sprint, we estimated the approximate time it would take to complete each story. As we recorded how the sprint went on, we can now evaluate the estimation (see figure X).

<table>
<thead>
<tr>
<th></th>
<th>Estimated (h)</th>
<th>Real (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>US022</td>
<td>15</td>
<td>21</td>
</tr>
<tr>
<td>US023</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>US024</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>Setup</td>
<td>15</td>
<td>18</td>
</tr>
</tbody>
</table>

Fig X. Estimated vs Real hours spent for User Story in Sprint 1

A common way to evaluate the work done during a sprint is a Burndown chart (figure X), where we show the total estimated and real-time for the first day and the time left to spend for the following days.

Fig x. Sprint 1 Burndown Chart
Sprint 2

<table>
<thead>
<tr>
<th></th>
<th>Estimated (h)</th>
<th>Real (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>US022</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>US023</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>US024</td>
<td>5</td>
<td>3,5</td>
</tr>
<tr>
<td>US001</td>
<td>4</td>
<td>4,5</td>
</tr>
<tr>
<td>US002</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>US025</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>Security</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>UX</td>
<td>15</td>
<td>18,5</td>
</tr>
</tbody>
</table>

Fig X. Estimated vs Real hours spent for User Story in Sprint 2

Burndown Chart of Sprint 2

Fig x. Sprint 2 Burndown Chart
Sprint 3

<table>
<thead>
<tr>
<th>User Story</th>
<th>Estimated (h)</th>
<th>Real (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>US022</td>
<td>17</td>
<td>38</td>
</tr>
<tr>
<td>US024</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>US002</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>US025</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>US005</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td>Security</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Fig X. Estimated vs Real hours spent for User Story in Sprint 3

Burndown Chart of Sprint 3

Fig x. Sprint 3 Burndown Chart
Sprint 4

<table>
<thead>
<tr>
<th></th>
<th>Estimated (h)</th>
<th>Real (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>US005</td>
<td>40</td>
<td>52</td>
</tr>
<tr>
<td>US006</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>US0007</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig X. Estimated vs Real hours spent for User Story in Sprint 4

Fig x. Sprint 4 Burndown Chart
## Sprint 5

<table>
<thead>
<tr>
<th>US005</th>
<th>20</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>US006</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>US007</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>US008</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>US011</td>
<td>1</td>
<td>3.5</td>
</tr>
<tr>
<td>US012</td>
<td>1</td>
<td>4.5</td>
</tr>
<tr>
<td>US014</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Fig X. Estimated vs Real hours spent for User Story in Sprint 5

![Burndown Chart of Sprint 5](image)

Fig x. Sprint 5 Burndown Chart