



SAPIENZA
UNIVERSITÀ DI ROMA

HiSchool: sviluppo di un'applicazione per il supporto didattico per la scuola secondaria

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di Laurea in Ingegneria Informatica e Automatica

Candidato
Silvia del Piano
Matricola 1759992

Relatore
Prof. Roberto Beraldì

Anno Accademico 2018/2019

Tesi non ancora discussa

HiSchool: sviluppo di un'applicazione per il supporto didattico per la scuola secondaria

Tesi di Laurea. Sapienza – Università di Roma

© 2019 Silvia del Piano. Tutti i diritti riservati

Questa tesi è stata composta con L^AT_EX e la classe Sapthesis.

Versione: today

Email dell'autore: delpiano.1759992@studenti.uniroma1.it

Da vedere

Indice

1 Raccolta dei requisiti	1
1.1 Descrizione della realtà di interesse	1
1.2 Dati gestiti dal sistema e principali funzionalità	2
1.2.1 MockUp e User Stories	2
1.2.2 Lo-Fi MockUp	3
1.2.3 User Stories	6
2 Tecnologie e Metodologie utilizzate	11
2.1 Rails e il paradigma Model-View-Controller	11
2.1.1 Autenticazione locale	11
2.1.2 Autenticazione tramite Google OAuth	12
2.1.3 Google Drive	13
2.1.4 Altre gemme	13
2.2 Agile per lo sviluppo software	15
2.2.1 Tool per applicare al meglio Agile	15
3 Analisi Concettuale del Sistema	17
3.1 Schema Entità-Relazione	17
3.2 Elenco delle entità e delle relazioni	18
3.2.1 Generalizzazione User	18
3.2.2 Entità School e Class	18
3.2.3 Entità Post	19
4 Progettazione del sistema	20
4.1 Archittura del sistema	20
4.1.1 Architettura dei Controller	20
4.1.2 Architettura dei Model	21
4.1.3 Architettura delle View	22
4.2 Progettazione logica del Data Layer	22

Capitolo 1

Raccolta dei requisiti

1.1 Descrizione della realtà di interesse

Un giorno parlando con i miei colleghi universitari, si è accennato alla comodità offerta dai siti relativi ai singoli corsi di studio. Essi infatti sono un porto sicuro per ogni studente iscritto, frequentante e non: è possibile trovarvi il materiale su cui studiare, informazioni e contatti dei professori a cui sono affidate le lezioni, loro comunicazioni e spesso molto altro. Tutti sanno che per qualsiasi dubbio possa sorgere a proposito di uno specifico corso il primo posto dove andare a guardare è il sito ad esso dedicato. Tutto ciò mi ha invogliato a realizzare qualcosa di simile per la scuola secondaria: una piattaforma che i principali attori coinvolti (studenti, insegnanti e genitori) possano usare per facilitare la formazione dei ragazzi ed il suo monitoraggio.

I servizi fondamentali per la didattica offerti sono i seguenti:

- Possibilità di visionare e gestire (per gli insegnanti) il materiale didattico.
- Avere sempre a portata di mano i corsi frequentati/insegnati e le informazioni sui relativi docenti.
- Ricevere/dare (per gli insegnanti) comunicazioni tramite post alle classi di interesse per gli utenti (le quali saranno la classe frequentata per l'alunno, la classe di cui fa parte il figlio per il genitore, e le classi affidate all'insegnante).
- Avere a disposizione un calendario dove saranno presenti le comunicazioni importanti (secondo quanto specificato al momento della loro pubblicazione).

In quanto pensata per affiancarsi alle piattaforme già esistenti per la scuola, nell'applicazione vengono memorizzati i dati anagrafici degli utenti, in modo che in futuro i servizi offerti possano essere integrati o usati parallelamente a quelli istituzionali già presenti. Sempre in quest'ottica, gli unici che possono inserire una scuola con le relative classi nel database dell'applicazione sono gli sviluppatori.

1.2 Dati gestiti dal sistema e principali funzionalità

Essa prevede tre tipi di utenti: gli studenti (Student), gli insegnanti (Teacher) e i genitori (Parent). Di ogni utente sono memorizzati in un database il nome, il cognome, il codice fiscale, la data e il luogo di nascita, il sesso, la scuola (tra quelle registrate sulla piattaforma dagli sviluppatori), la classe (tra quelle registrate per ciascuna scuola), il ruolo (tra Studente, Insegnante e Genitore). Saranno memorizzati anche una mail e una password con cui si potrà effettuare il login al sito, e nel caso del Genitore, anche il nome e il cognome del figlio iscritto alla scuola. Ad ogni modo, per agevolare l'accesso alla piattaforma sarà fornito anche il servizio di autenticazione tramite Google. Ogni utente ha una propria Dashboard da dove, a seconda del ruolo, può usufruire di diverse funzionalità. Tutti gli utenti possono vedere le informazioni fondamentali associate al proprio account, il calendario e i post contenenti comunicazioni relative alla classe di interesse pubblicate dagli insegnanti; le comunicazioni possono avere associata anche una scadenza che verrà mostrata sul calendario. I post saranno memorizzati nell'applicazione. Lo Studente può navigare nell'applicazione per consultare le singole pagine relative alle materie studiate, dove è caricato il materiale di cui può fare il download. L'Insegnante può creare i post; togliere, fare il download e l'upload delle risorse relative alla materia che insegna.

Per semplicità nell'implementazione si assume che ogni Insegnante abbia affidata una sola materia, che ogni Genitore possa avere solo un figlio iscritto alla scuola, che ogni materia sia spiegata da un solo Insegnante e che abbia senso definire una scuola senza classi; tutto ciò considerando la classe frequentata dall'alunno. Tuttavia, lo schema del database è pensato per una maggiore scalabilità, e il codice l'applicazione è stato scritto in modo che essa sia facilmente modificabile o ampliabile.

Tutti i file che costituiscono risorse per gli Studenti sono conservati sul Google Drive associato all'account dell'applicazione.

1.2.1 MockUp e User Stories

Le funzionalità e l'aspetto che l'applicazione deve avere sono meglio specificate rispettivamente nelle user stories e nei lo-fi mockup.

Questi due strumenti sono fondamentali nella fase di raccolta dei requisiti e progettazione: fungono da ponte tra il team che si occuperà di relizzare il software e i committenti, così che le aspettative di questi ultimi siano pienamente soddisfatte. Essi infatti sono abbastanza generici in modo che chiunque possa comprenderli, ma allo stesso tempo sono sufficientemente specifici da poter essere implementati dagli sviluppatori.

1.2.2 Lo-Fi MockUp

Di seguito i lo-fi mockup, realizzati con carta e penna, rappresentano ciò che l'utente vedrà sulla piattaforma e come potrà navigarla.

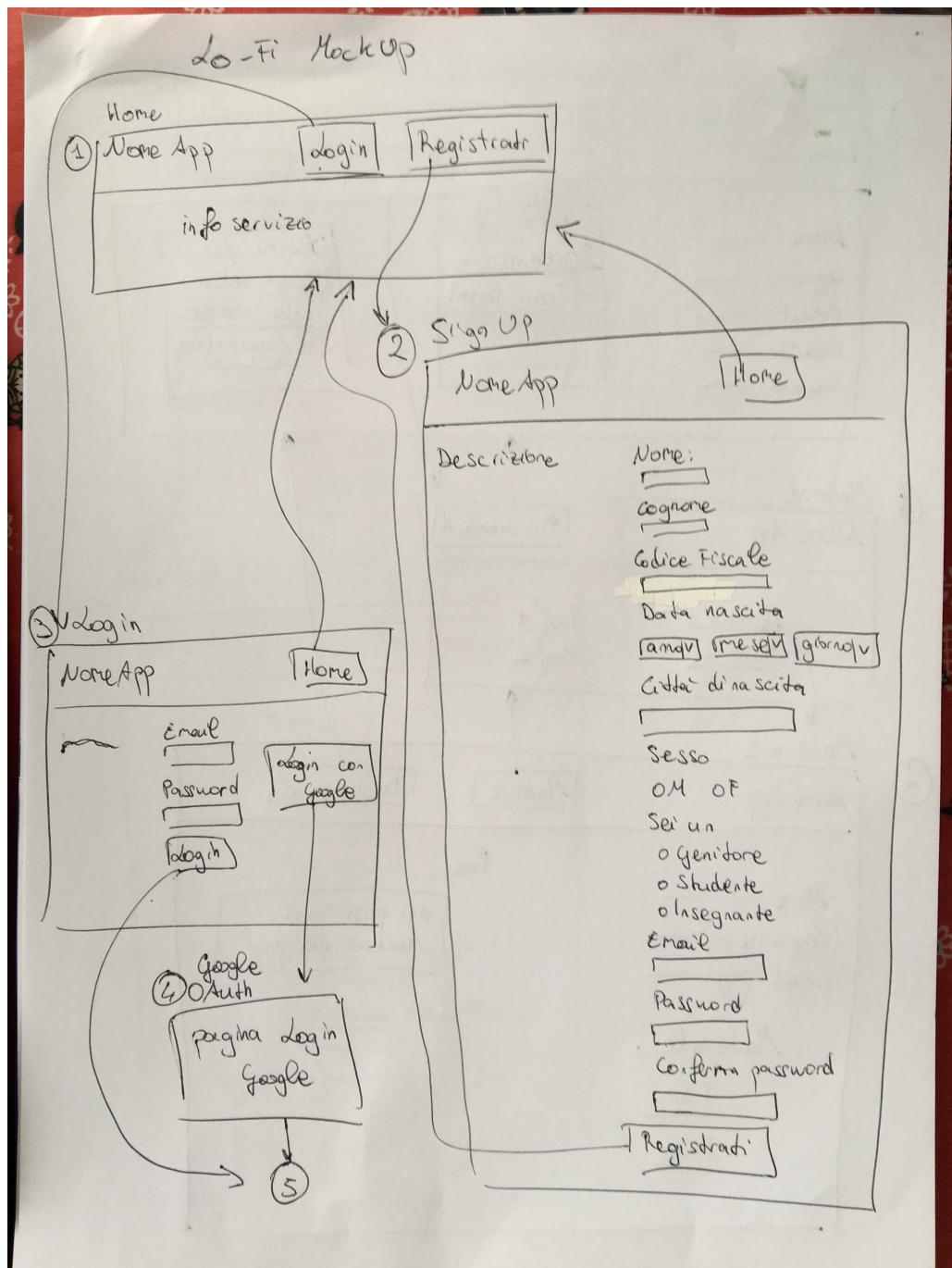


Figura 1.1. Lo-Fi-Mockup 1

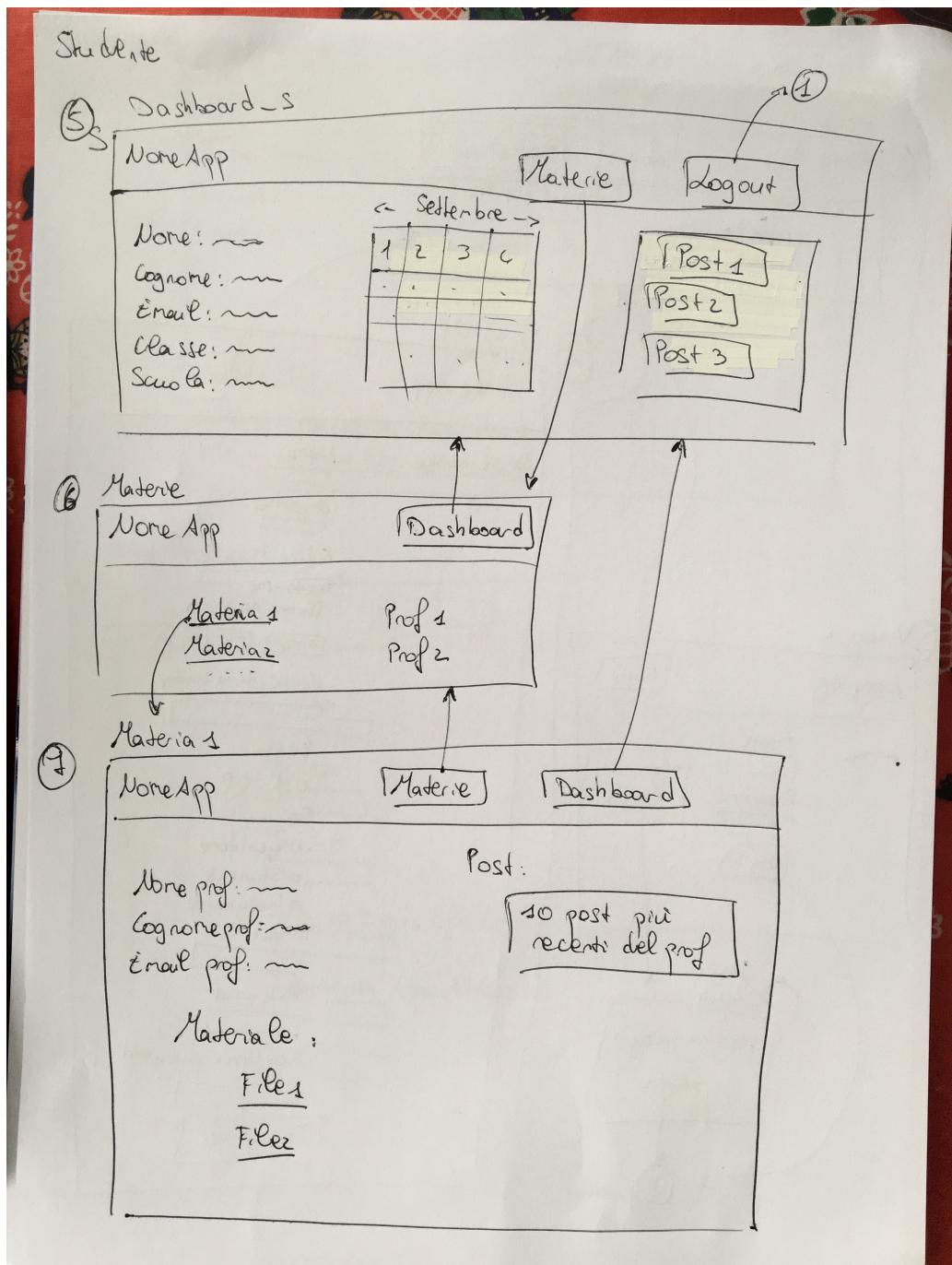


Figura 1.2. Lo-Fi-Mockup 2

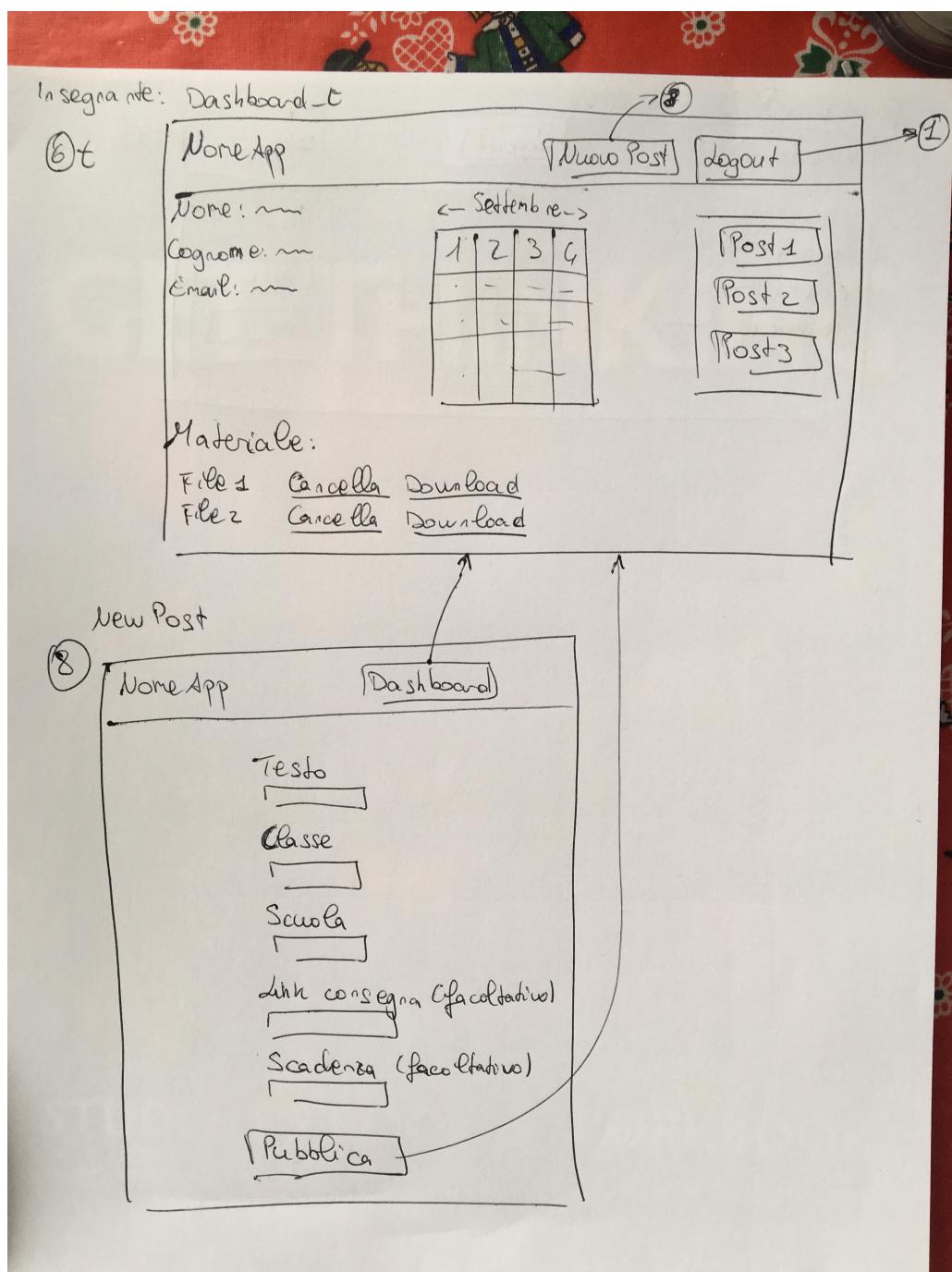


Figura 1.3. Lo-Fi-Mockup 3

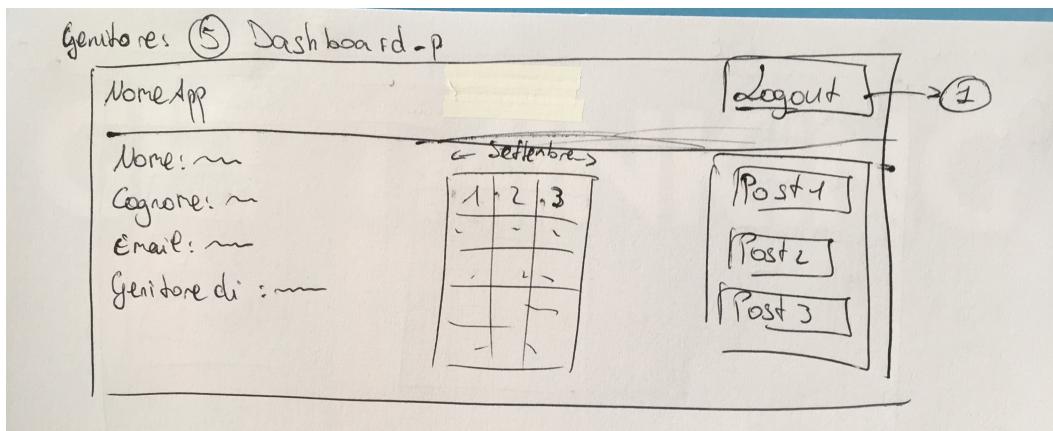


Figura 1.4. Lo-Fi-Mockup 4

1.2.3 User Stories

Di seguito si riporta l'elenco delle user stories: descrivono brevemente ma in maniera efficace le azioni che verranno implementate fornendo i servizi precedentemente descritti agli utenti.

1. Go to the "SignUp" page from the "Home" page

As a user
 So that I can sign up
 Given I'm in the "Home" page
 When I press the button "SignUp"
 Then I should be in the "SignUp" page

2. SignUpAs a user

So that I can use the app's services
 Given I'm in the "SignUp" page
 When I fill the form correctly
 and I press the button "SignUp"
 Then I should be registered in the app
 and I should be in the "Home" page

3. Return to the "Home" page from the "SignUp" page

As a user
 So that I can return to the "Home" page without signing up
 Given I'm in the "SignUp" page
 When I press the button "Home"
 Then I should be in the "Home" page

4. Go to the "LogIn" page from the "Home" page

As a user

So that I can login
Given I'm in the "Home" page
When I press the button "Login"
Then I should be in the "Login" page

5. LogIn without Google

As a user
So that I can use the app's services
Given I'm in the "LogIn" page
and that I've signed up as a "Parent"
When I fill the form correctly
and I press the button "LogIn"
Then I should be logged in as a "Parent" and I should be in the "Dashboard_g" page

6. LogIn with Google

As a user
So that I can use the app's services
Given I've signed up as a "Student" or as a "Teacher"
I want to log in with Google OAuth

7. Go to the "Home" page from the "LogIn" page

As a user
So that I can return to the "Home" page without logging in
Given I'm in the "LogIn" page
When I press the button "Home"
Then I should be in the "Home" page

8. LogOut

As a user
So that I can stop using the app's services
Given I'm in the correct "Dashboard_%" page
When I press the button "LogOut"
Then I should have logged out correctly
and I should be in the "Home" page

9. Calendar

As a user
So that I can know the current date and organize my work properly
Given I'm in the correct "Dashboard_%" page
Then I should see the current date

10. Posts seen by "Student" As a "Student"

So that I can see the posts all my "Teacher"s made
Given I'm on the "Dashboard_s" page
Then I should see the 10 most recent posts

11. Go to the "Dashboard_%" page

As a user
So that I can go to the "Dashboard_%" whenever I need it
When I press the button "Dashboard"
Then I should be in the correct "Dashboard_%" page

12. Materie

As a "Student"
So that I can see the subjects teached at school
Given I'm on the "Dashboard_s" page
When I press the button "Materie"
Then I should be on the "Materie" page

13. Subject info

As a "Student"
So that I can have more information about a subject
Given I'm in the "Materie" page
When I click che link corresponding to subject *
Then I should be in the page "Materia*" corresponding to subject *

14. Go to the "Materie" page from "Materia*" page

As a "Student"
So that I can go to the "Materie" page from the "Materia*" page
Given I'm in the "Materia*" page
When I press the button "Materie"
Then I should be in the "Materie" page

15. Resources for a specific subject

As a "Student"
So that I can use the resources on subject*
Given I'm in the "Materia*" page
and I see the files that are the resources
When I click the link corresponding to a file
Then I should be able to download that file

16. "Teacher"'s posts

As a "Student"
So that I can have a focus on the specific subject*

Given I'm in the "Materia*" page

Then I should see the 5 most recent posts made by the "Teacher" corresponding to that subject

17. Posts seen by "Teacher"

As a "Teacher"

So that I can see understand the workload my "Student"s have

Given I' in the "Dahsboard_t" page

Then I should see the 10 most recent posts made by me and the other "Teacher"s who teach my "Student"s

18. Remove a file

As a "Teacher"

So that I can remove files

Given I'm in the "Dashboard_t" page

When I click on the "Cancella" link corresponding to the file I want to remove

Then the file corresponding to the link I clicked should be removed

19. Download a file

As a "Teacher"

So that I can download files

Given I'm in the "Dashboard_t" page

When I clicked on the "Download" link corresponding to the file I want to download

Then the file corresponding to the link I clicked should be downloaded

20. Upload a file

As a user

So that I can upload a file

Given I'm in the "CaricoFile" page

And that I've specified what file I want to upload

When I press the button "Aggiungi"

Then the file should be uploaded correctly

21. Go to the "NewPost" page from the "Dashboard_t" page

As a "Teacher"

So that I can communicate with my "Student"s and assign homework to them

Given I'm in the "Dashboard_t" page

When I press the button "Nuovo Post"

Then I should be in the "NewPost" page

22. Publish a post

As a "Teacher"

So that I can communicate with my "Student"s and assign homework to them
Given I'm in the "NewPost" page
and that I filled the form correctly
When I press the button "Pubblica"
Then a new post should be published correctly

23. Posts seen by a "Parent"

As a "Parent"
So that I can be updated on my son's scholastic situation
Given I'm in the "Dashboard_p" page
Then I should see the posts made by my son's "Teacher"s

Capitolo 2

Tecnologie e Metodologie utilizzate

2.1 Rails e il paradigma Model-View-Controller

L'applicazione è stata realizzata con Ruby on Rails, seguendo i principi su cui esso è basato, e varie gemme in esso contenuto.

In particolare, Rails è un framework per lo sviluppo di applicazioni web SaaS (Software as a Service) scritto in Ruby. Esso è stato sviluppato tenendo a mente le più importanti linee guida nella progettazione e programmazione di questo tipo di software, in particolare i principi fondamentali sono due: Don't Repeat Yourself (non ci deve essere codice ridondante, ma una singola scrittura efficace delle varie funzionalità e caratteristiche) e Conventions Over Configurations (bisogna specificare delle configurazioni solo nel caso in cui non si seguano quelle raccomandate naturalmente dal framework). Tutto ciò è implementato col paradigma Model-View-Controller: i modelli servono per interagire con i dati del sistema, le view li visualizzano in maniera appropriata rendendo possibile l'interazione con gli utenti, mentre i controller ricevono i comandi di questi (solitamente attraverso le view e il browser) e li eseguono.

2.1.1 Autenticazione locale

Per l'autenticazione è stata utilizzata la gemma Canard per implementare un modello di Role-Based Authentication (ulteriori dettagli nella sezione TODO) insieme alla gemma Devise. Devise è uno strumento molto utile per ogni sviluppatore Rails: è organizzata in moduli, per cui è possibile costruire dal più semplice meccanismo di identificazione e verifica, a quello più complesso con diversi servizi aggiuntivi. In questa applicazione sono stati usati soprattutto i seguenti moduli:

- **Database Authenticatable:** memorizza in maniera sicura e cifrata (grazie al supporto della gemma Bcrypt) la password dell'utente nel database del sistema
- **Registrable:** permette agli utenti di registrarsi
- **Rememberable:** gestisce le sessioni degli utenti tramite i cookie

- **Validatable:** convalida le email e le password
- **Omniauthable:** gestisce autenticazione tramite OAuth (ulteriori dettagli nella sezione successiva).

Ulteriori comodità di questa gemma sono le route e le view da essa create automaticamente per tutti i processi relativi all'autenticazione, personalizzabili in maniera opportuna

2.1.2 Autenticazione tramite Google OAuth

OAuth è un protocollo di rete basato su HTTP che consente di delegare l'autenticazione ad una terza entità (quale potrebbe essere Google, Facebook, Twitter...). In esso sono definiti quattro ruoli principali:

1. **Utente (Resource Owner):** è il possessore dell'account, e deve autorizzare l'accesso alle informazioni contenute in esso e alle risorse protette conservate nel Resource Server
2. **Client (Consumer):** è l'applicazione che vorrebbe accedere alle informazioni e risorse dell'Utente, di cui chiede l'autorizzazione
3. **Resource Server:** il server dove si trovano le risorse protette dell'Utente
4. **Authorization Server:** è il server che si occupa di richiedere all'Utente di autenticarsi (con le credenziali di quest'ultimo) per autorizzare l'accesso alle sue risorse

Lo schema di funzionamento è descritto dalla seguente immagine:

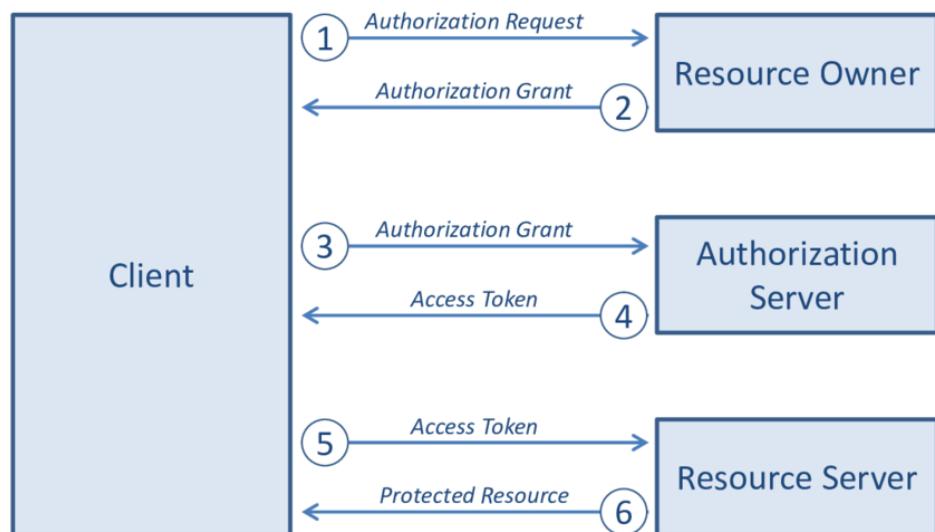


Figura 2.1. OAuth Flow

In questo caso, l'Authorization Server e il Resource Server coincidono (Google), mentre il ruolo del client è svolto dalla nostra applicazione.

Per implementare tutto ciò ci si è serviti, come accennato in precedenza, del modulo `Omniauthable` di Devise in aggiunta alla gemma `omniauth-google-oauth2`.

2.1.3 Google Drive

Come affermato in precedenza, per memorizzare le risorse a disposizione degli studenti si è usato Google Drive.

La prima cosa da fare è stato associare all'applicazione un service account dalla Google API Console. Un service account è un particolare account Google che rappresenta un' entità non umana che necessita di accedere a delle risorse protette. Di conseguenza all'applicazione in esame sono stati associate una email, una password e un ID per Google.

Per gestire i file (upload, download) ed interagire con questo servizio è stata usata la gemma `google-api-client`.

2.1.4 Altre gemme

Per una più completa trattazione dello sviluppo si è cominciato ad impostare il deploy su Heroku, tramite le gemme `rails_12factor` (perconfigurare l'ambiente di produzione dell'applicazione), `pg` (in quanto il database di Heroku è basato su PostgreSQL e non SQLite usato per lo sviluppo) e `figaro` (per gestire le chiavi e gli ID segreti negli ambienti di sviluppo e produzione).

Per realizzare il calendario nella Dashboard degli utenti si è usata `simple_calendar`, mentre per fare i diagrammi UML dei controller e dei model si è utilizzata `railroady`. Di seguito è riportato il `Gemfile` con tutte le gemme utilizzate:

```

1 source 'https://rubygems.org'
2
3 git_source(:github) commentstyledo |repo_name|
4   repo_name = "#{repo_name}/#{repo_name}" commentstyleunless repo_name.include?("/")
5   "https://github.com/#{repo_name}.git"
6 commentstyleend
7
8 # Fix Ruby version
9 ruby '2.4.1'
10 # Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
11 gem 'rails', '~> 5.1.7'
12 # Use Puma as the app server
13 gem 'puma', '>~ 3.7'
14 # Use SCSS for stylesheets
15 gem 'sass-rails', '~> 5.0'
16 # Use Uglifier as compressor for JavaScript assets
17 gem 'uglifier', '>= 1.3.0'
18 # See https://github.com/rails/execjs#readme for more supported runtimes
19 # gem 'therubyracer', platforms: :ruby
20
21 # Use HAML to render the views
22 gem 'haml'
23

```

```
24 # Bootstrap 3 for styling
25 gem 'bootstrap-sass'
26
27 # Calendar
28 gem "simple_calendar", "~> 2.0"
29
30 # Google Drive
31 gem 'google-api-client', '~> 0.11'
32
33 # Use CoffeeScript for .coffee assets and views
34 gem 'coffee-rails', '~> 4.2'
35 # Turbolinks makes navigating your web application faster. Read more:
36     https://github.com/turbolinks/turbolinks
37 gem 'turbolinks', '~> 5'
38 # Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
39 gem 'jbuilder', '~> 2.5'
40 # Use Redis adapter to run Action Cable in production
41 # gem 'redis', '~> 4.0'
42 # Use ActiveModel has_secure_password
43 # gem 'bcrypt', '~> 3.1.7'
44
45 # Security help
46 gem 'devise', '~> 4.7.1'
47 gem 'canard', '~> 0.5.0.pre'
48 gem 'omniauth-google-oauth2'
49
50 # Use Capistrano for deployment
51 # gem 'capistrano-rails', group: :development
52
53 gem 'figaro'
54
55 group :production commentstyledo
56     # Use PostgreSQL in production (Heroku)
57     gem 'pg'
58     # Heroku-specific production settings
59     gem 'rails_12factor'
60 commentstyleend
61
62 group :development, :test commentstyledo
63     # Call 'byebug' anywhere in the code to stop execution and get a debugger console
64     gem 'byebug', platforms: [:mri, :mingw, :x64_mingw]
65     # Use sqlite3 as the database for Active Record
66     gem 'sqlite3'
67     gem 'coveralls', :require => commentstylefalse # to use Coveralls.io for test
68         coverage
69     gem 'simplecov', :require => commentstylefalse
70     #gem 'jasmine-rails'           # to test JavaScript/CoffeeScript, not sure if it
71         will be used
72 commentstyleend
73
74 group :development commentstyledo
75     # Access an IRB console on exception pages or by using <%= console %> anywhere in
76         the code.
77     gem 'web-console', '>= 3.3.0'
78     gem 'listen', '>= 3.0.5', '< 3.2'
79     # Spring speeds up development by keeping your application running in the
80         background. Read more: https://github.com/rails/spring
```

```

76 gem 'spring'
77 gem 'spring-watcher-listen', '~> 2.0.0'
78 # Security testing
79 gem 'brakeman'
80 #gem 'guard-brakeman'
81 commentstyleend
82
83 group :test commentstyledo
84   gem 'gemsurance'           # to check gems' security
85   gem 'rspec-rails'
86   gem 'guard-rspec'
87   gem 'cucumber-rails', :require => commentstylefalse
88   gem 'cucumber-rails-training-wheels' # some pre-fabbed step definitions
89   gem 'database_cleaner'          # to clean Cucumber's test database between runs
90   gem 'capybara'                # lets Cucumber pretend to be a web browser
91   gem 'launchy'                 # a useful debugging aid for user stories
92   gem 'factory_bot_rails'
93 commentstyleend
94
95 # Windows does not include zoneinfo files, so bundle the tzinfo-data gem
96 gem 'tzinfo-data', platforms: [:mingw, :mswin, :x64_mingw, :jruby]

```

Listing 2.1. Gemfile

Una discussione degli strumenti usati per i test si può trovare nella sezione TODO

2.2 Agile per lo sviluppo software

Nel realizzare questa applicazione mi sono ispirata alla metodologia Agile. Essa è in realtà pensata per gruppi di persone che lavorano insieme allo stesso software: le interazione tra i vari membri di un singolo team e tra i responsabili di questi hanno un'enorme importanza, e sono il vero punto di forza di questa metodologia, insieme al contatto con il committente.

Nonostante ciò, in questo lavoro ho voluto tenere fede ai principi di Behavior-Driven Design e Test-Driven Development. Di conseguenza, prima dello sviluppo del codice, nella fase di progettazione, sono stati realizzati i lo-fi mockup e le user stories per fare chiarezza su cosa doveva essere realizzato. Successivamente, durante lo sviluppo, per ogni user story veniva scritto un test e il codice per soddisfarlo, in modo che tutto fosse fatto in modo corretto. Ponendo l'attenzione sulla qualità e sui contenuti da subito si sono evitati lunghi e tortuosi processi di debug, oltre all'implementazione di funzionalità inutili o non richieste.

2.2.1 Tool per applicare al meglio Agile

Per applicare al meglio la metodologia Agile e lavorare efficientemente ho utilizzato i seguenti strumenti:

- **Git:** per il versioning control. La repository su cui è conservato il progetto è la seguente: <https://github.com/SilviadelPiano/HiSchool.git>
- **Visual Studio Code** come editor

- **Draw.io** per la realizzazione degli schemi ER e UML
- **Pivotal Tracker** per il tracking degli sprint e per monitorare i progressi e l'andamento del lavoro. Il link alla pagina corrispondente all'applicazione in esame è il seguente: <https://www.pivotaltracker.com/n/projects/2393952>

Capitolo 3

Analisi Concettuale del Sistema

3.1 Schema Entità-Relazione

In questa sezione viene riportato lo schema ER del database. Come si può vedere esso rappresenta i requisiti descritti nella sezione 1.2, assieme ad altri vincoli necessari per lo sviluppo di questa applicazione. Sono indicati anche gli identificatori principali.

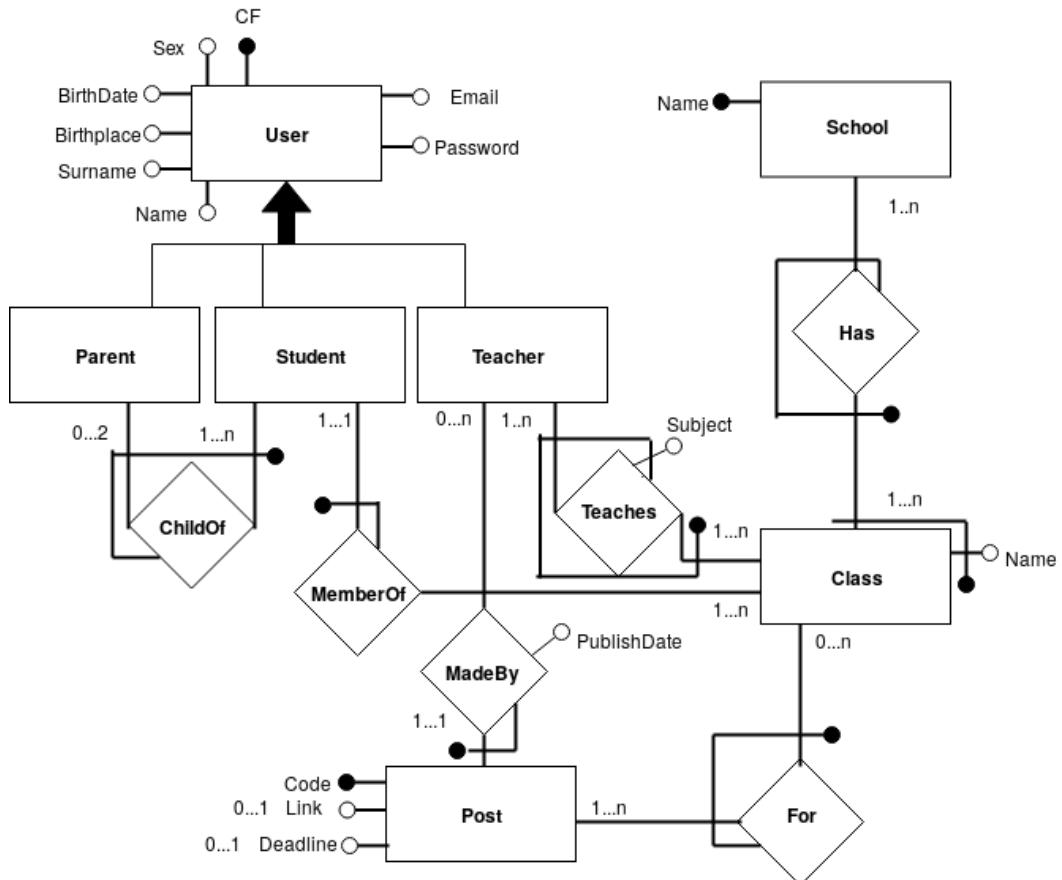


Figura 3.1. Schema ER

3.2 Elenco delle entità e delle relazioni

In questa sezione verranno presentate in maniera più approfondita le varie entità e relazioni situate nello schema.

3.2.1 Generalizzazione User

User è l'entità che rappresenta l'utente che andrà ad usufruire dell'applicazione: si può notare infatti la generalizzazione che differenzia i tre tipi di utilizzatori che si avranno.

Parent e **Student**, cioè le entità che rappresentano il genitore e lo studente, sono legate dalla relazione **ChildOf**: essa esprime la parentela tra lo studente iscritto e il genitore (potrebbe essere anche il tutore legale del ragazzo in mancanza di parenti più stretti) dello stesso. Un Parent per registrarsi sulla piattaforma deve avere almeno un figlio Student registrato su di essa. D'altro canto, uno Student non deve per forza avere un Parent ad esso associato per registrarsi, ma può averne al massimo 2 associati ad esso.

Durante la ristrutturazione Parent, Student e Teacher sono state accorpate a User per generare un'unica entità omonima, dove i vari ruoli sono distinti da un flag, tramite l'ausilio della gemma Canard.

3.2.2 Entità School e Class

L'entità **School** rappresenta la scuola registrata nel database dell'applicazione, ed è legata a **Class** tramite la relazione **Has**. Class è infatti identificata dal proprio nome e dalla Scuola a cui appartiene. Per esempio, la classe 1^a del liceo scientifico sarà presente in tutti i licei scientifici, ed almeno in 1, altrimenti non esisterebbe, da qui l'associazione 1....n. Una Scuola deve avere almeno una classe.

La relazione **Teaches** esprime l'incarico di insegnamento conferito ad un **Teacher** (insegnante) nei confronti della classe, ed è presente anche la materia (subject) oggetto di lezione. Ogni classe può essere presa in carico da diversi insegnanti, ma ne deve avere almeno 1 (si pensi solo alle varie materie presenti in un liceo) ed un singolo insegnante, per essere tale, deve fare lezione ad almeno un classe, ma può averne affidate più di una.

La relazione **MemberOf** invece esprime il fatto che ogni classe debba essere frequentata da almeno uno studente, e questo per essere tale deve, e può, essere iscritto ad una sola classe.

Nella ristrutturazione Class e Has sono state accorpate per generare un'unica entità che tenga traccia del nome della classe e della scuola ad essa associata. School rimane singolarmente, in modo che nel futuro sia possibile associare ad essa più classi. Inoltre, al momento della registrazione all'utente viene chiesto di specificare una classe ed una scuola (in diversi punti dei form da compilare a seconda del ruolo), e sono effettuati due controlli: si verifica l'esistenza della singola scuola (School) nel database, e poi si verifica che per questa sia registrata la classe dichiarata nel form tramite l'entità appena creata con l'accorpamento. Quest'ultima è utile anche per motivi di efficienza del codice.

3.2.3 Entità Post

L'entità **Post** rappresenta le comunicazioni che possono essere fatte sulla piattaforma. Ogni Post deve essere creato da un singolo insegnante secondo quanto espresso dalla relazione **MadeBy**, mentre un insegnante non ha l'obbligo di creare Post, e non c'è limite a quanti ne può fare. L'attributo PublishDate registra la data della pubblicazione. Da notare gli attributi facoltativi Link e Deadline. Essi sono stati inseriti per fornire maggiori opzioni al fine di rendere più efficaci le comunicazioni: come insegnante vorrei poter mettere in evidenza un link dove trovare materiale utile oppure una scadenza o una data importante relativa alla comunicazione appena pubblicata (che sarà segnata sul calendario).

Un Post deve essere associato ad almeno una classe, secondo quanto indicato dalla relazione **For**, in modo che i suoi membri possano vederlo.

Capitolo 4

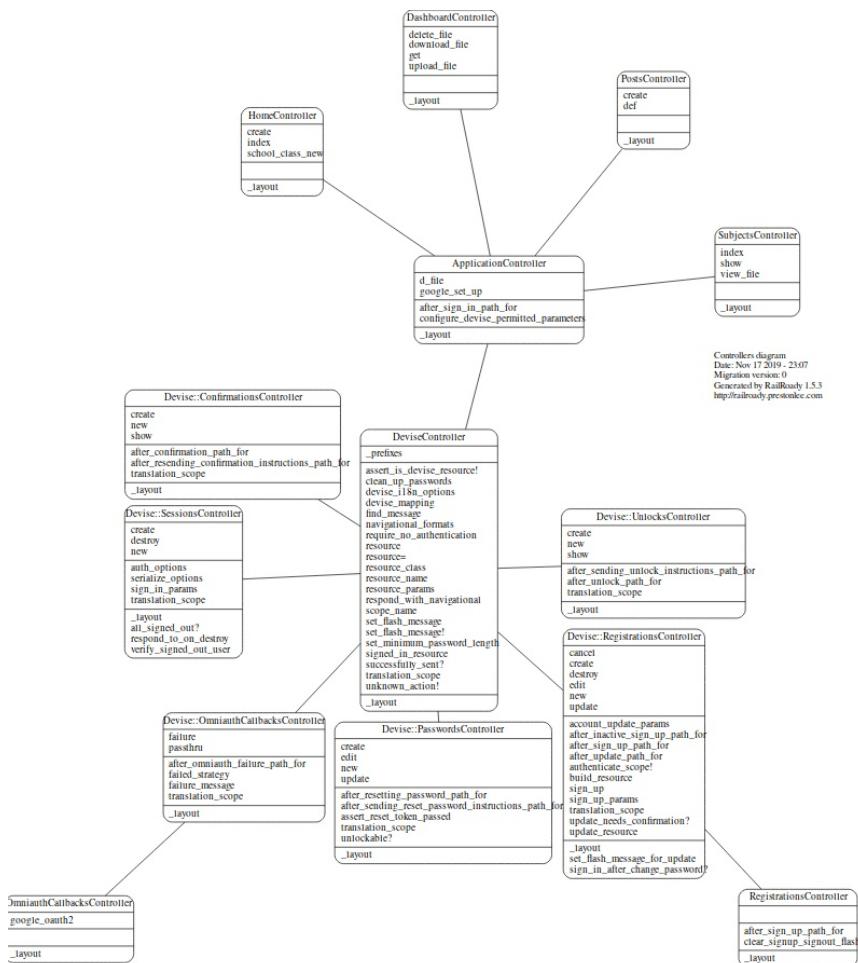
Progettazione del sistema

Questa applicazione è basata su tre layer principali: il Data-Layer, l'Application-Layer e il Presentation-Layer. Di seguito sono esposti gli schemi architetturali per una maggiore comprensione del sistema.

4.1 Archittura del sistema

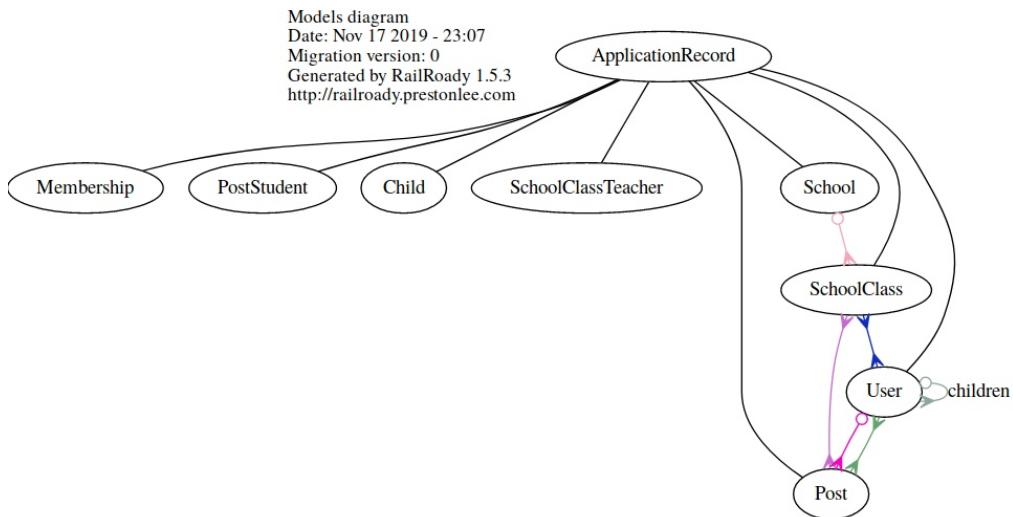
4.1.1 Architettura dei Controller

In questa sezione è riportato lo schema UML dei controller:

**Figura 4.1.** Schema UML Controllers

4.1.2 Architettura dei Model

In questa sezione invece è riportato lo schema UML dei model:

**Figura 4.2.** Schema UML Models

4.1.3 Architettura delle View

Schema UML da fare con Draw.io delle view.

4.2 Progettazione logica del Data Layer

Come spiegato nella sezione 2.1 Rails è basato sul paradigma Model-View-Controller, quindi nel Data-Layer sono conservati i dati che verranno usati dalle altre componenti dell'applicazione. Tutto ciò è possibile, nell'ambiente di sviluppo, grazie alla gemma *sqlite3*.

Rails offre al programmatore un buon livello di automatizzazione di diversi compiti, tra cui la creazione e la modifica del database: lo schema è conservato nel file *db/schema*, il quale viene modificato tramite altri file detti migration, raccolti nella cartella *db/migrate*. In questo modo la probabilità di commettere errori umani (distrazione, battitura, ecc.) è minimizzata e si aumenta l'efficienza del lavoro. Di seguito il file *db/schema.rb*:

```

1 # This file is auto-generated from the current state of the database. Instead
2 # of editing this file, please use the migrations feature of Active Record to
3 # incrementally modify your database, and then regenerate this schema definition.
4 #
5 # Note that this schema.rb definition is the authoritative source for your
6 # database schema. If you need to create the application database on another
7 # system, you should be using db:schema:load, not running all the migrations
8 # from scratch. The latter is a flawed and unsustainable approach (the more
9 # migrations
10 # you'll amass, the slower it'll run and the greater likelihood for issues).
11 #
12 # It's strongly recommended that you check this file into your version control
   system.
  
```

```
13 ActiveRecord::Schema.define(version: 20191014161620) do
14
15   create_table "children", force: :cascade do |t|
16     t.integer "parent_id"
17     t.datetime "created_at", null: false
18     t.datetime "updated_at", null: false
19     t.bigint "child_id"
20     t.index ["child_id"], name: "index_children_on_child_id"
21     t.index ["parent_id"], name: "index_children_on_parent_id"
22   end
23
24   create_table "memberships", force: :cascade do |t|
25     t.integer "user_id"
26     t.integer "schoolClass_id"
27     t.datetime "created_at", null: false
28     t.datetime "updated_at", null: false
29     t.index ["schoolClass_id"], name: "index_memberships_on_schoolClass_id"
30     t.index ["user_id"], name: "index_memberships_on_user_id"
31   end
32
33   create_table "posts", force: :cascade do |t|
34     t.string "link"
35     t.date "deadline"
36     t.integer "user_id"
37     t.datetime "created_at", null: false
38     t.datetime "updated_at", null: false
39     t.text "description"
40     t.bigint "school_class_id"
41     t.index ["user_id"], name: "index_posts_on_user_id"
42   end
43
44   create_table "school_classes", force: :cascade do |t|
45     t.string "name"
46     t.integer "school_id"
47     t.datetime "created_at", null: false
48     t.datetime "updated_at", null: false
49     t.index ["school_id"], name: "index_school_classes_on_school_id"
50   end
51
52   create_table "school_classes_teachers", force: :cascade do |t|
53     t.integer "school_class_id", limit: 8
54     t.integer "teacher_id", limit: 8
55     t.string "subject"
56     t.index ["school_class_id"], name:
57       "index_school_classes_teachers_on_school_class_id"
58     t.index ["teacher_id"], name: "index_school_classes_teachers_on_teacher_id"
59   end
60
61   create_table "schools", force: :cascade do |t|
62     t.string "name"
63     t.datetime "created_at", null: false
64     t.datetime "updated_at", null: false
65   end
66
67   create_table "users", force: :cascade do |t|
68     t.string "email", default: "", null: false
69     t.string "encrypted_password", default: "", null: false
70   end
```

```
69   t.string "reset_password_token"
70   t.datetime "reset_password_sent_at"
71   t.datetime "remember_created_at"
72   t.datetime "created_at", null: commentstylefalse
73   t.datetime "updated_at", null: commentstylefalse
74   t.integer "roles_mask"
75   t.string "name", default: "", null: commentstylefalse
76   t.string "surname", default: "", null: commentstylefalse
77   t.string "birth_place", default: "", null: commentstylefalse
78   t.date "birth_date", default: "2019-09-04", null: commentstylefalse
79   t.string "sex", default: "", null: commentstylefalse
80   t.string "CF", default: "", null: commentstylefalse
81   t.index ["CF"], name: "index_users_on_CF", unique: commentstyletrue
82   t.index ["email"], name: "index_users_on_email", unique: commentstyletrue
83   t.index ["reset_password_token"], name: "index_users_on_reset_password_token",
     unique: commentstyletrue
84   commentstyleend
85
86 commentstyleend
```

Listing 4.1. db/schema.rb

Da notare gli attributi *index*: essi indicano delle foreign key, cioè riferimenti ad altre tabella.

Capitolo 5

Realizzazione del sistema

5.1 Application Layer

Qualche parola sul file application.controller + tabella con le route principali + codice controller principali (da scegliere)

5.2 Presentation Layer

Spiegazione dell'organizzazione delle view (application.layout), uso di bootstrap e scss + screenshot delle view con 1 riga di descrizione per ognuna

Capitolo 6

Dispiegamento e Validazione

6.1 Installazione

Passi da git clone a rails server

6.2 Test

Metodo Agile -> Test con cucumber e capybara (mostra i file più importanti) + file rspec con oauth. brakeman e gembulance per i test di sicurezza. byebug per il debugging durante lo sviluppo