Human Computer Interaction
Exam Final Report
EVs, a webapp for events management

Lorenzo Pisaneschi*,
Master's Degree in Computer Engineering, University of Studies of Florence, Florence, Italy
Email: *lorenzo.pisaneschi1@stud.unifi.it,

***Abstract*—Human Computer Interaction is a discipline which concentrates on the interaction between humans an machines using well defined interfaces in order to lead users to their goals and to a good use experience. Between such interfaces we count also the GUI (Graphic User Interfaces), which are now widely used and accessible to anyone to carry out countless operations of the most disparate categories: we can think about GUI used in e-commerce, home banking and social networking, for example. In this work I present my prototype project called EVs, a web application thought to help people to find events everywhere, in their city but also on holiday. The aim of this project is to develop from the high level design to the low level code implementation an user friendly application, using modern tooling and implementation approaches. I will describe all the process, from idea to design to existing EVs app to final usability tests, which provide an important feedback about the work done and give hints about potential future developments and UX improvements.**

***Index Terms*—back-end, design, front-end, prototyping, python, usability, UX, vue, wire-framing**

I. Introduction

When we want to develop a web application, we absolutely can not start directly from coding: there is a series of steps to be followed (with many variants), which guide us through all the development process: we surely start from an idea, which can be related to any task we think could be simplified; then we will do some analysis about requirements and potential users needs, in order to start a design process aimed at defining how effectively our product will be used and what will be its look and feel. Finally, after an intermediate step of prototyping, and only after we have a clear idea about the substance of our project and its effectiveness, it will be possible to design our application from the code point of view: we select our development frameworks, both for back-end and front-end, we plan models and database structure, we define our REST API, we do low level configurations and so on. Simply, we start coding based on all the design aspects come out from all the previous work of analysis.
In this paper I will introduce all these aspects. As we can imagine, there are tools which help during the upstream project analysis: so I will talk about **Miro** [1], used for UX development and **Figma** [2], a powerful GUI design tool. Then I will present how was the coding phase and do some considerations about usability tests and users feedback about EVs prototype.

II. Users mapping

A. EVs idea

The idea on which this project lives is the following: some social networks which make us able to deal with events already exist (for example Facebook Events) but they are underrated by the users, which are not so confident about the events that are shown. This confidence problem is related to the fact that often many events representing the same show or party, for example, are displayed to the user with different information, creating a misleading situation; furthermore, users are not always sure about who is the organizer of a specific event. Finally, it has to be notice that social networks as Facebook provide many features and could be dispersive. We have also to consider that it could be fantastic to have only one place where look for events, since at the moment we have to search on many different web pages to find useful information about interesting manifestations. Another consideration on why I thought about the creation of a management web app is that maybe users prefer to have things separated: to give one more example, consider Facebook again. It allows their users, which are billions, to send messages and post photos, among many other nice features: so why people use massively *WhatsApp*, which is an instant messaging app, and *Instagram*, which essentially allows to post only photos and images? Why not to do all in one place?

B. Need-finding

Clearly an application has to be ”user oriented”: a good application solves users' problems, makes their life easier and has to be designed in a way that does *not make them think* [3]. So the question now is: is my project idea potentially useful for end users?
To answer this question I created a Google Form to make simple interviews, collect data and considerations about my idea from the potential users point of view. This process is called **need-finding** and is an important phase of development because for the first time design team and developers face their final customers to know about something to be created in the better way possible, that is, in order to satisfy clients' needs and expectations.
Forty people have participated to this survey. They have been asked to answer question about their personal information

(gender, age and occupation) in the first part; then the second part, more specific questions have been presented: for example survey participants were asked what they think about the ”Events” section of Facebook, how they collect information about events when on holiday or in their city and how would they like to search, retrieve and manage events on a potential application. It is evident the goal: shape my application idea on users’ needs.
This survey has been compiled with enthusiasm and I have been given many advice on possible features to be implemented. Since the purpose of this project is to develop a working prototype, I have simply organized common ideas to cover all the requests came out from these interviewees. The majority of people likes to program their holidays, knowing in advance what to do; there are also users which like to take a last minute decision about what to do on a night out; generally, everyone would like to have an application where each event is detailed (venue, date and time, tickets, website), where they can indicate if will participate, and where a personal area exist to manage own events.

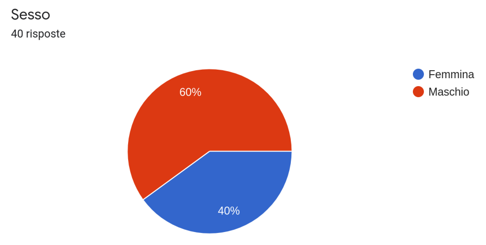


Fig. 1. Gender distribution of respondents.

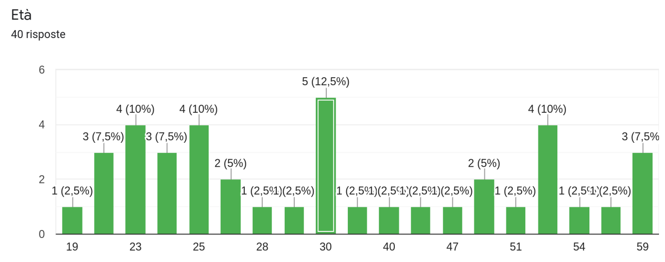


Fig. 2. Age distribution of respondents.

C. Personas

As stated before, need fining is crucial to understand what users expect from a product; on the other hand, find commons requests and expectations by user is important to develop something which is useful to the largest possible audience of people. On this ways, design begins by outlining **personas** [4], which are abstractions of people which we think will use our application. Personas can be thought as people which use our product to complete a specific task. Such scenarios give us design solutions to be used for a better usability. Scenarios where our personas act must have the right level of granularity and to be an high level representation of some task completion. Personas make us aware about real users’ frustrations, goals and needs doing some operation.
For EVs I considered that there are two main categories of final customers: **simple users** (that is, persons which will enjoy events) and **organizers**. For this reason I have considered four personas, two organizers and two users, of different age and occupations. User personas came from my need fining survey, while organizers are imagined.

- **Ilaria, user**: student, 23 years old who loves to perfectly plane her holidays;
- **Alessandro, user**: employee, 45 years old, who works a lot, has few free time which likes to spend with his family;
- **Paola, organizer**: employee, 40 years old, state worker who would like to start a collaboration with a start-app to create a modern application to promote city events;
- **Michele, organizer**: employee, 35 years old, who works at important events tickets provider company;

D. User story mapping and site map

Now that personas are set up, thanks to need finding and a little idealization on what an organizer could do when creates events, it is time to go through **user story mapping** [5]. User story mapping is a lean UX mapping method that helps to outline the interactions that we expect final users will do using our digital product to complete their tasks. Starting from personas, with user story mapping we can depict some main common tasks that users will perform. In particular, a user story map depicts three different actions at different granularity:

- **Activities**: high level tasks that users aim to complete using our application.
- **Steps**: more detailed actions to be done in order to complete an activity.
- **Details**: describe the lowest granularity of user interactions.

Before write activities, steps and details some brainstorming is necessary to discover them. To this aim, I selected two personas which I developed before, specifically Ilaria and Michele and I proceeded considering a brief sentence that framed the user and its needs. For Ilaria, as an example, I considered the following: *”Ilaria is planning an holiday, for this reason she wants to know in advance events that will take place where she will stay”*. Given this sentence, that is coherent with the Ilaria persona, I have started to think about very high level actions that an user like Ilaria could have done. Once identified them, I have worked on three main topics for each high level action. These topics are **”Design Ideas”**, which are about what I expected to be aware during the implementation phase of some aspect involved in the analyzed user story, **”Comment and Considerations”**, which

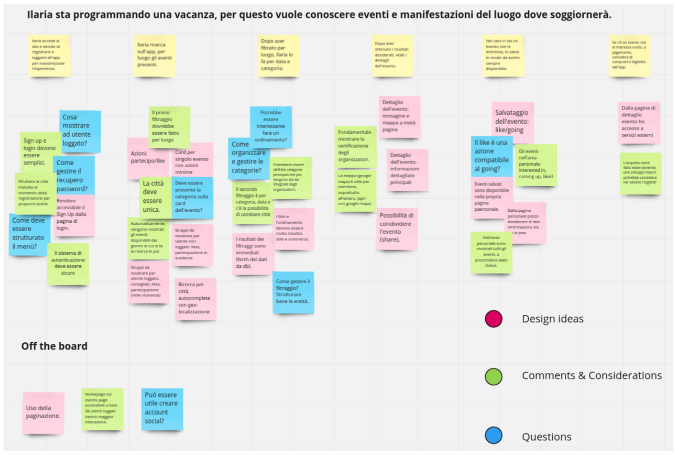

Ilaria sta programmando una vacanza, per questo vuole conoscere eventi e manifestazioni del luogo dove soggiornerà.
Off the board
Design ideas
Comments & Considerations
Questions

Fig. 3. First part of user story mapping: brainstorming on Ilaria persona on a Miro board. Yellow notes are the high level Ilaria's actions I expect to be done interacting with EVs.

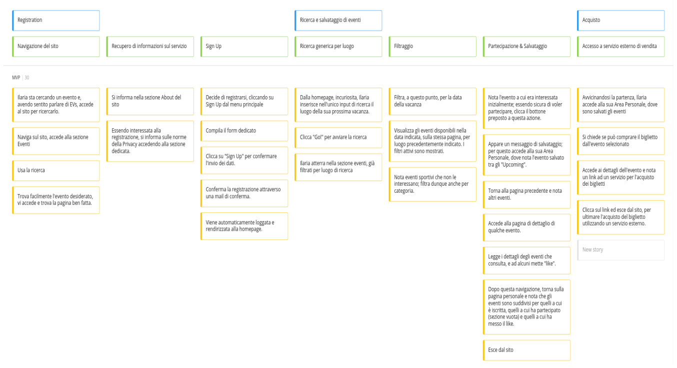

Registration
Navigazione del sito
Recupero di informazioni sul servizio
Sign Up
Ricerca e salvataggio di eventi
Ricerca generica per luogo
Filtraggio
Partecipazione & Salvataggio
Acquisto
Accesso a servizio esterno di vendita

Fig. 4. Second part of user story mapping: a Miro board which show the Ilaria actions (blue spots), steps (green spots) and details (green ones) obtained from information processing in the previous brainstorming phase. This board has to be read in horizontal for actions; each action is organized in steps, readable in horizontal also, while each step is detailed by green spots, to be read in vertical instead.

are generic remarks, and **"Questions"**, which are points to be explored.

This mapping has been done, as mentioned before, using the Ilaria persona, to explore what an Ev's customer user could do with the app and, on the other hand, I also used the Michele persona, as representative of the category of organizers. This work, as it may seem an end in itself, has proved extremely useful when programming, because many fundamental aspects to be covered have come up. Anyway, it is important to stress that this kind of activities are usually done inside an Agile team. As far as possible, I tried to follow the **Agile manifesto principles** [6], both in design and software development phases, because I really believe in its principles, which I think they are modern and dynamic, as a software exactly could be. I will return on these principles in the paragraph dedicated to EVs development.
Well, now that both customer and organizer personas has been brainstormed and essential points have been come to the surface, we can get closer to final site map, paginating the real user's story maps from previous information processing.

Again, I have wrote a map like the one in fig. 4 also for organizer users, for distinguish the two different users of Evs. The next step to the final goal of building a site map, is **card sorting** [7]. *"Card sorting is a UX research method in which study participants group individual labels written on notecards according to criteria that make sense to them. This method uncovers how the target audience's domain knowledge is structured, and it serves to create an information architecture that matches users' expectations."* All is in this definition of card sorting: the goal of this UX technique is to define an information architecture that makes sense to all the previous processing, getting closer and closer to the final site map, to how site pages will be effectively organized in final product. Starting from user story maps, which contain the main actions we think will be done by users, we identify the main topics under which summarize actions and pages' responsibilities.

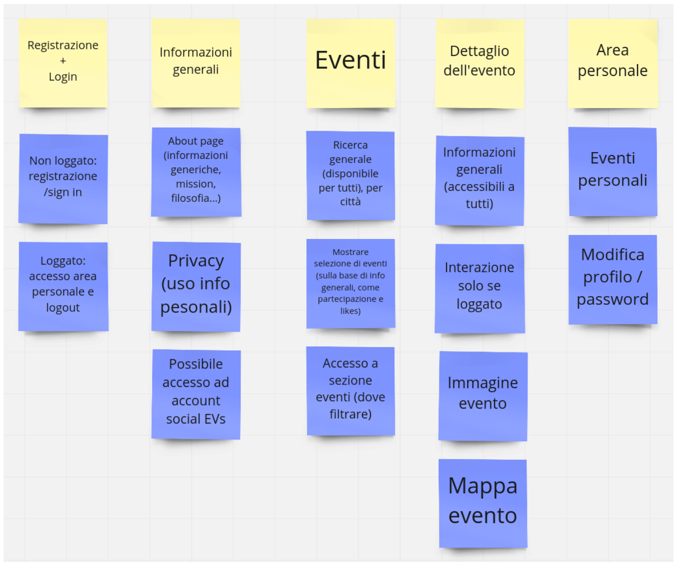

Registrazione + Login
Informazioni generali
Eventi
Dettaglio dell'evento
Area personale
Non loggato: registrazione /sign in
About page (informazioni generiche, mission, filosofia...)
Ricerca generale (disponibile per tutti), per città
Informazioni generali (accessibili a tutti)
Eventi personali
Loggato: accesso area personale e logout
Privacy (uso info pesonali)
Mostrare selezione di eventi (sulla base di info generali, come partecipazione e likes)
Interazione solo se loggato
Modifica profilo / password
Possibile accesso ad account social EVs
Accesso a sezione eventi (dove filtrare)
Immagine evento
Mappa evento

Fig. 5. Third part of user story mapping, the card sorting Miro board. The main topics (yellow notecards) are split in high level cases which lead us to the final site map which synthesize the overall site architecture in its final shape.

After all this work, finally the sitemap. The sitemap is the very first element which represents what effectively will be developed. It contains the views with their names, links that exists between different pages, and actions which can be performed in each page. EVs will be structured as follow:

- **Navigation menu**. The navigation menu will we displayed on a header in each page, to let site browse experience as comfortable as possible.
- **Homepage**. It will contains best events (measured in terms of interest and number of participants) divided in "Most participated", "Most interested" and "Expiring"; there will be also a search form where user can indicates the city she/he is interested in viewing the events of.

- **Logo**. The first menu item is the logo, which is really important to give an identity to a product. It will be clickable to redirect user to Homepage.
- **About page**. The About page is where site philosophy is displayed: ideas, product owners information and mission are some topics which are here displayed. As for the logo, the About page is important in product identity terms and also to get a good reputation.
- **Sign in / Sign up**. These items are used to let user to login or register a new account in EVs. However, I imagined that EVs should have been usable by anyone, also who is not registered, with some restrictions (for example, events management and interactions: a non registered user is able only to see events, but not to indicate if he likes it or will going to).
- **Events** This page is the app core, since here customer users can search and filters events by different criteria (for example, date range and type of events).
- **Notifications** In this page, as for classic social network, notifications are displayed. An user is notified every time that an event to which she/he will participate or is interested in is changed by its organizer.
- **Profile** In this page, the user can access his Personal Area, where it is possible to visualize the personal events, manage them and change Profile information, as well as upload a profile photo.
- **FAQ** Here an user can read Frequent Asked Question for troubleshotting
- **Contacts** Here an user can send an email to EVs for problems experienced using the application.
- **404** This page is shown when an error occurs.

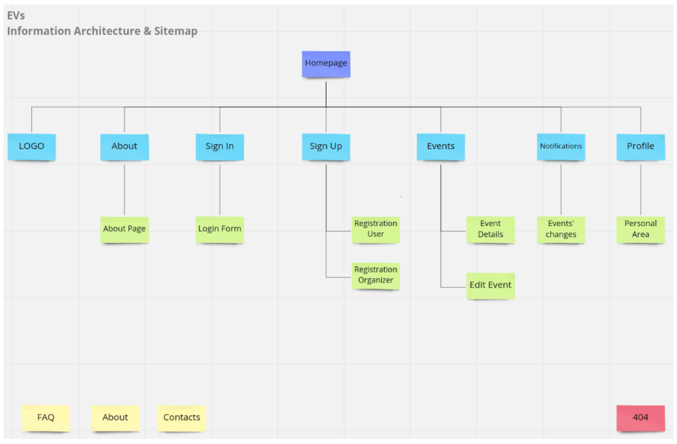


Fig. 6. EVs final sitemap. This map summarizes all the work done in this phase: from personas, we were able to do some brainstorming to understand principal user needs, which we considered to be exploited via user story map actions, steps and steps' details; then, we have further condense all with card sorting to finally obtain this site map, which is valid both for customer users and organizers.

The EVs information architecture and users needs are now clearer. This phase was hard and difficult, but as mentioned above it turned out to be one of the more important part of all the project, beyond the design phase. These results have significantly make simpler and clearer the design phase, which I describe in next paragraph.

III. EVs design and style

Information architecture and very first application ideas are now defined. I started from my personal idea of EVs, I then gathered information with need-finding; from collected information I defined my personas, both for customer and organizer users, and thank to this users abstraction I set up my Miro boards for user story mapping and information architecture, summarized in a net and clean site map. This is the base from which the styling and actual site design process starts. As stated before, I have used Figma to do this task. Figma allows to do everything for design: in fact, I split this phase in sub tasks, that is **Wireframing**, definition of **Style Guide**, site **Components** and final **Design**. Finally, always thanks to Figma, I did a little work also on **Prototyping**. In the follow I will explain everything about all of what I mentioned.

A. *Wireframing*

Wireframing is where design begins. Wireframes are firts sketches on what is going to be build. Starting from site map, in my case, I have defined all my site pages: Homepage, Events page, pages used for main information as FAQ, About and Contact Us, Sign Up and Sign In forms, forms used for created and edit events and personal area, the user/organizer profiles. It is important to think in a careful way about wireframes, since they are foundations on which design lives. In this phase the site style (as, for example, colors or fonts, button and input shape) is not covered and is not taken into consideration; what really matters is that wireframes should fit well the needs come out in previous phase, in order to build a functional, easy to use and clear interface, able to guide user in site navigation to let him achieve his goals using the app. To conclude, we could say that wireframes are the underlying structure of a web application, for the front-end part at least.

B. *Style Guide*

In this paper I only included some pages which I created during wireframing, however everything is available on my Figma Human Computer Interaction personal project [8]. Now we can talk about **style**. Style is important: primary and secondary colors, fonts and icons guide user through all the application; they are also crucial for pleasure and satisfaction of use. Style guide composition allow designers and developers to think about spacing (literally **margins** and **paddings** inside html and css classes) and principal components dimension (such as buttons, inputs, modal, badges and chips and textarea). In fact, after style guide is defined, it must be used in components definition to ensure homogeneity throughout the application.
Colors define a *color palette*. Primary colors are the most used and present inside the app, while secondary are marginal. Then it is possible to define other colors for other needs: for example for this projects I have also defined *success*, *warning* and

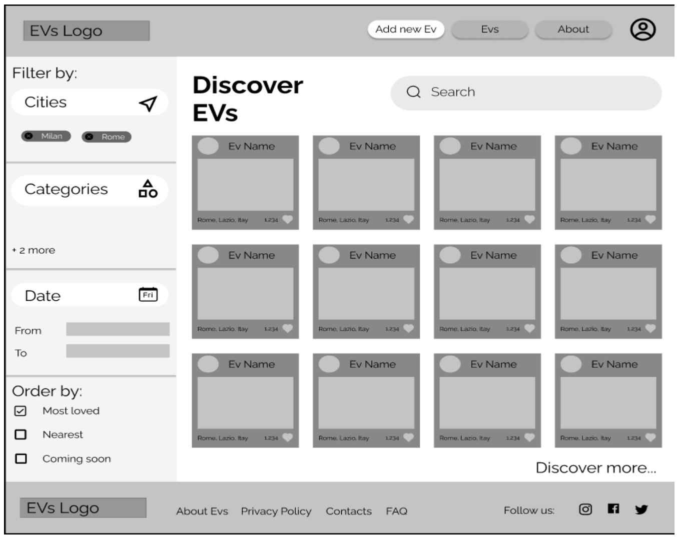


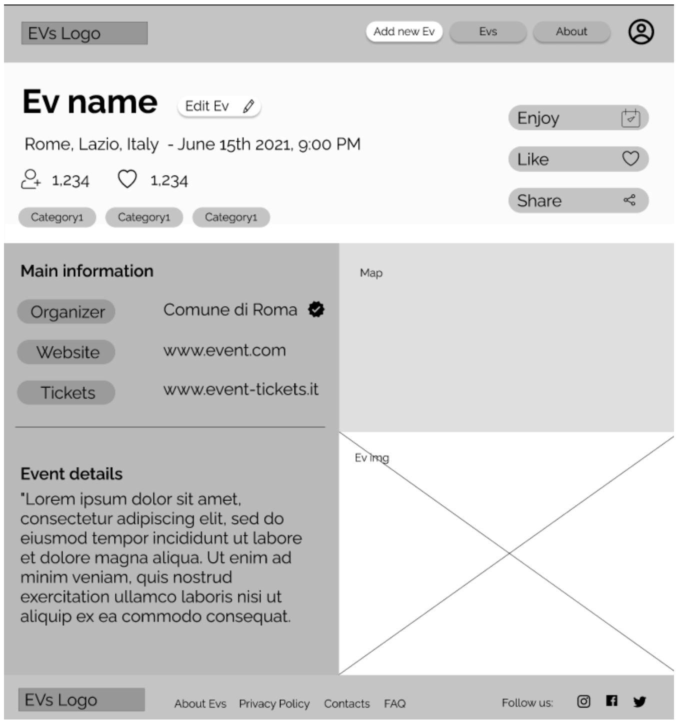
Fig. 7. The events page, an EVs wireframe example. This page representation is abstract and easy to understand, ready to be styled.

danger colors (the now consolidated green, yellow/orange and red in order) to better communicate to users what is happening during navigation and interaction with the website. Always in relation with colors, it is important to be aware about contrast inside the app. It should be always verified when defining colored components with text, for example buttons or chips: for this, many online tools exist: one is **Contrast Checker** [9] which checks if a site accessibility is consistent with the **Web Content Accessibility Guidelines** [10]. To conclude, fonts and icons. I used **Roboto** a much used font by Google [11], while for icons I used the **Material Icons**, by Google again [12]. About font is important to notice that inside Style Guide we have to indicates font dimensions to be used in different site sections: surely headings will be different by tooltips, which themselves are different by paragraphs and so one.

C. Design

Design process is all about little parts to be put together to build the bigger ones and so on. From style guide, we put together little parts to obtain the elementary components which are combined to define design.
I defined the components listed below:

- **Inputs**. Inputs are essentially elements which make able user to insert information to be persisted on a database or to be used for filtering or searching tasks. Inputs are important: they allow to create accounts, events and search for events by some criteria, in EVs app.
- **Buttons**. Buttons are used to perform call to actions. Speaking from software engineering point of view, buttons are used to call some JavaScript function which requests for some back-end operations, in turn. Buttons must be evocative for user: they must follow consolidated design criteria, for example the correct positioning inside a modal (the "positive" button is commonly positioned on the right, the "negative" one on the left).



"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Fig. 8. Another wireframe example: the event detail page. I would stress again on how it simple for anyone to read this representation. As mentioned before, color, fonts, images and other highest level components are not complimented here.

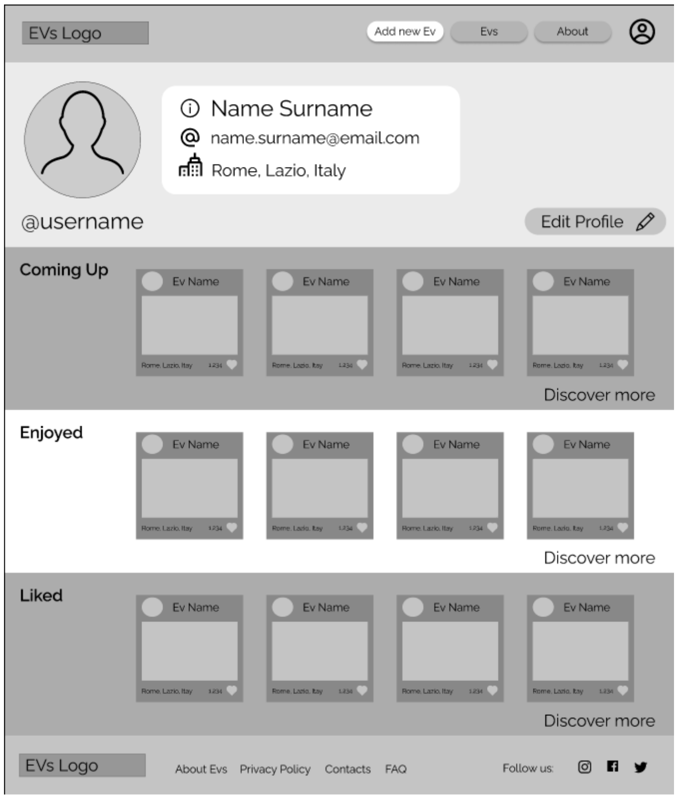


Fig. 9. Finally, the profile page wireframe. Here we can imagine how the app will works, interconnecting events and users, thanks to a continuous interaction between the two parts. Again, the representation is really bare.

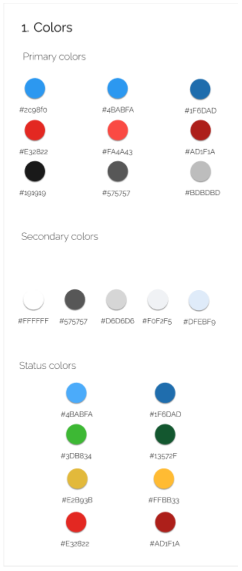
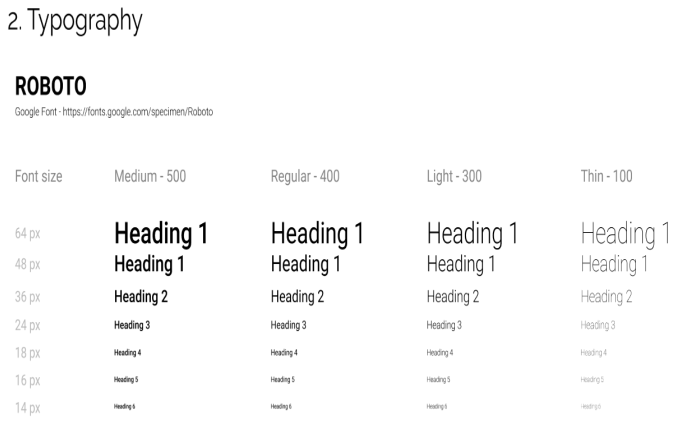


Fig. 10. EVs color palette defined inside the Style Guide. Here primary, secondary and ”service” color listed. Them will be used in the entire application: HEX codes are provided for an easier consultation and use.



2. Typography

ROBOTO

Google Font - https://fonts.google.com/specimen/Roboto

Font size	Medium - 500	Regular - 400	Light - 300	Thin - 100
64 px	Heading 1	Heading 1	Heading 1	Heading 1
48 px	Heading 1	Heading 1	Heading 1	Heading 1
36 px	Heading 2	Heading 2	Heading 2	Heading 2
24 px	Heading 3	Heading 3	Heading 3	Heading 3
18 px	Heading 4	Heading 4	Heading 4	Heading 4
16 px	Heading 5	Heading 5	Heading 5	Heading 5
14 px	Heading 6	Heading 6	Heading 6	Heading 6

Fig. 11. EVs fonts defined inside the Style Guide. Here we have to specify different font sizes and weights.

- **Header**. In EVs app, the header is a navigation menu. It allows user to sign up (if not registered), to sign in (if user has an account) and to visit the ”Events” and ”About” pages; once logged in, user can also visit his personal profile and view notifications (about changes on his events, which have been modified for dome reason). On the left, there is also a **logo**, which represent the whole site identity.
- **Footer**. The footer contains again, on the left, the Evs logo, and some useful links: one to the ”About page”, one to ”Contacts” page (where an user can send an email for help requests) and the last to the ”FAQ” page, where Frequently Asked Questions are listed.
- **Cards**. Cards are used to display events list, both in pages accessible to all and in personal profile page. In the event card main event’s information are provided: event name, venue, date, organizer and a representational image. There are also displayed number of participants and the number of people which is interested to the event itself. An example is in figure 13.

Fig. 12. EVs logo, designed by my friend Massimiliano Maestripieri.

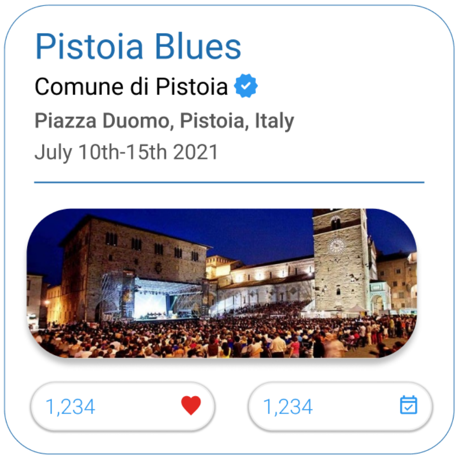


Fig. 13. The card component used to display main information about events. This component is fundamental all around the web application.

- **Tags and Badges**. These components are marginal, but useful for user’s orientation: they simply indicates events categories and their availability. I included here also the circled user profile image to be displayed inside the app, when requested.
- **Modal and Popups**. The last component I developed are modal and popups. Modals are useful because creates a better communication flow from application to users, mostly when user is going to perform delicate operations, such delete and update actions, which will overwrite or remove information about profile or events, the latter for users of class ”organizer”. In simple terms, modals give to users the possibility to rethink about what they are going to do. On the other ends, popups inform users about how an operation went, if there were errors or everything was fine, giving feedback, which are vital for a good application flow understanding.

Above I listed all the components which I thought and developed in order to define a final design system to be used to let users to feel comfortable when interact with the app. I tried to imagine as many scenarios as possible to be confident when developing. Some things have been changed while software developing, but the structure has been completely respected, and the design phase turned out to be fundamental.

D. Prototyping

The last design step is **prototyping**. Prototyping is the connection point between design and app implementation. It let to understand how the developed design is feasible and near to the needs identified during need-finding and interviews. It is the closing of the circle of the whole process before start developing. Figma makes prototyping easy to build up. It is possible to use the designed pages and link them with arrows which make us able to simulate the interaction between user and our designed app.
For EVs I wanted to check the user journeys for both user and organizer, so I tested the registration and login, the events research from Homepage by city name, then events filtering, events participation and interesting for user, events create and update for organizer. Everything looked good, so I was allowed to go to next step: the real software implementation and development environment setup.

IV. EVs SOFTWARE DEVELOPMENT

EVs webapp is based on two main frameworks, one for back-end and the other for front-end development: **Django** and **Vue** [13] [14]. Both are two powerful tools which make work easier: they have incredible functionalities which speedup productivity and lead to outstanding results in a short time; not by chance, Django motto is *”The web framework for perfectionists with deadlines”*. On one hand Django is completely written in Python, that is a fantastic programming language, easy to read, to learn and to understand; on the other hand, Vue is obviously a JavaScript framework, but really flexible with a lot of nice features. Before start the software development illustration, I rapidly motivate my choices and I will present all the EVs webapp ecosystem.

A. Project layout and principal components

Before starting software development, it is necessary to have clear in mind the whole process we want to follow while developing and the tools we want to use in development environment and production environment. I have mentioned Django and Vue yet: I have chosen these two frameworks both for my personal both because I wanted to learn them and for personal tastes. The main idea was to create two different project, one Django project and one Vue project, and link them creating the final web application. This choice led me not to take the most from Django templates system, which I used in only few parts of the application, but on the other hand it made me more free to experiment with Vue, on the front-end. Remaining always on the front-end, I chose **bootstrap** [15] to create the layout of my app, because I had a previous knowledge about it and I think that choosing bootstrap makes you always fall to your feet. It is evident that the idea is the following: create a Django application which interfaces

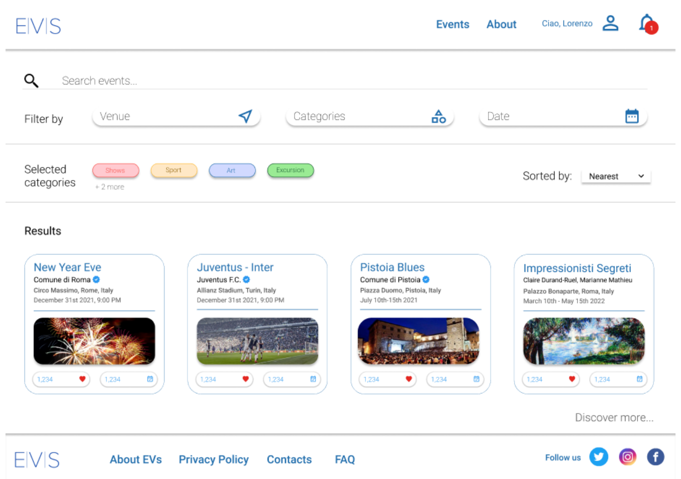


Fig. 14. The designed final events page. Here we can see cards, badges, inputs components and we can compare the same page, in its wireframing shape, at figure 7 .

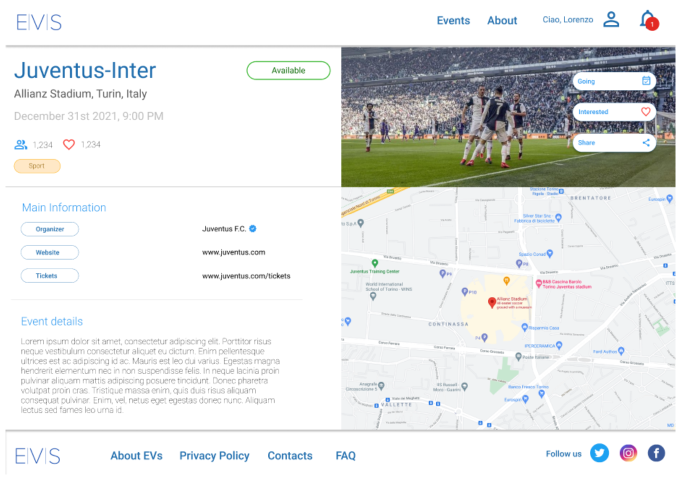


Fig. 15. The designed final detail event page. We can compare it with the wireframe at figure 8 .

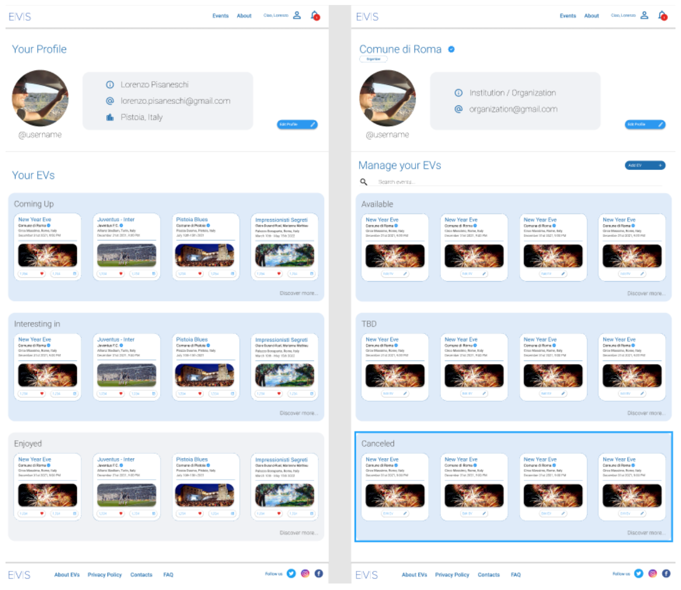


Fig. 16. The designed final profile pages, both for user and organizer. Compare with 9 .

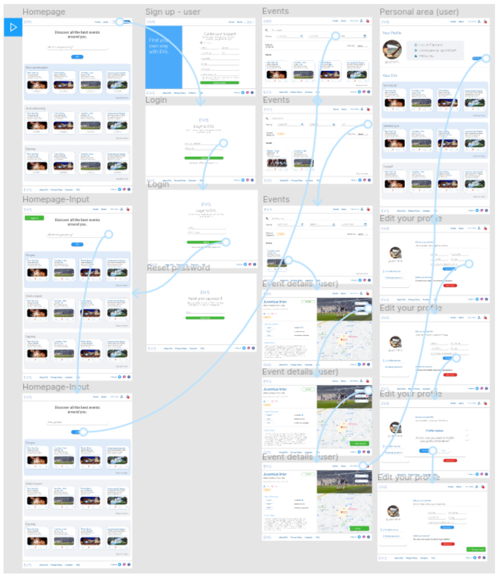


Fig. 17. User prototyping in Figma. This configuration allows designer to see their work in presentation mode, interacting with the elements created by them on the screen, in a similar way on how the final product will looks like.

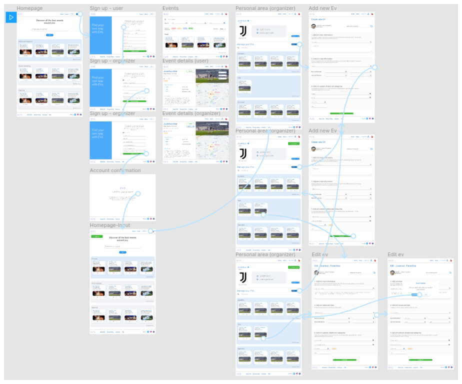


Fig. 18. The organizer prototyping in Figma. Prototyping gives a concrete idea on how product will be effectively.

with the database (PostreSQL) to retrieve, delete and modify various objects (that is, events and users) and prepares data to be served on the front-end with Vue. A smart way to do this is to use **REST API**, an approach I preferred to classic Django "url-view-template" system, precisely because there is Vue in the middle: so I worked with **Django REST Framework** [16], a library (whose developers are the same as Django) thought to build rapidly powerful REST APIs, using always urls and views concepts, which are at the core of Django, but introducing also **serializers**, which allow us to transform JSON objects to native Python data structure, which are in turn transformed into our own objects (Django models) and viceversa. Business logic lives inside serializers, where we effectively define the data shape to be delivered to front-end, and views, where we communicate with database and handle our objects. Summarizing, a Django project, assisted by DRF, manages data, exploiting Django ORM, and passes them to Vue, which presents them. This is a modern and very simple **Model View Controller** paradigm implementation. User uses the front-end build up with Vue to perform operation controlled by Django, which makes **CRUD** operations talking with database; the communication channel between user and the app exists thank to REST APIs.

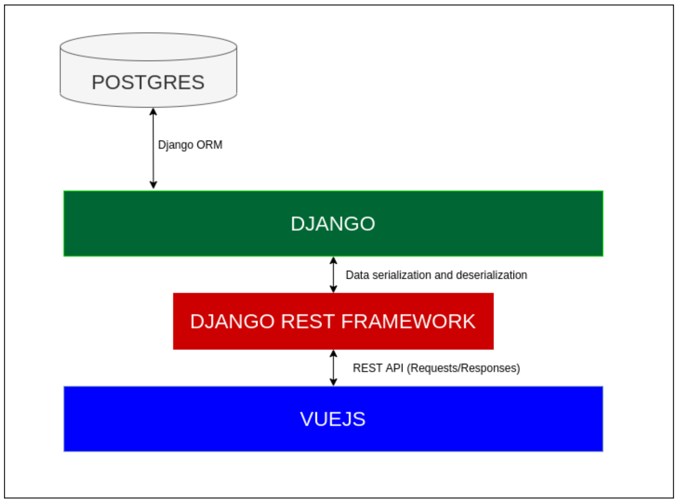

POSTGRES
Django ORM
DJANGO
Data serialization and deserialization
DJANGO REST FRAMEWORK
REST API (Requests/Responses)
VUEJS

Fig. 19. An high level illustration about EVs main structure DRF is a middle-ware between Django and Vue.

Since in design I defined the possibility for users and organizers to upload images (profile images and events images) I also needed a cloud storage where save these files. In this case, a natural choice is **AWS** (Amazon Web Services) [17], in particular, S3 Cloud Storage Service. Thanks to Django, is really simple to manage file uploads with S3, using well documented third party libraries as **whitenoise** [18], **django storages** [19] and **boto3** [20].
Since I needed to work with email, I used **Sendgrid** [21], an email service provider, for two main tasks:

- **Recover credentials**. If a user for some reason forgot the password, he should have the possibility to restore it. In EVs exist a form where users can insert the mail used at registration time to which the application will automatically send a recover link.
- **Contact Us**. In EVs app, an user is able to contact the site administration (for example for troubleshotting and suggestions) compiling a form which is used to send an email to EVs email inbox.

Finally, I used **Geo-localization APIS by Google** [22] to display the event map in its detail page, as shown in figure 15. As all developers know, development is a thing, production is another: when **deploying**, countless unforeseen events of the most diverse types can occur. I relied on **Heroku** [23], a deployment cloud platform. Heroku has everything I needed: a simple environment variables management and a great automatic deployment system. In fact, Heroku integrates easily with my development tools: an **Heroku CLI** exists to be used in IDE (personally I used **PyCharm**) and it is possible to link Heroku to a **GitHub repository**. Since EVs code lives right on GitHub [24] I have exploited this feature: every time I pushed on EVs origin, master branch, after test passing, the deployment task was triggered on Heroku itself.

B. Back-end development

I have decided to start from back-end implementation. Creating a Django project is straightforward: first of all, we have to create a **Python virtual environment** and to install Django via pip; then we can create a django project via *django-admin*, a command line tool wich is installed with django. A django project contains two important files, above the others:

- **settings.py**. As its name says, the *settings.py* file contains all the project configurations. When we install an external library to be used inside the project, it must be inserted there, and there there we define how it works, using fundamental python data structure, ad dictionaries or lists. For example, here I defined AWS S3 and SendGrid configurations.
- **urls.py**. In *urls.py* we define the entry points for our project. There we can also insert urls which came from other **django apps**, which can be either installed externally or created by ourselves. I talk about the django apps which I developed for this project soon.

For security and better readability I also used **python-dotenv**, an external library for storing environment variables (such as AWS keys) inside a file which is not published on GitHub. For packet management I used **setuptools**, python package to easily organize project packages. After I completed these low level configurations I started to think about database structure, i.e. about the **models** that interact with each other. It is a good practice, not only in Django but commonly in object oriented programming, to divide responsibilities into different modules: for this reasons we use **Django apps.** A Django app is not a Django project: a Django app is an independent module with its functionalities, while a Django project is a collection of configurations (defined inside *settings.py* file) and just Django apps. Based on the above concepts, I have broken down code in following parts:

- **AWS app**. This app is a middleware between the evs project and AWS services (s3 in this case). It defines an AWS Document model which is associated both with Users and Events.
- **Users app**. This app is responsible of all users management features, such as image profile upload and profile information handling. The two user types (”customer” and ”organizer”) are both handled here, thank to a simple discriminative Boolean field.
- **Events app**. This app is probably the core of all the project. It is responsible of events management. An event has relations with users, since it has an organizer (which is a User model instance) and a list of people which are interested and another list of people which are going to the event itself. The event model obviously defines all the event fields such as date, schedule, venue and so on.

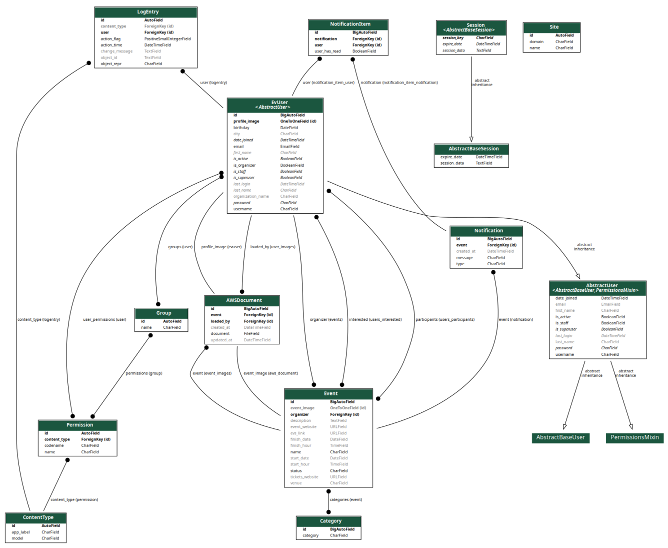


Fig. 20. Evsapp models and their relationships.

- **Notifications app**. This app is responsible to deliver notifications to users when an event to which they are going or are interested in, changes. Then, it has relationships both with users and events.

Since when an user loads a profile image or an organizer loads an event image I want to keep trace of these files on database (because maybe I would like that users and organizer may have the possibility to reuse an old image) every time there is an upload I have to save a record on AWS Document table; the same reasoning is valid for notifications: every time an event changes, I have to create a notification record which is to be associated to the people involved in the event. For these tasks, Django provides a **signals system**, to perform operations before or after elementary database operations such as create, update or delete. Inside each application, I then created an **api folder**, where the Django REST Framework code to build APIs lives: serializers, views and urls for the specific model. Application urls have been then included inside the evsapp Django project urls, which contains all the app entry points, as I stated before.
A great aspect of Django is that is comes with some boilerplate code for standard features, such as authentication system. In fact, EVs authentication system is based on **django-authentication** code for login, logout, password change and password retrieve actions. A great aspect of Django is that is comes with some boilerplate code for standard features, such as authentication system. I have only styled related templates, exploiting the Django templates system, for this task. Finally, I have to mention **automatic tests** and **CI**. I set up a test suite using Django **TestCase** and DRF **APITestCase** classes. Then, I added **tox**, a library writte to automate and standardize testing in Python and wrote a **tox.ini** file where I specified all the necessary. Since EVs project is hosted on Github, I defined some **GitHub actions** in order to run my tests with tox in a

CI environment every time a push occurs on any git branch or Pull Request. Automatic tests and Ci are fundamental I think: they force you to write net and clean code, which is consistent with the new features added over time.

C. *Front-end development*

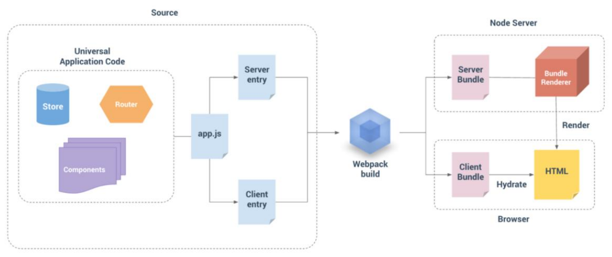


Fig. 21. A Vue project structure and its build process using webpack.

As I said before, EVs back-end and front-end live in two different dedicated project, one developed with Django, the other with Vue. The first thing to be done in this case was to integrate them, in order to have a working local development environment and to be sure everything will be working once in production, on Heroku. For this reason, after I have created the Vue project using **Node.js** [25] (assisted by the Node Package Manager, i.e. **npm**), I have gone for the use of **webpack-loader** in Django and **webpack bundle tracker** on the front-end to write two index html files, one for development and one for production: the development html file were responsible of integrate the vue development server inside a Django template, which is inserted inside the Django registration template, in turn; on the other hand, the production html file were responsible to load the bundles generated with the Vue build inside a Django template for production use. This mechanism is possible thank to a **vue.config.js** file where build directives are specified to work with Django. This configuration phase is not trivial, but once defined works fine.
Like most JavaScript frameworks, Vue works with the **components**: we have to organize our interface in many little reusable components, for a better development and an easier code maintainability and hygiene. Vue files are subdivided in three main parts:

- **Template**. In template we define the html to be rendered for the specific component.
- **Script**. In script part we write our logic for the specific component. We can define **computed properties**, **methods**, **data** and some objects to be used for components communication, such as **props** and **emit**, the latter for custom component events handling. Props are injected from parent to child component, while events are emitted by child to notify parent of something. Script properties can be used inside template, with the smart Vue **data binding system**. **Style**. In style section we can add css style for our html defined inside template.

A Vue project can be seen as four main parts:

- **App.vue**. This file is the project root, from which everything is mounted and initialized. Inside **main.js** file the App.vue is effectively mounted and whole app dependencies, as router and store, are defined.
- **Views and components**. Views are the entire pages to be loaded inside the web application, while components are parts of views, and should be reusable everywhere. Both views and components are vue files.
- **Router**. Vue router is responsible of pages serving. Inside a **index.js file**, pages are linked to their route. The router is callable from every vue file.
- **Store**. The store in Vue works thank to the powerful Vuex package, a Vue design library for state management. Vuex interfaces with back-end and saves data to make them accessible inside views and components, exploiting the Vue life-cycle hooks [26].

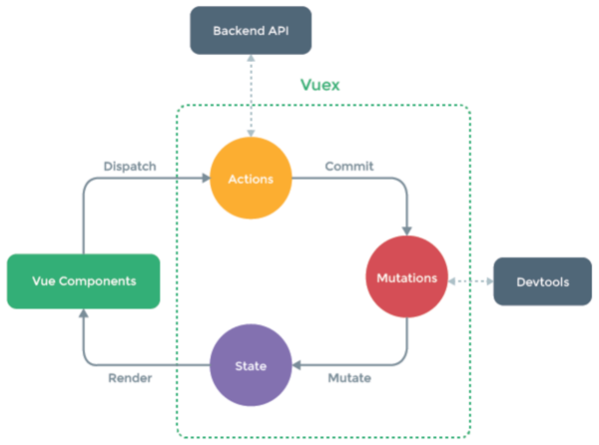


Fig. 22. Vuex. Vue components ask for data from back-end using the dispatch function on Vuex actions, which actually interfaces itself with back-end, handle responses and update Vue app state with mutations, in order to let available new fresh data to be rendered inside components.

For styling, as mentioned before, I used Bootstrap, which gave me the possibility to design a functional interfaces for desktop, mobile and tablets, with a view to responsiveness.
As developing process I tried to following a "light Agile" one. I was the only to work on this project, but I organized the work in little sprints lasting seven days, where I started with a planning, defining minimum goals to be reached within the sprint, effective development and a final retrospective to improve what have not worked. Sprints were designed to reach as quickly as possible the MVP (Minimum Valuable Product).

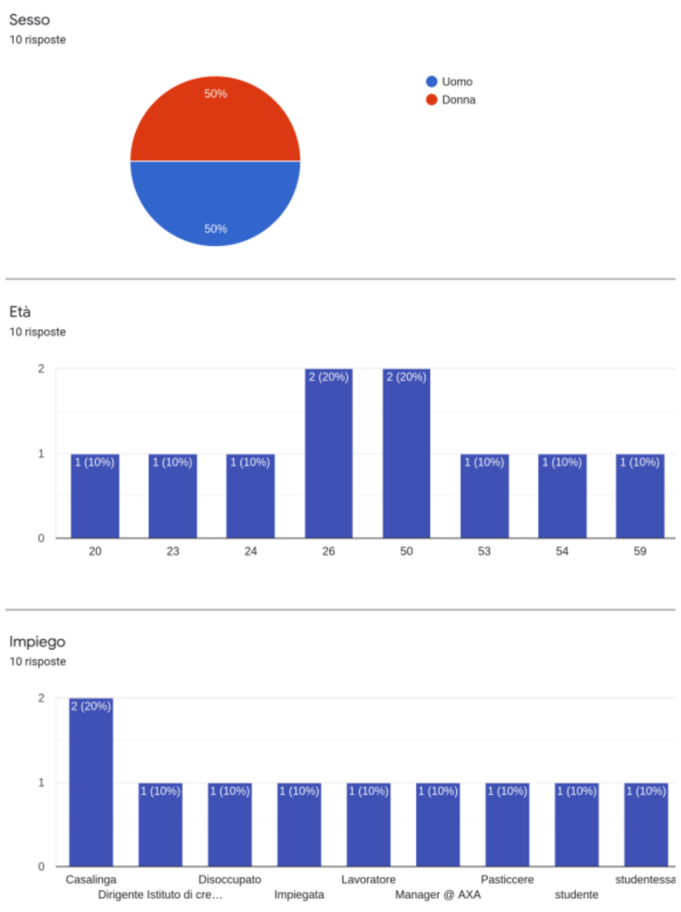
D. *Deployment*

As I wrote before, I used Heroku for deployment purposes. It is smart and makes deployment easy. First of all, I proceeded with a setup phase, creating the Heroku app, adding a **Procfile** [27] inside my repository root (containing the commands to be executed by app startup) and exporting my local database structure to be extended in future deployments. Then I proceeded incrementally as my project grew, adding necessary environment variables and **buildpacks**: first I added the python buildpack since initial deployments contained only the back-end project with Django; then I also added the Node.js buildpack, when deploying also the front-end. For Django project deploy was necessary to include also a **runtime.txt** file where the python version to be used (3.9.5) is specified, while for Node.js and Vue I had to specify two file: a **package.json** file at root project, with a dedicated script to navigate to the front-end directory and perform build to create the bundles to be loaded inside the *index.html* and a **static.json** file containing information needed for Vue app deploy on Heroku. Including these files we tell to Heroku how our application is build and let Heroku itself to automatically collect all the information to follow a correct deployment flow.

V. Scenarios and usability tests

Usability tests are an important practice which make us aware about what works and what not in our application. Generally, tests are carried out by potential final users, with modalities which give important indications for future development and improvements to be made. Summarizing, tests are set up to evaluate if the application is **user-friendly**, considering five main factors: **Learnability, Efficiency, Memorability, Errors, Satisfaction**. Obviously many kind of usability tests exist; in this case, I opted for the definition of some *scenarios*, that is real use cases that may arise using the product; scenarios are a widely used practice, because they should make the user forget to be tested. Scenarios have been asked to be resolved by tested people, which after had to fill out a questionnaire about how was their experience with EVs app. First of all is necessary to define the scenarios we want to show to our tested audience. I came up with nine scenarios that I thought covered most of the application's functionalities. Here it is to be noticed that I developed scenarios only for ”customer users” and not organizers. The scenarios are the following:

1) It is summertime and you want to check your city events: so you search events by city name.
2) You find the application interesting so you decide to create a profile to exploit at the most the app.
3) You want to complete your profile uploading a profile image.
4) Supposing that you will be on holiday in Florence, you search events in Florence, filtering by date and category.
5) Looking at the search results, you notice an event you want to know more about, then you access that event detail page.
6) Since you are satisfied about the info contained inside the detail page, you decide to indicate that you will participate, by clicking on ”going” button.
7) Now you want to check your events inside your profile, considering that in your profile there are upcoming, expired and saved events.
8) During navigation, you had some trouble: so first of all you read the About page, then the FAQ. However, since you did not find what you were looking for, decide to contact the site administration, sending an email.
9) Now you see a notification: then you access the Notification page.



https://evs-hci.herokuapp.com/

Fig. 23. Tested people gender, age and occupation distribution. Test have been submitted to 10 people, 50% male and 50% female, respondents age goes from 20 to 59 years. Occupation is various and could be divided in students and workers.

After the scenarios definition, I have written the questionnaire that have been filled by tested people. The questionnaire contained three registry questions (the answers to which are highlighted in figure 23), fifteen questions about user experience and a section where to write suggestions. Questions about user experience were SEQ (Single Ease Question), with a scale ranging from 1 (completely disagree) to 7 (completely agree).

With scenarios and questionnaire ready to be used, I conducted a **pilot test** to try out my test suite, then I started to looking for people to be tested. Tested people should be people which potentially use the application. Audience should be also heterogeneous in age, gender and occupation in order to have a broader view on tests result. I have tested

TABLE I
QUESTIONNAIRE RESULTS

Question	Average points μ	Variance σ^2
It was easy to find events searching	6.9	0.09
You had no difficulty in registering and modifying your profile	6.9	0.09
You think that events filtering as is in EVs is useful and functional for users needs	6.9	0.09
The interaction with an events (Like and Going functionalities) are easy to understand and intuitive	6.2	1.16
The detail event page contains all the information which are useful to know about an event	6.7	0.21
It is easy to manage personal events in Personal Area (Profile)	6.9	0.09
I think it is useless to know how many people will attend an event or are interested in	5.2	4.76
I think that if a person will attend an event should not be able to like it	3.3	1.41
Notifications are useless and/or not so clear	2	1.8
I think that how events are organized in homepage (most participated, most interested and expiring) is useless	1.89	3.43
I think the navigability of the application is pleasant and intuitive	6.6	0.44
I found easy to interact with the application	6.2	0.96
The layout of the pages and the contents are clear and explanatory	6.6	0.44
I believe that the service provided by the product designed is potentially useful, to be used regularly in the future	7	0
I am generally satisfied with the application	6.1	1.29

ten persons, making them complete the developed scenarios and compile the questionnaire; it is interesting to stress that respondents have also participated in the need-fining phase, closing a process started 3 months before. I start illustrating the suggestions that I received. Someone did not find the notifications counter very clear, and suggests to highlight it more; others suggest to change ”Like” and ”Going” buttons to let them be more evident on the page, and to give to the user the possibility to choice application language, which now is only in English. However the most controversial point is related to the ”Like” function, and I think is correct that it has generated some problems, for its ambiguity: respondents thought about the ”Like” button as a function to express an opinion about the event, while I designed it simply to save an event, about which user wants to be informed (through notifications) but he is not sure will attend; so maybe it is better to change the ”Like” to ”Interest in” or something similar, so as not to create confusion. Finally, other tested users have suggested to introduce a profiling system inside EVs, for example to let users know about events that may interested them in a dedicated section. Other notes were made on the layout which I think is always improvable. Generally, tested users were satisfied with the webapp.

VI. Conclusion and future developments

Suggestions and results came out from usability tests are important as a basis for future development. Surely labels, text and overall design can be improved, considering that for now, EVs is only a prototype; on the other hand, I think that language is crucial if the goal is to build a web application which is usable by the largest possible audience. Another point to consider is the fact EVs could be a social application: it could be interesting that users can give feedback to organizers about the events they have attended but also to other users about organizers themselves. Later, it might be logical to think about interaction between users inserting a following system as we are used to seeing with the main social networks; surely I'd like to add profiling in order to automatically suggest events to people considering, for example, categories of events most attended by users, geographic information such as living cities and so on. Anyway, I think that this the idea behind EVs is a winning one: It could be great to have only one place where search certified events not only when home, but all around the world, in a virtuous system also used by the tourist authorities.

References

[1] Miro, https://miro.com/.
[2] Figma, https://www.figma.com/.
[3] Steve Krug. Don't Make Me Think: a Common Sense Approach to Web Usability. New Riders, 2014.
[4] Scenario Mapping: Design Ideation Using Personas, https://www.nngroup.com/articles/scenario-mapping-personas/.
[5] Mapping User Stories in Agile, https://www.nngroup.com/articles/user-story-mapping/.
[6] Agile manifesto, https://www.agilealliance.org/agile101/the-agile-manifesto/.
[7] Card sorting, https://www.nngroup.com/articles/card-sorting-definition/.
[8] Figma Human Computer Interaction project. Wireframing, desing and prototyping for EVs. https://www.figma.com/file/woo5UXQ2Bk8rPAkm5L1oEc/HCI_Wireframing?node-id=0%3A1.
[9] Contrast checker, https://contrastchecker.com/.
[10] Contrast checker, https://www.w3.org/WAI/standards-guidelines/wcag/.
[11] Roboto font, https://fonts.google.com/specimen/Roboto.
[12] Google material icons, https://fonts.google.com/icons.
[13] Django project, https://www.djangoproject.com/.
[14] Vuejs, https://vuejs.org/.
[15] Vuejs, https://getbootstrap.com/.
[16] Django REST Framework, https://www.django-rest-framework.org/.
[17] Amazon Web Services, https://aws.amazon.com/it/
[18] Whitenoise, http://whitenoise.evans.io/en/stable/
[19] django-storages, https://django-storages.readthedocs.io/en/latest/
[20] Boto3, https://boto3.amazonaws.com/v1/documentation/api/latest/index.html
[21] Sendgrid, https://sendgrid.com/
[22] Google Maps APIs, https://cloud.google.com/maps-platform
[23] Heroku, https://www.heroku.com/
[24] EVs GitHub repository, https://github.com/pisalore/evs
[25] Nodejs, https://nodejs.org/en/
[26] Vue life-cycle hooks, https://v3.vuejs.org/guide/instance.html#lifecycle-hooks
[27] Heroku Procfile, https://devcenter.heroku.com/articles/procfile
[28] Heroku Buildpacks, https://devcenter.heroku.com/articles/buildpacks