# **High-Level Software Specification**
for
EvenTicket

Prepared by
Brunetti Jacopo (1856271)
Carmignani Federico (1845479)
Cicchetti Giordano (1853840)

Sapienza University of Rome
Laboratory in Advanced Programming
Software Engineering Project
Prof. Mecella

Rome, 9^{th} September 2022

Contents

Chapter 1

The idea of the project

1.1 Purpose

The purpose of our project is to build a web application providing a set of services aimed at the **management and proposals of events and distribution of tickets**.
The goal is provide to the users a way to discover events and buy or preorder tickets directly online. At the same time the application allows also the events' managers to publish their events in order to promote them and eventually to give the possibility to sell tickets of the event through the website.

1.2 Intended Audience

The website could have two types of users:

- **Events' managers** who want to promote their events and to sell tickets or just providing the possibility to ask for a reservation.
- Any **registered user** who wants to discover new events around him and have the possibility to buy a ticket.

Furthermore, the website is opened to every **web user** to obtain information about events.
The **administrators** of the website will be able to manage the different layers of the software system, from apporting modifications to the web interface till solving generic issues; they have the possibility to admit new events' managers and to monitor the evolution of the service.

1.3 Product Scope

The scope of the product is to develop a **customized Ticket Selling Service** which allows people to search for an event where they can spend a evening or just an afternoon. For example **Mecellone's Pub** decides to organize an **aperitif** for university students and promotes it using our website, people who are near the location will see the event as one of the first presented in the site or just searching for the category "aperitif" it will be easy to find it.
Therefore, our website is opened to whatever type of event: from a musical concert to a generic event of a club. Naturally, the managers of the events will be allowed to publish their events by the administrators of the application in order to be trusted by the website.

1.4 Conceptual scheme

Here the high-level conceptual scheme of the system, where the types of users are presented and their relationships and functionalities.
The admin handles the guests and the managers who are going to propose events to be published, the guest will be able to make reservations or buy tickets, the two types of purchases that are offered by the application.
The manager manages the event and heads the place where it will be offered.
Cardinality constraints are specified.
The database is decentralized and spread into different containers as proposed in the architecture of the system.

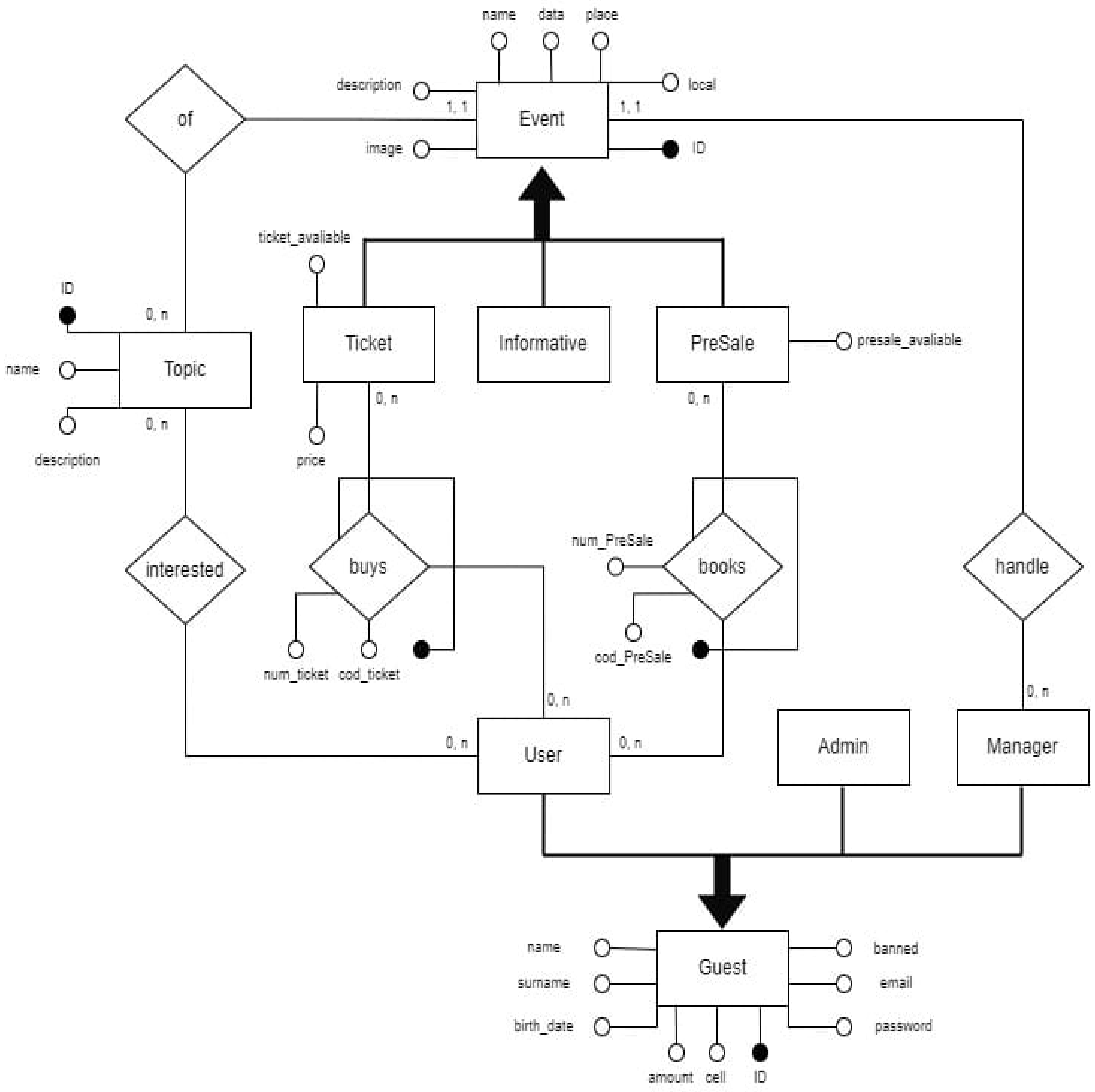


Figure 1.1: High-Level Conceptual Scheme

Chapter 2

The architectural style and functionalities

2.1 Architecture

The intention is to build a layered application based on a **microservices** architecture. Microservices architecture consists of an architectural style for developing applications. Microservices give the possibility to a large application to be separated into smaller independent parts, with each part having its own responsibility.

Our application will consists of:

- The front-end will be developed using standard technologies like HMTL, CSS, JavaScript and JQuery.
- The back-end will be divided into several microservices running into custom containers communicating with each other through **REST APIs**.
- The database system will be decentralized and spread over different microservices in order to reduce **latency** and to increase **security**.

The microservices are created as **Docker containers**, orchestrated by **Docker Compose**, to manage the different functionalities offered by the website, described in the following section.

2.2 Use cases

The application tries to semplify the main steps when buying a ticket for an event providing the following services:

- **Search of events**: the website shows automatically the events proposed by the managers.
- **Advanced search of events**: registered users can choose to search for particular types of events using filters on types and categories.
- **Ticket booking**: registered users can buy tickets for events before the event date, they have also a section where they can see their past booked events. They can also ask for pre-sale if they want to pay directly at the event place.
- **Payment service**: registered users can pay using an external payment service.
- **Notification service**: a user can show an interest towards particular categories of events and he/she will be notified when a new event of that class is published. The tickets are represented by **secret codes** to be presented to the managers of the events.

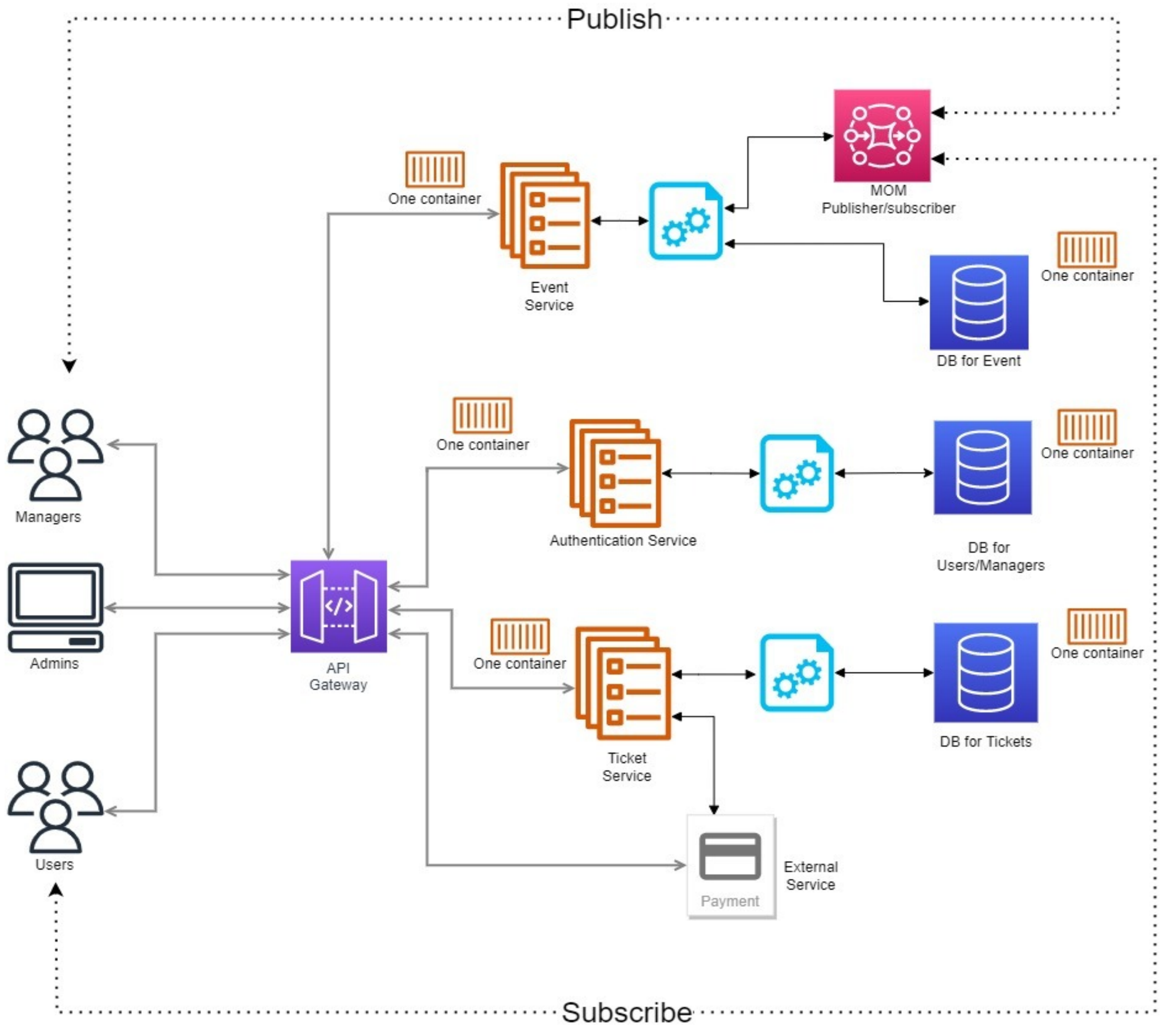


Figure 2.1: Architecture in detail

2.3 Microservices and functionalities

As shown in the figure above, the microservices of the system are:

- **Event service**: to publish, search and modify events.
- **Authentication service**: to authenticate the users.
- **Ticket service**: to sell the tickets or pre-buy them.
- **MongoDB services**: different MongoDB instances for tickets, events and users.
- **RabbitMQ services**: two containers for the producer and the consumer of the queues.

Each of them is characterized by the presence of databases like figured and an **API Gateway** is used as a dispatcher of the requests made by the users. The API Gateway is represented by an **NGINX proxy server** which is able to dispatch requests towards the other containers or just to serve static pages.

2.3.1 External services

The architecture is characterized by the use of 3 external services:

- **SendGrid**: used to send emails by the consumer of RabbitMQ to the users interested in a specific event type when an event is published.
- **Stripe**: to manage payments of tickets in the website.
- **CloudAMQP**: a cloud solution for RabbitMQ, the message-oriented middleware which implements the Advanced Message Queuing Protocol.
- **MongoDB services**: different MongoDB instances for tickets, events and users.
- **RabbitMQ services**: two containers for the producer and the consumer of the queues.

2.3.2 Message Oriented Middleware

MOM is primarily middleware that facilitates communication between distributed applications. While MOM supports both synchronous and asynchronous messaging, it is most closely identified with asynchronous messaging using queuing. MOM sends messages from one application part to another using a queue as an interim step. Client messages are sent to a queue and remain there until they are retrieved by the server application. The system is characterized by a **MOM**, it is used to handle the **notification service**: users are notified in order to receive information about interesting events.
This is done through a classic **Publisher-Subscriber** mechanism: where the manager of the events can publish their events, whereas the users are notified about events of a particular category they like. The notifications are triggered by the consumer of RabbitMQ queues, using SendGrid to send emails to the users.
For example, if I like football events, when a new match is published, I will be notified by email by the application.

2.4 The steps

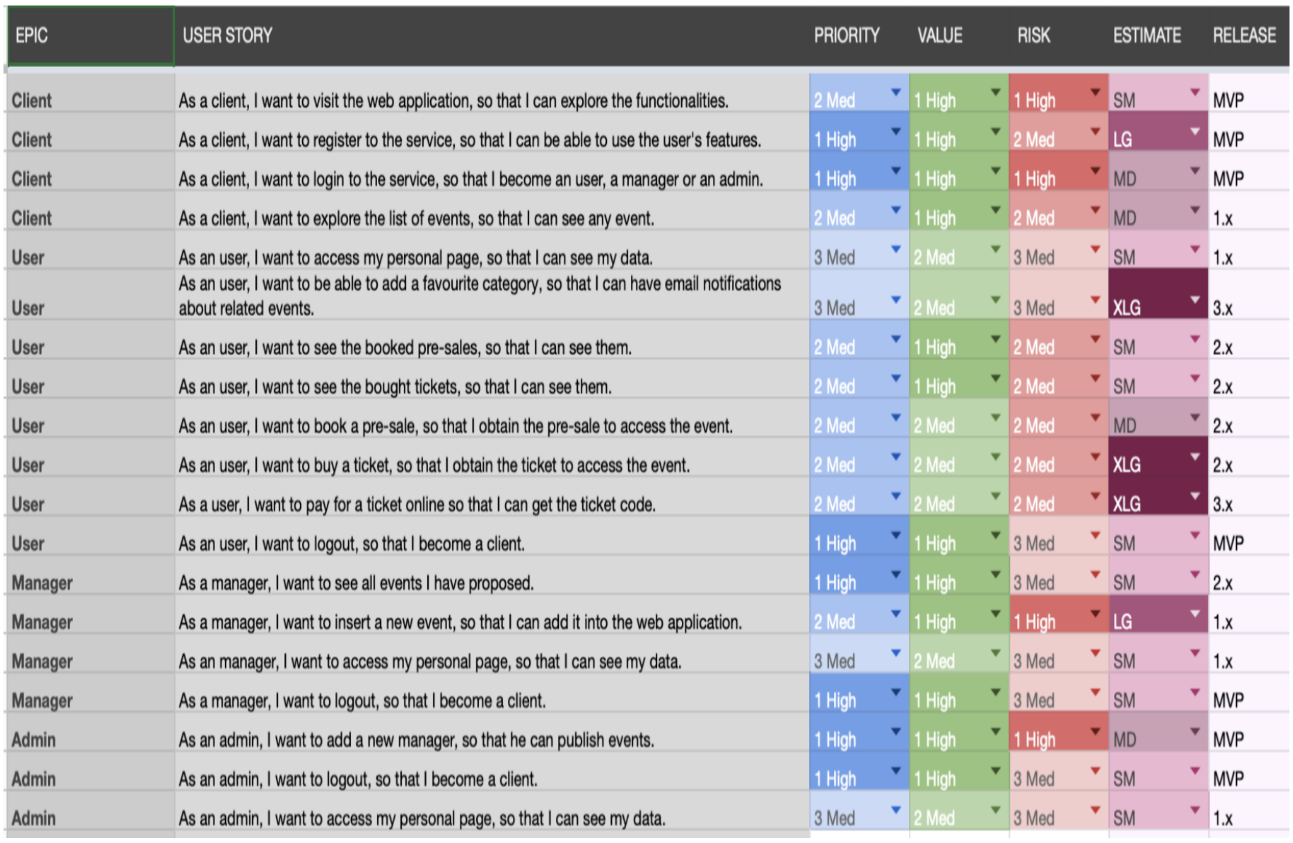
The idea is to make the project usable beyond the scope of the exam. The software development will follow the rules of generic software engineered projects: from preparing the requirements using user stories and mockups till the tracking and evaluation of the implementation process. The last step will be testing the application and the distribution of the website.

Chapter 3

Requirements

3.0.1 User Stories

In software development, user stories represent an informal and natural language description of the functionalities, better known as "features", of a software system. They are produced from the perspective of an end user, the various "stakeholders" of the application, and draw a line between the developers and the clients of the project. The "Agile" development method, pursued in the development of the EvenTicket project, provides for an initial phase of proposals and agreements between clients and developers which, in the case of the project in question, led to the drafting of the following user stories:



EPIC	USER STORY	PRIORITY	VALUE	RISK	ESTIMATE	RELEASE
Client	As a client, I want to visit the web application, so that I can explore the functionalities.	2 Med	1 High	1 High	SM	MVP
Client	As a client, I want to register to the service, so that I can be able to use the user's features.	1 High	1 High	2 Med	LG	MVP
Client	As a client, I want to login to the service, so that I become an user, a manager or an admin.	1 High	1 High	1 High	MD	MVP
Client	As a client, I want to explore the list of events, so that I can see any event.	2 Med	1 High	2 Med	MD	1.x
User	As an user, I want to access my personal page, so that I can see my data.	3 Med	2 Med	3 Med	SM	1.x
User	As an user, I want to be able to add a favourite category, so that I can have email notifications about related events.	3 Med	2 Med	3 Med	XLG	3.x
User	As an user, I want to see the booked pre-sales, so that I can see them.	2 Med	1 High	2 Med	SM	2.x
User	As an user, I want to see the bought tickets, so that I can see them.	2 Med	1 High	2 Med	SM	2.x
User	As an user, I want to book a pre-sale, so that I obtain the pre-sale to access the event.	2 Med	2 Med	2 Med	MD	2.x
User	As an user, I want to buy a ticket, so that I obtain the ticket to access the event.	2 Med	2 Med	2 Med	XLG	2.x
User	As a user, I want to pay for a ticket online so that I can get the ticket code.	2 Med	2 Med	2 Med	XLG	3.x
User	As an user, I want to logout, so that I become a client.	1 High	1 High	3 Med	SM	MVP
Manager	As a manager, I want to see all events I have proposed.	1 High	1 High	3 Med	SM	2.x
Manager	As a manager, I want to insert a new event, so that I can add it into the web application.	2 Med	1 High	1 High	LG	1.x
Manager	As an manager, I want to access my personal page, so that I can see my data.	3 Med	2 Med	3 Med	SM	1.x
Manager	As a manager, I want to logout, so that I become a client.	1 High	1 High	3 Med	SM	MVP
Admin	As an admin, I want to add a new manager, so that he can publish events.	1 High	1 High	1 High	MD	MVP
Admin	As an admin, I want to logout, so that I become a client.	1 High	1 High	3 Med	SM	MVP
Admin	As an admin, I want to access my personal page, so that I can see my data.	3 Med	2 Med	3 Med	SM	1.x

Figure 3.1: User Stories

3.0.2 LoFi Mockups

Low-fidelity mockups represent initial layouts of web pages and applications, also known as wireframes. They help focus on the key purpose and functionality of each page by deliberately excluding any specific details such as colors, fonts, logo and exact size, which can be added and changed in development. Wireframes use basic shapes, image placeholders, and generic text to represent, in grayscale, the draft layout for future design. Names and elements have a purely indicative and illustrative purpose.
Below are the main low-fidelity mockups for EvenTicket developed through the Balsamiq framework:

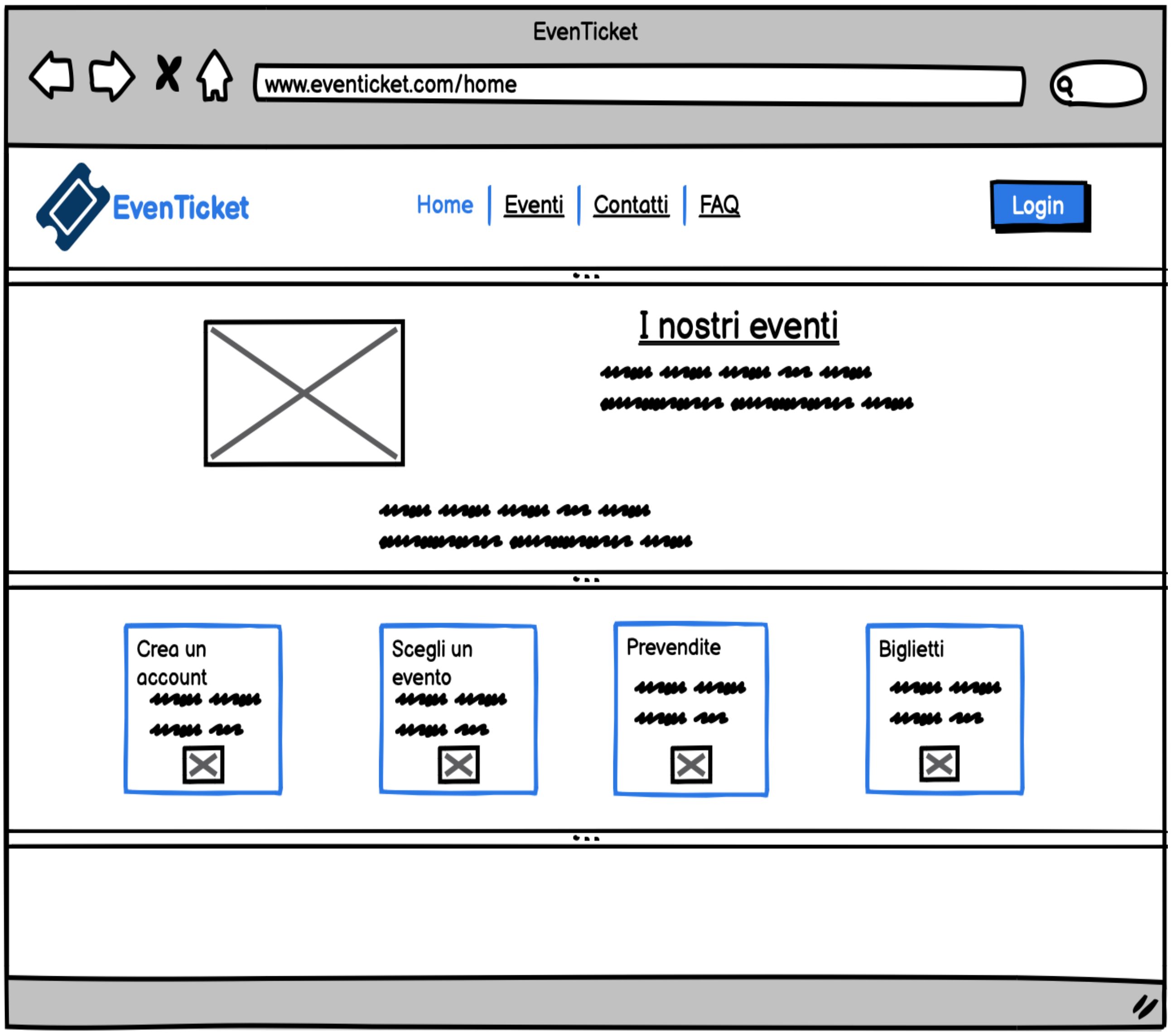


Figure 3.2: Home page

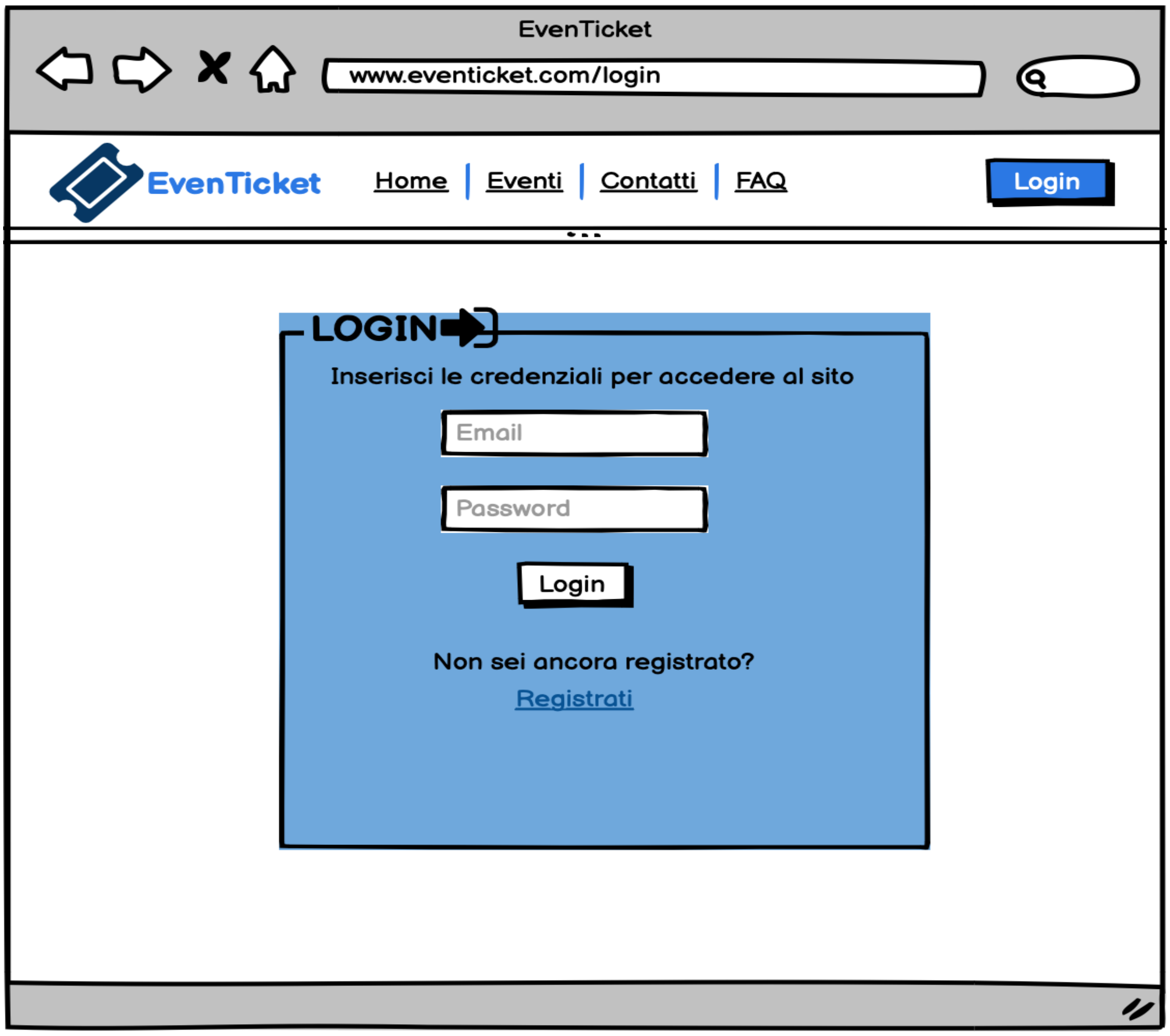

EvenTicket
www.eventicket.com/login
EvenTicket
Home | Eventi | Contatti | FAQ
Login
LOGIN
Inserisci le credenziali per accedere al sito
Email
Password
Login
Non sei ancora registrato?
Registrati

Figure 3.3: Login page

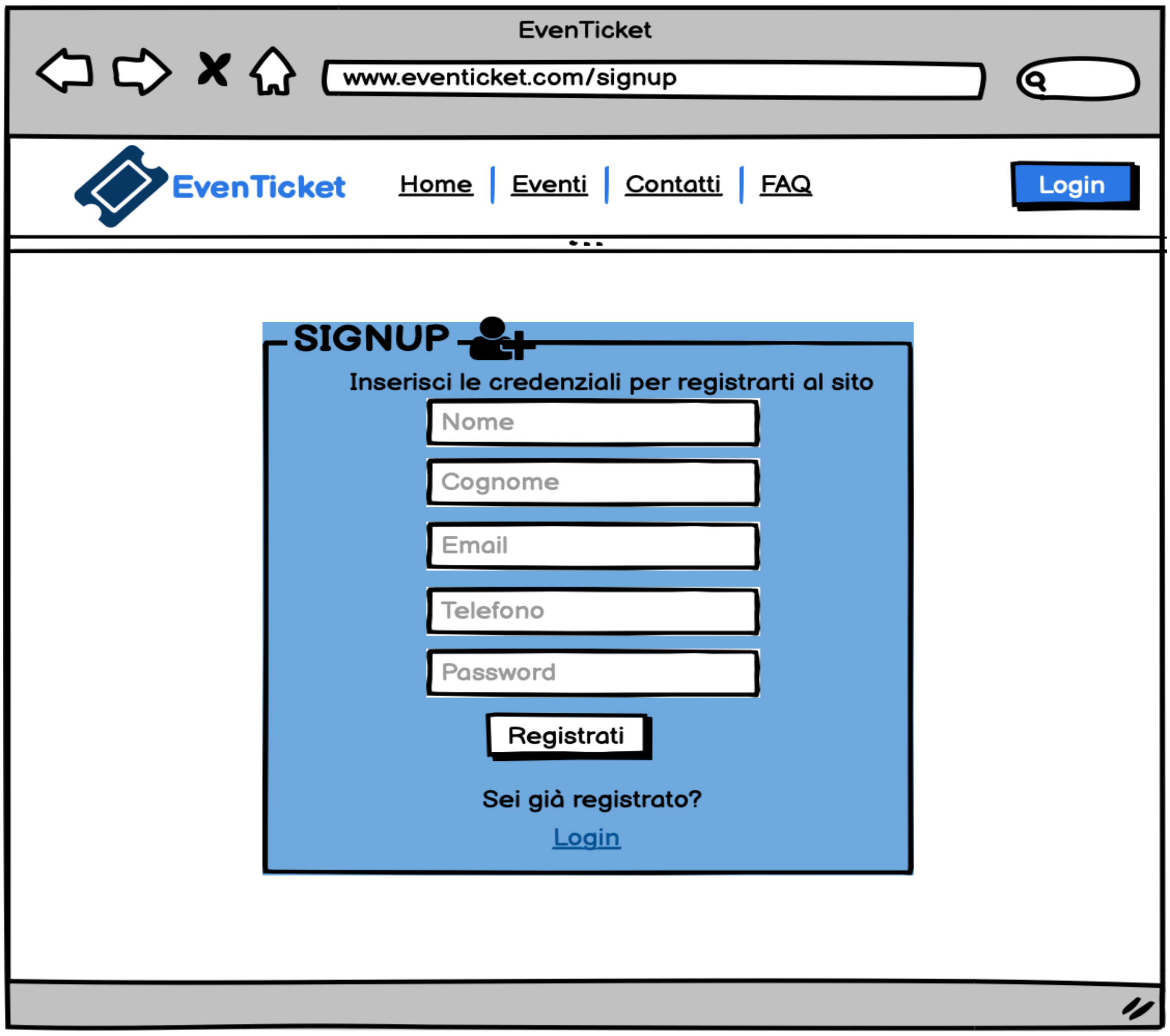


Figure 3.4: Signup page

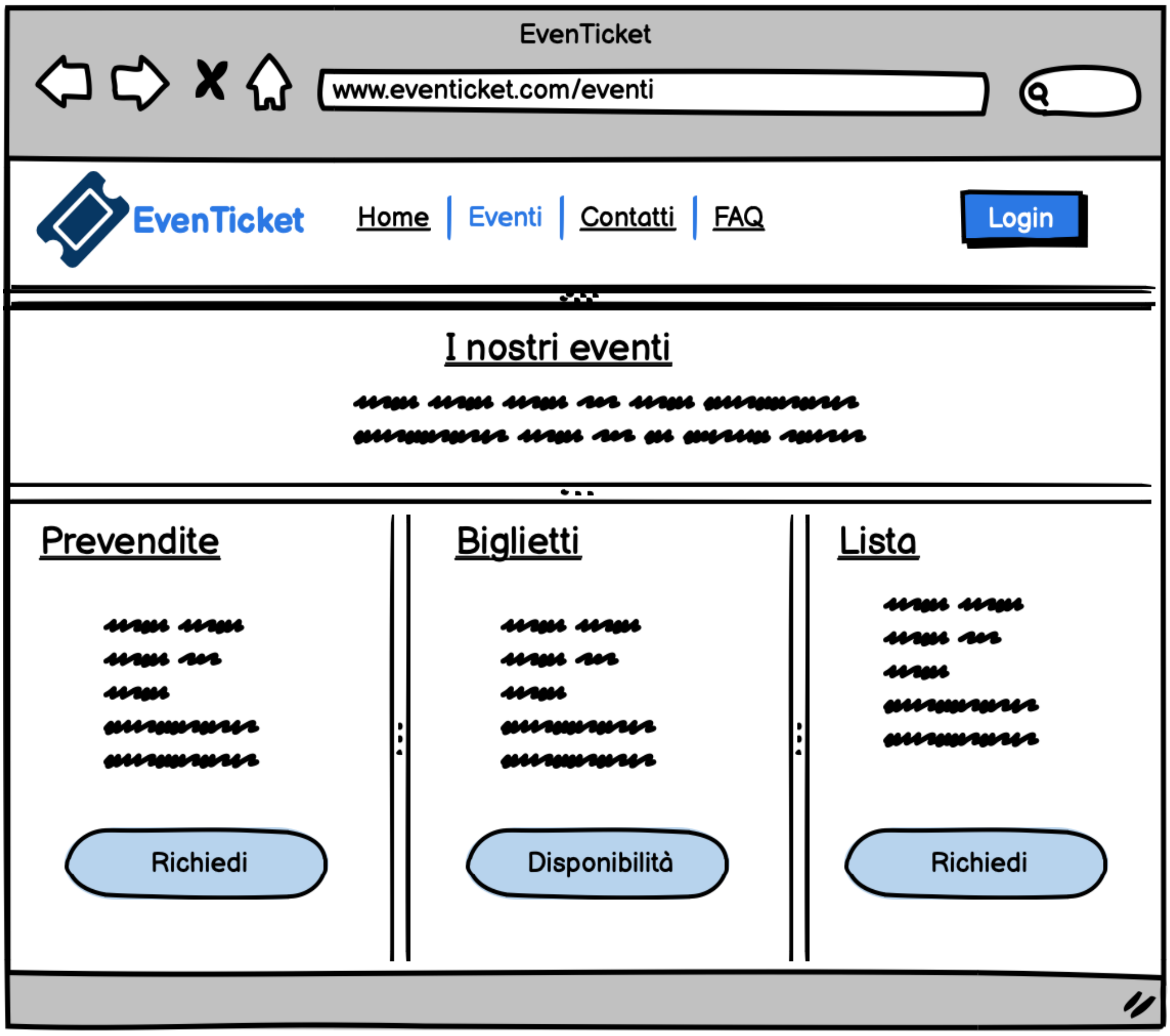


Figure 3.5: Eventi page

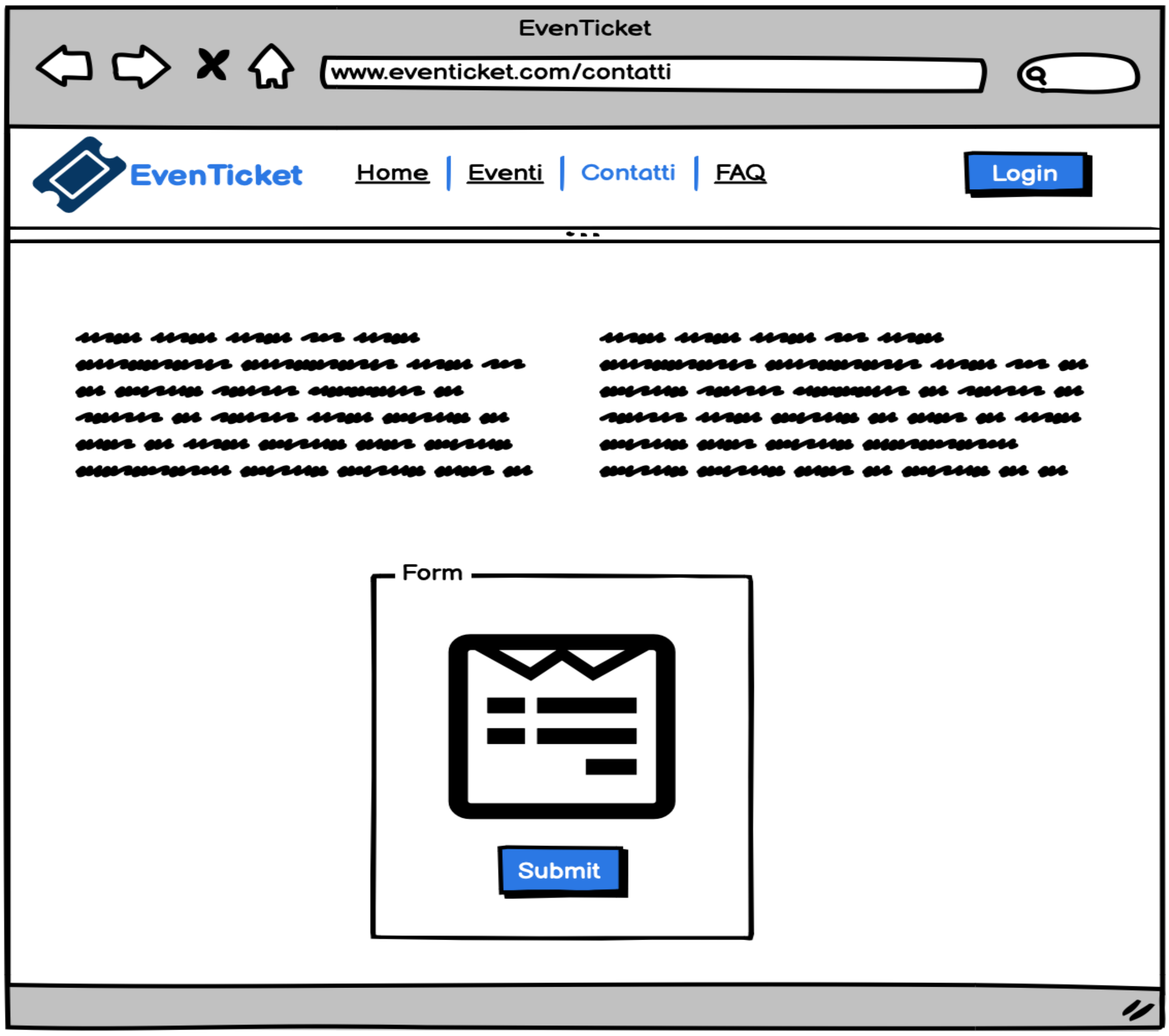


Figure 3.6: Contatti page

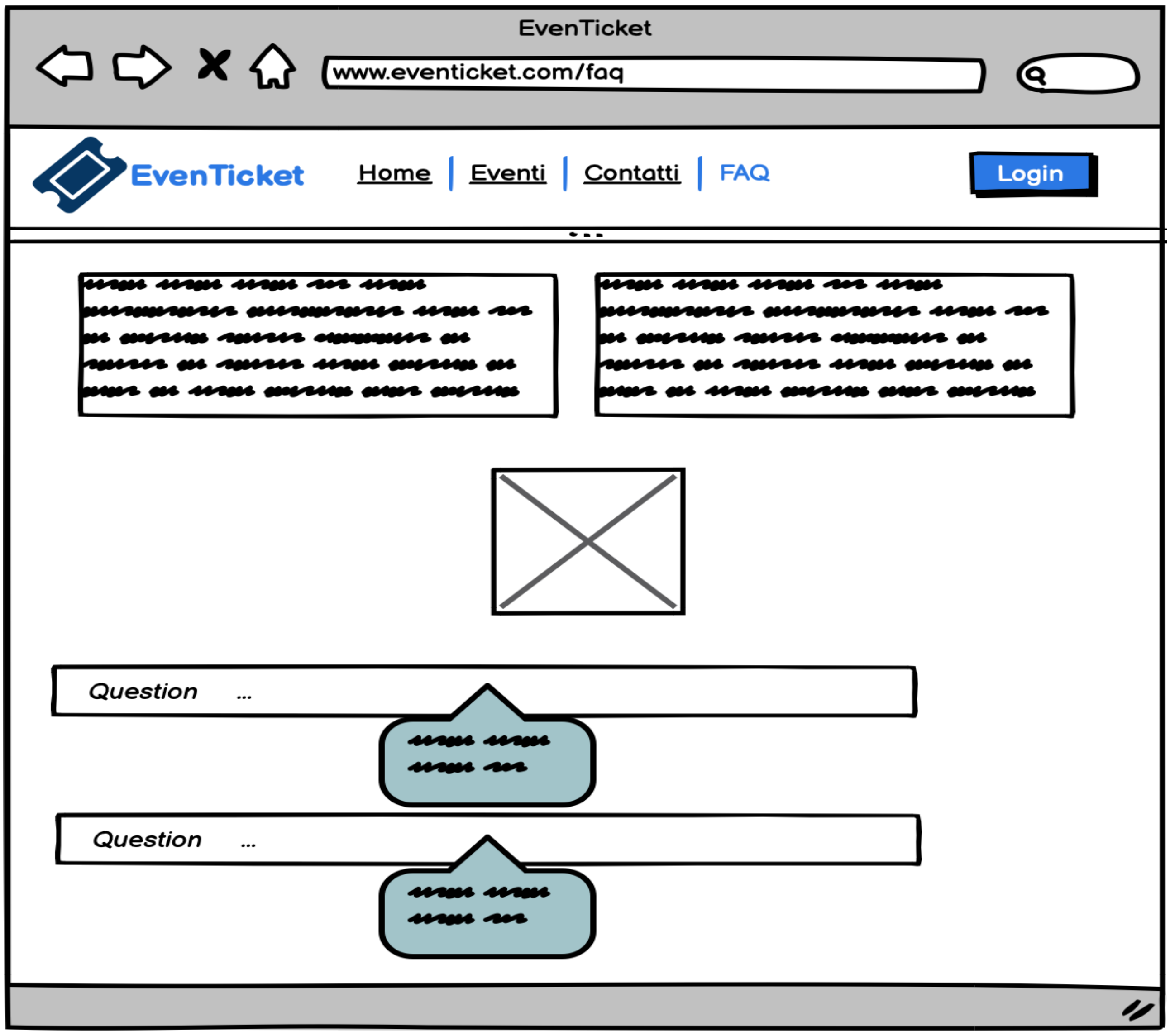
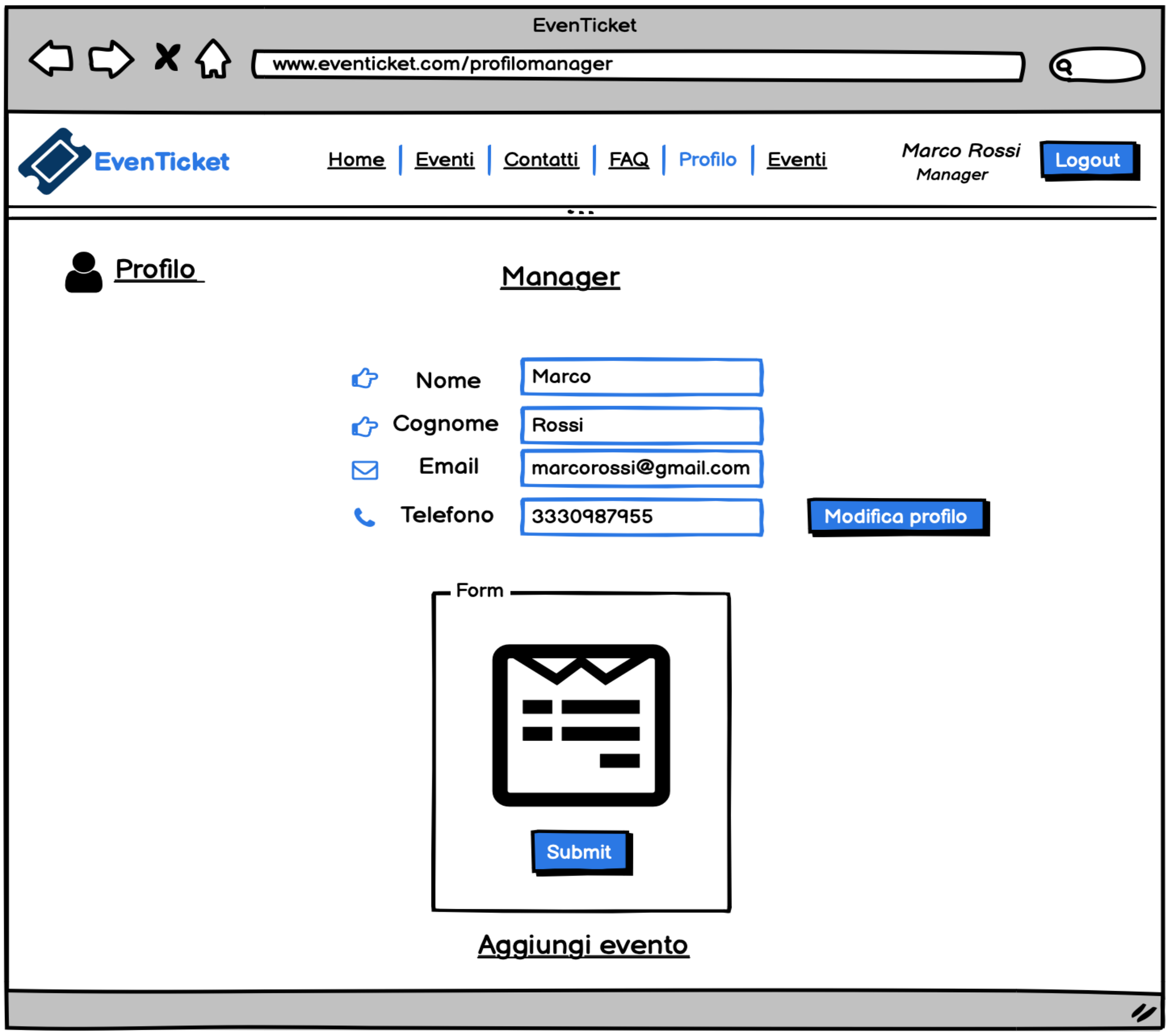
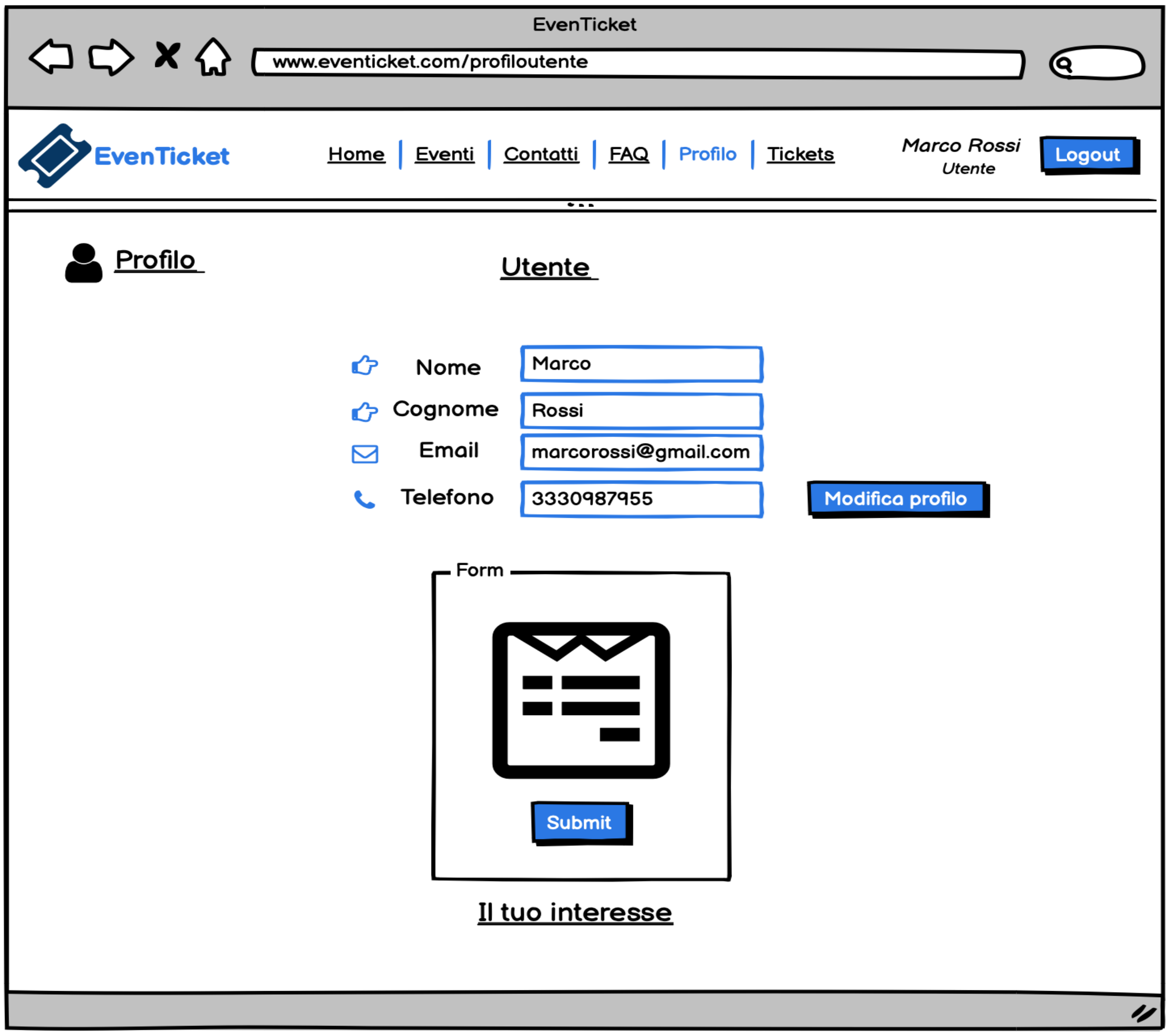


Figure 3.7: Faq page



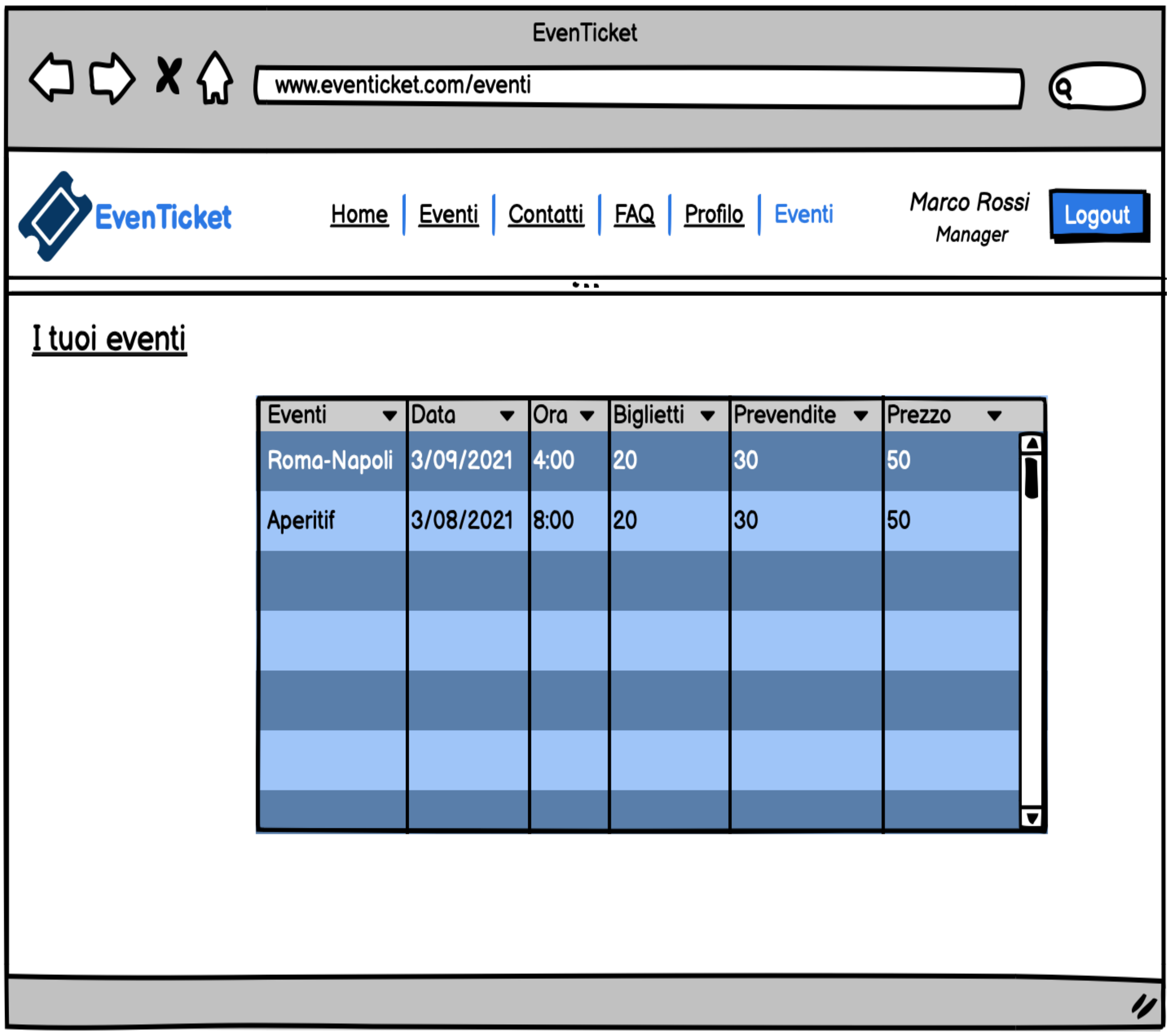
Nome	Marco
Cognome	Rossi
Email	marcorossi@gmail.com
Telefono	3330987955

Figure 3.8: Profilo Manager page



Nome	Marco
Cognome	Rossi
Email	marcorossi@gmail.com
Telefono	3330987955

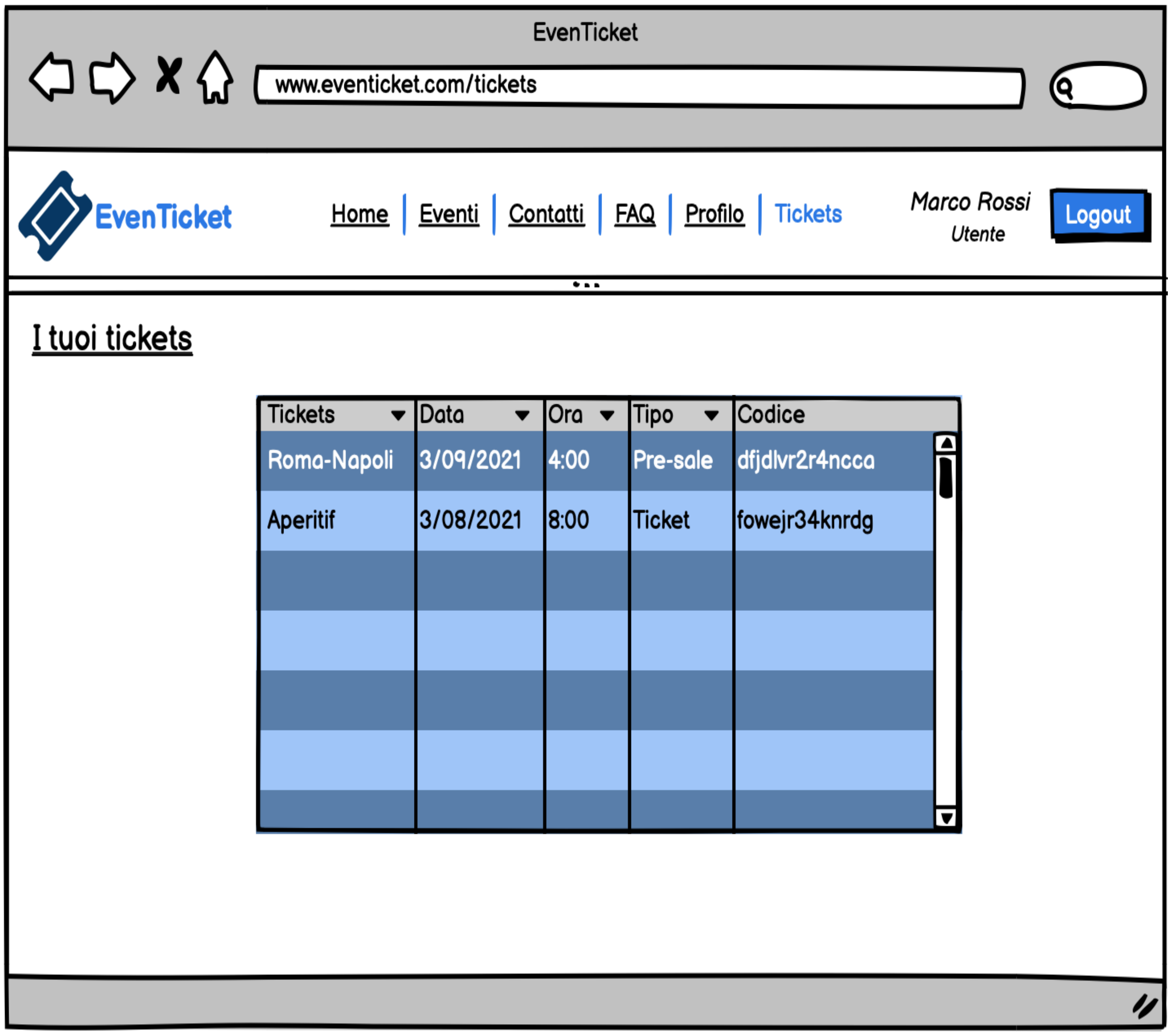
Figure 3.9: Profilo Utente page



I tuoi eventi

Eventi	Data	Ora	Biglietti	Prevendite	Prezzo
Roma-Napoli	3/09/2021	4:00	20	30	50
Aperitif	3/08/2021	8:00	20	30	50

Figure 3.10: Eventi Manager page

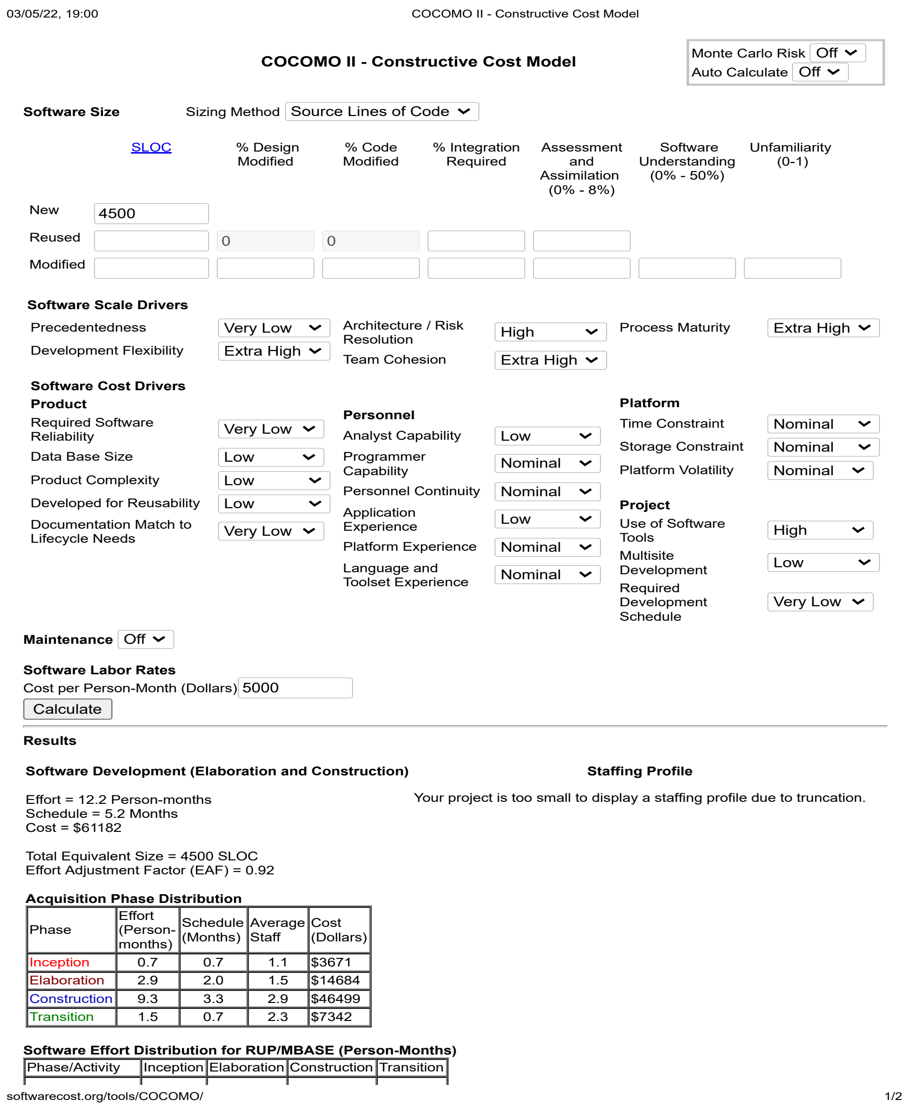


Tickets	Data	Ora	Tipo	Codice
Roma-Napoli	3/09/2021	4:00	Pre-sale	dfjdlvr2r4ncca
Aperitif	3/08/2021	8:00	Ticket	fowejr34knrdg

Figure 3.11: Tickets page

Chapter 4

Complexity of the software development



03/05/22, 19:00 COCOMO II - Constructive Cost Model

COCOMO II - Constructive Cost Model

Monte Carlo Risk: Off
Auto Calculate: Off

Software Size Sizing Method: Source Lines of Code

	SLOC	% Design Modified	% Code Modified	% Integration Required	Assessment and Assimilation (0% - 8%)	Software Understanding (0% - 50%)	Unfamiliarity (0-1)
New	4500						
Reused		0	0				
Modified							

Software Scale Drivers

Precedentedness	Very Low	Architecture / Risk Resolution	High	Process Maturity	Extra High
Development Flexibility	Extra High	Team Cohesion	Extra High		

Software Cost Drivers

Product

Required Software Reliability	Very Low
Data Base Size	Low
Product Complexity	Low
Developed for Reusability	Low
Documentation Match to Lifecycle Needs	Very Low

Personnel

Analyst Capability	Low
Programmer Capability	Nominal
Personnel Continuity	Nominal
Application Experience	Low
Platform Experience	Nominal
Language and Toolset Experience	Nominal

Platform

Time Constraint	Nominal
Storage Constraint	Nominal
Platform Volatility	Nominal

Project

Use of Software Tools	High
Multisite Development	Low
Required Development Schedule	Very Low

Maintenance Off

Software Labor Rates
Cost per Person-Month (Dollars) 5000
Calculate

Results

Software Development (Elaboration and Construction)

Effort = 12.2 Person-months
Schedule = 5.2 Months
Cost = $61182

Total Equivalent Size = 4500 SLOC
Effort Adjustment Factor (EAF) = 0.92

Staffing Profile

Your project is too small to display a staffing profile due to truncation.

Acquisition Phase Distribution

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	0.7	0.7	1.1	$3671
Elaboration	2.9	2.0	1.5	$14684
Construction	9.3	3.3	2.9	$46499
Transition	1.5	0.7	2.3	$7342

Software Effort Distribution for RUP/MBASE (Person-Months)

Phase/Activity	Inception	Elaboration	Construction	Transition

softwarecost.org/tools/COCOMO/ 1/2

Figure 4.1: COCOMO evaluation

Function Point Analysis is the engineering discipline for measuring the functional size of software.

Function Points are an ISO -standard for software size. The COCOMO II software estimation model is a set of equations that take a Function Point (size) count as the primary input. COCOMO II is a parametric estimation model - a set of equations that were conceived from studies of projects that take into account the diseconomies of scale associated with software development. These equations can take function point size as the primary input to help you estimate effort and schedule for a software project.

4.0.1 Function Points and COCOMO II method

FPA provides a standardized method to functionally size the software work product. This work product is the output of software new development and improvement projects for subsequent releases. It is the software that is relocated to the production application at project implementation. It measures functionality from the user's point of view i.e. on the basis of what the user requests and receives in return.
Function Point Analysis (FPA) is a method or set of rules of Functional Size Measurement. It assesses the functionality delivered to its users, based on the user's external view of the functional requirements.
It measures the logical view of an application, not the physically implemented view or the internal technical view.
The Function Point Analysis technique is used to analyze the functionality delivered by software and Unadjusted Function Point (UFP) is the unit of measurement.

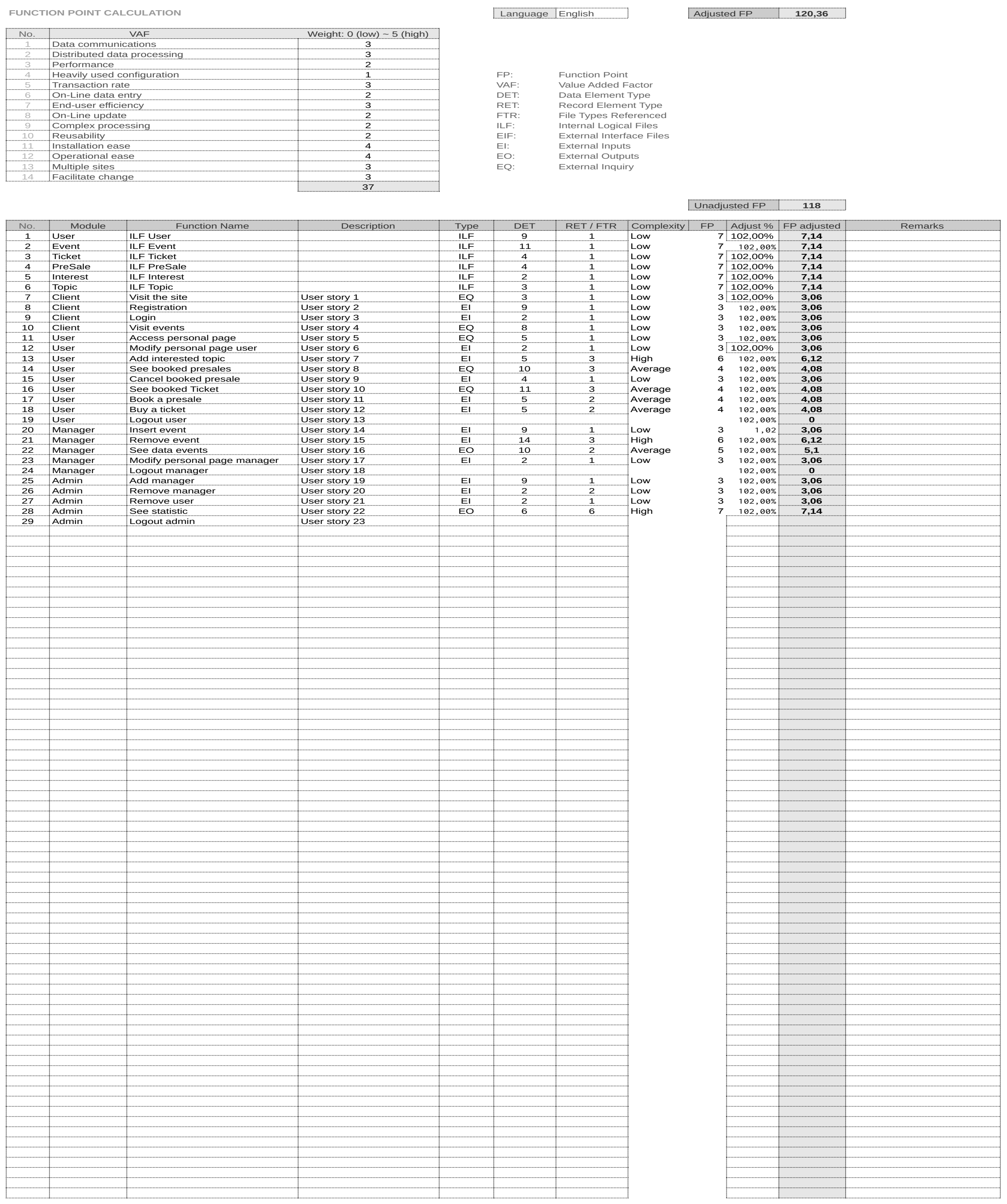
Types of FPA:

- Transactional Functional Type – External Input (EI): EI processes data or control information that comes from outside the application's boundary. The EI is an elementary process.
- Transactional Functional Type – External Input (EI): EI processes data or control information that comes from outside the application's boundary. The EI is an elementary process.
- External Output (EO): EO is an elementary process that generates data or control information sent outside the application's boundary.
- External Inquiries (EQ): EQ is an elementary process made up of an input-output combination that results in data retrieval.
- Data Functional Type – Internal Logical File (ILF): A user identifiable group of logically related data or control information maintained within the boundary of the application.
- External Interface File (EIF): A group of users recognizable logically related data allusion to the software but maintained within the boundary of another software.

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e number of Lines of Code.
It is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality. It was proposed by Barry Boehm in 1981 and is based on the study of 63 projects, which makes it one of the best-documented models. The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort and Schedule:

- Effort: Amount of labor that will be required to complete a task. It is measured in person-months units.
- Schedule: Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, months.

The estimation in terms of FP and COCOMO II is presented in the spreadheets above and in the one below. It is possible to estimate the LOCs around 5KLOCs considering an average of 50 LOCs/FP using modern programming languages like Java or Python. These lines are obviously including even the source code of the libraries that are going to be used in the project.



FUNCTION POINT CALCULATION

Language	English
Adjusted FP	120,36

No.	VAF	Weight: 0 (low) ~ 5 (high)
1	Data communications	3
2	Distributed data processing	3
3	Performance	2
4	Heavily used configuration	1
5	Transaction rate	3
6	On-Line data entry	2
7	End-user efficiency	3
8	On-Line update	2
9	Complex processing	2
10	Reusability	2
11	Installation ease	4
12	Operational ease	4
13	Multiple sites	3
14	Facilitate change	3
		37

FP:	Function Point
VAF:	Value Added Factor
DET:	Data Element Type
RET:	Record Element Type
FTR:	File Types Referenced
ILF:	Internal Logical Files
EIF:	External Interface Files
EI:	External Inputs
EO:	External Outputs
EQ:	External Inquiry

Unadjusted FP	118

No.	Module	Function Name	Description	Type	DET	RET / FTR	Complexity	FP	Adjust %	FP adjusted	Remarks
1	User	ILF User		ILF	9	1	Low	7	102,00%	7,14	
2	Event	ILF Event		ILF	11	1	Low	7	102,00%	7,14	
3	Ticket	ILF Ticket		ILF	4	1	Low	7	102,00%	7,14	
4	PreSale	ILF PreSale		ILF	4	1	Low	7	102,00%	7,14	
5	Interest	ILF Interest		ILF	2	1	Low	7	102,00%	7,14	
6	Topic	ILF Topic		ILF	3	1	Low	7	102,00%	7,14	
7	Client	Visit the site	User story 1	EQ	3	1	Low	3	102,00%	3,06	
8	Client	Registration	User story 2	EI	9	1	Low	3	102,00%	3,06	
9	Client	Login	User story 3	EI	2	1	Low	3	102,00%	3,06	
10	Client	Visit events	User story 4	EQ	8	1	Low	3	102,00%	3,06	
11	User	Access personal page	User story 5	EQ	5	1	Low	3	102,00%	3,06	
12	User	Modify personal page user	User story 6	EI	2	1	Low	3	102,00%	3,06	
13	User	Add interested topic	User story 7	EI	5	3	High	6	102,00%	6,12	
14	User	See booked presales	User story 8	EQ	10	3	Average	4	102,00%	4,08	
15	User	Cancel booked presale	User story 9	EI	4	1	Low	3	102,00%	3,06	
16	User	See booked Ticket	User story 10	EQ	11	3	Average	4	102,00%	4,08	
17	User	Book a presale	User story 11	EI	5	2	Average	4	102,00%	4,08	
18	User	Buy a ticket	User story 12	EI	5	2	Average	4	102,00%	4,08	
19	User	Logout user	User story 13						102,00%	0	
20	Manager	Insert event	User story 14	EI	9	1	Low	3	1,02	3,06	
21	Manager	Remove event	User story 15	EI	14	3	High	6	102,00%	6,12	
22	Manager	See data events	User story 16	EO	10	2	Average	5	102,00%	5,1	
23	Manager	Modify personal page manager	User story 17	EI	2	1	Low	3	102,00%	3,06	
24	Manager	Logout manager	User story 18						102,00%	0	
25	Admin	Add manager	User story 19	EI	9	1	Low	3	102,00%	3,06	
26	Admin	Remove manager	User story 20	EI	2	2	Low	3	102,00%	3,06	
27	Admin	Remove user	User story 21	EI	2	1	Low	3	102,00%	3,06	
28	Admin	See statistic	User story 22	EO	6	6	High	7	102,00%	7,14	
29	Admin	Logout admin	User story 23								

Figure 4.2: FP calculation

Chapter 5

Development process based on SCRUM

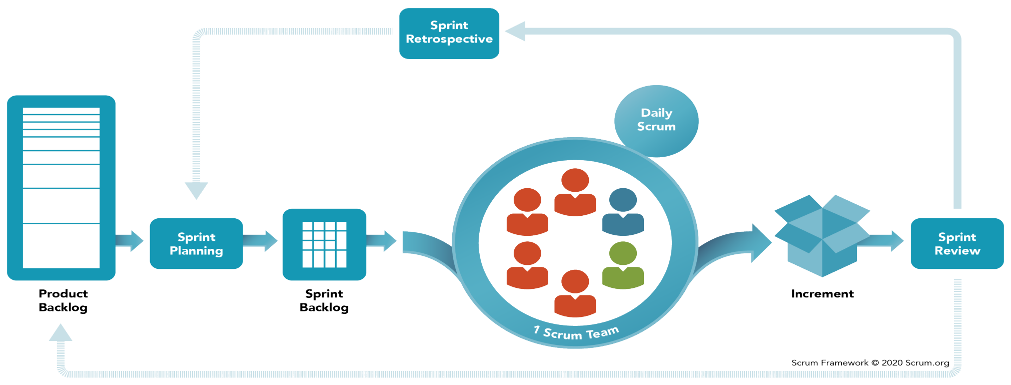


Figure 5.1: SCRUM process

5.0.1 An overview

Scrum is an Agile framework for project management that emphasizes teamwork, accountability and iterative progress toward a well-defined goal. The framework begins with a simple premise: Start with what can be seen or known. After that, track the progress and tweak, as necessary. Scrum is often part of Agile software development. It is named for a rugby formation in which everyone plays a role. Software development Scrum roles include the following:

Product owner. This person serves as the liaison between the development team and its customers. The product owner is responsible for ensuring that expectations for the completed product are communicated and agreed upon.
Scrum Master. The Scrum Master is referred to as the project facilitator. They ensure Scrum best practices are followed. They must be good leaders and project managers, skilled at collaboration, conflict resolution and process improvement.
Development Team. Members of the Scrum development team work together to create and test incremental releases of the final product. Developers must know Scrum and Agile development practices.

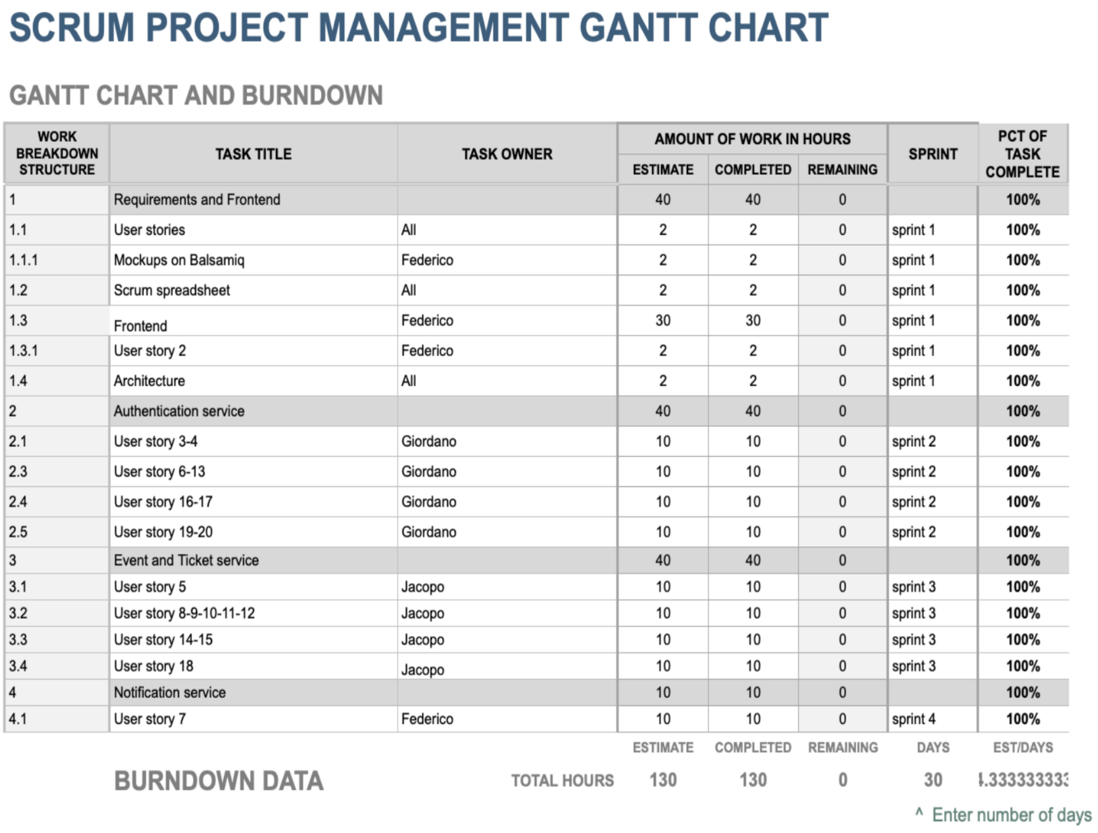
The Scrum process encourages practitioners to work with what they have and continually evaluate what is or is not working. Good communication is essential and is carried out through meetings, called "events." Scrum events include the following:

Daily Scrum. This event is a short, stand-up daily meeting that takes place in the same place and time each day. In these meetings, the team reviews work accomplished the previous day and plans what will be done in the next 24 hours. This is the time when team members discuss problems that might prevent project completion.
Sprint. A Sprint is the time frame in which work must be completed – often 30 days. New Sprints start right after the end of the previous one.
Sprint Planning Meeting. In these meetings, everyone participates in setting goals. At the end, at least one increment – a usable piece of software – should be produced.
Sprint Review. This is the time to show off the increment.
Sprint Retrospective. A Sprint Retrospective is a meeting held after a Sprint ends. During this meeting, everyone reflects on the process. A team-building exercise may also be offered. An important goal of this event is continuous improvement.

An artifact is something of historical interest that warrants being reexamined. In Scrum product development, artifacts are used to see what has been done and what is still in the queue. It is useful to look at Scrum artifacts in Sprint Planning Meetings. Scrum artifacts include the following:
Product backlog. This refers to what remains to be done. During a product backlog grooming session, the development team works with the business owner to prioritize work that has been backlogged. The product backlog may be fine-tuned during a process called backlog refinement.
Sprint backlog. This is a list of tasks that must be completed before selected product backlog items can be delivered. These are divided into time-based user stories.

5.0.2 The spreadsheet

These are the sprints and the product backlog of the project documented in a SCRUM spreadsheet:



SCRUM PROJECT MANAGEMENT GANTT CHART

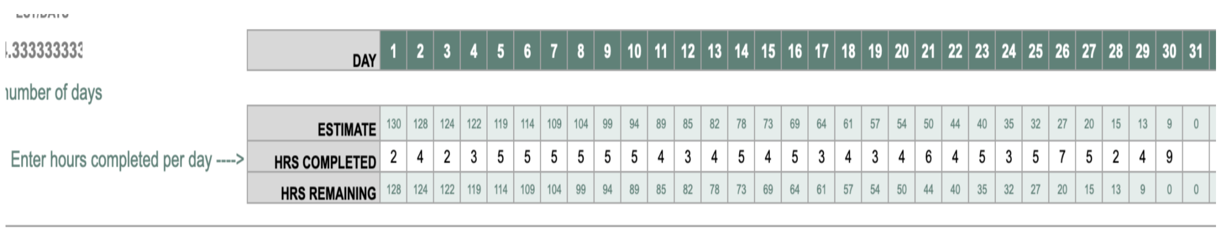
GANTT CHART AND BURNDOWN

WORK BREAKDOWN STRUCTURE	TASK TITLE	TASK OWNER	AMOUNT OF WORK IN HOURS			SPRINT	PCT OF TASK COMPLETE
			ESTIMATE	COMPLETED	REMAINING		
1	Requirements and Frontend		40	40	0		100%
1.1	User stories	All	2	2	0	sprint 1	100%
1.1.1	Mockups on Balsamiq	Federico	2	2	0	sprint 1	100%
1.2	Scrum spreadsheet	All	2	2	0	sprint 1	100%
1.3	Frontend	Federico	30	30	0	sprint 1	100%
1.3.1	User story 2	Federico	2	2	0	sprint 1	100%
1.4	Architecture	All	2	2	0	sprint 1	100%
2	Authentication service		40	40	0		100%
2.1	User story 3-4	Giordano	10	10	0	sprint 2	100%
2.3	User story 6-13	Giordano	10	10	0	sprint 2	100%
2.4	User story 16-17	Giordano	10	10	0	sprint 2	100%
2.5	User story 19-20	Giordano	10	10	0	sprint 2	100%
3	Event and Ticket service		40	40	0		100%
3.1	User story 5	Jacopo	10	10	0	sprint 3	100%
3.2	User story 8-9-10-11-12	Jacopo	10	10	0	sprint 3	100%
3.3	User story 14-15	Jacopo	10	10	0	sprint 3	100%
3.4	User story 18	Jacopo	10	10	0	sprint 3	100%
4	Notification service		10	10	0		100%
4.1	User story 7	Federico	10	10	0	sprint 4	100%

		ESTIMATE	COMPLETED	REMAINING	DAYS	EST/DAYS
BURNDOWN DATA	TOTAL HOURS	130	130	0	30	4.333333333

^ Enter number of days

Figure 5.2: Product backlog and Sprints in SCRUM



.333333333

umber of days

Enter hours completed per day ---->

DAY	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ESTIMATE	130	128	124	122	119	114	109	104	99	94	89	85	82	78	73	69	64	61	57	54	50	44	40	35	32	27	20	15	13	9	0
HRS COMPLETED	2	4	2	3	5	5	5	5	5	5	4	3	4	5	4	5	3	4	3	4	6	4	5	3	5	7	5	2	4	9	
HRS REMAINING	128	124	122	119	114	109	104	99	94	89	85	82	78	73	69	64	61	57	54	50	44	40	35	32	27	20	15	13	9	0	0

Figure 5.3: Planning in SCRUM

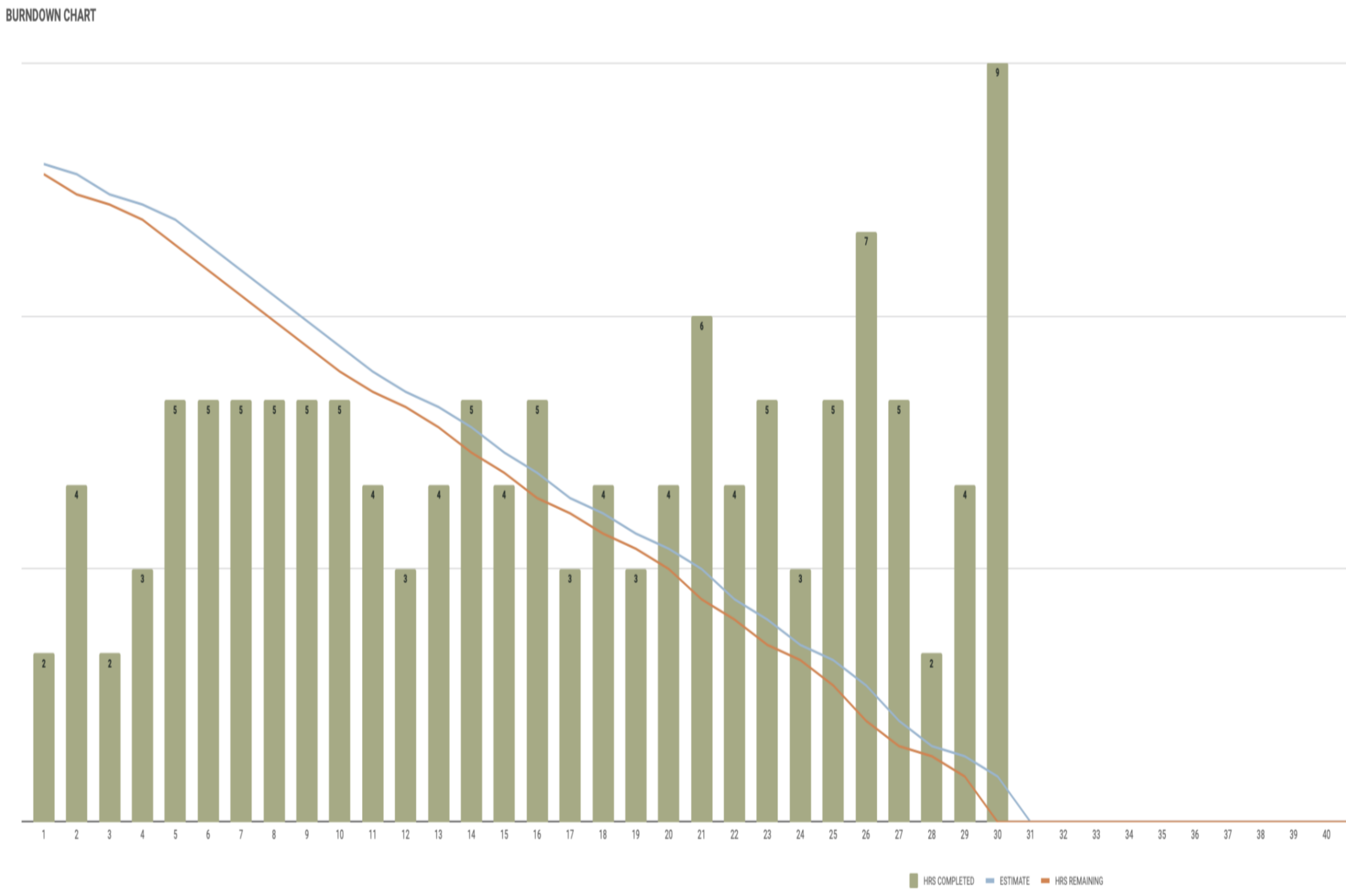


Figure 5.4: Burndown Chart in SCRUM

Chapter 6

Technologies and distribution

6.0.1 Technologies

The containers are created using **Docker** and they are orchestrated using **Docker Compose**.
The functional containers are developed in different languages: **Python** and **NodeJS**.
The containers for the databases are standard **MongoDB** containers.
The container for the API Gateway is an **NGINX** container.

External services are used as **SendGrid** for email notifications, **Stripe** for payments and **CloudAMQP** for the **RabbitMQ** queues handling the notification service.

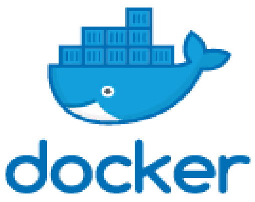


Figure 6.1: Docker logo

6.0.2 Distribution

The steps to follow to build and deploy the system on whatever platform (either on-premise or on cloud) using a IAC approach are the following ones:

- Install Docker Desktop on your device.
- Download the zip of the project from GitHub at **https://github.com/Giordano-Cicchetti/Event_Ticket**.
- Unzip the project and open a terminal inside it.
- Execute **docker compose up –build**.
- From a Browser digit **localhost:80** to access the website.

The **GitHub** link contains all the source code, configuration files, docker compose files and dockerfiles.