



**SAPIENZA**  
UNIVERSITÀ DI ROMA

# CondominioOrganizer: progetto ed implementazione di un'applicazione web per la gestione di condomini

**Facoltà di Ingegneria dell'informazione, informatica e statistica**  
**Dipartimento di INGEGNERIA INFORMATICA, AUTOMATICA E GESTIONALE**  
**Corso di laurea in Ingegneria Informatica e Automatica**

**Andrea Panceri**  
**Matricola 1884749**

Relatore  
Ing. Giovanni Farina

A.A 2021/2022



*“Faber est suae quisque fortunae.”*

Sallustio

# Indice

<b>Indice delle figure.....</b>	<b>3</b>
<b>Indice delle tabelle .....</b>	<b>4</b>
<b>1. Introduzione.....</b>	<b>5</b>
1.1. Scopo della tesi.....	5
1.2. Architettura di riferimento.....	6
1.3. Organizzazione della tesi.....	8
<b>2. Raccolta e analisi dei requisiti .....</b>	<b>10</b>
2.1. Descrizione della realtà d'interesse .....	10
2.2. Funzionalità offerte .....	12
<b>3. Tecnologie e metodologie utilizzate .....</b>	<b>22</b>
3.1. I metodi agili.....	22
<b>4. Analisi e progettazione concettuale .....</b>	<b>28</b>
4.1. Analisi dei requisiti .....	28
4.2. La progettazione concettuale.....	30
4.3. Lo schema Entità-Relazione .....	30
4.4. <i>I Vincoli d'integrità esterni</i> .....	36
<b>5. Progettazione del sistema.....</b>	<b>38</b>
5.1. Progettazione logica del Data Layer .....	38

---

5.2. Architettura dell'applicativo software.....	43
<b>6. Realizzazione del sistema .....</b>	<b>51</b>
6.1. Accesso al sito web.....	51
6.2. Ciclo di vita di un condomino .....	55
<b>7. Validazione e dispiegamento .....</b>	<b>57</b>
7.1. Il Software Testing.....	57
7.2. Behavior-driven development .....	58
7.3. Test-driven development .....	64
7.4. Dispiegamento .....	68
<b>Bibliografia .....</b>	<b>72</b>

## Indice delle figure

Figura 1 Architettura three-tier.....	6
Figura 2 Architettura three-tier in Ruby on rails.....	7
Figura 3 Mokup relativa alla user stories 1 e 2 .....	13
Figura 4 Mokup relativa alla user story 3.....	13
Figura 5 Mokup relativa alle user story 4-5-6-7-8-9-10-11 .....	15
Figura 6 Mokup relativa alle user stories 1-2-3-12 .....	16
Figura 7 Mockup delle funzioni di un condomino .....	17
Figura 8Mockup delle funzioni di un Leader condominio.....	20
Figura 9 dashboard admin .....	21
Figura 10 Schema programmazione estrema.....	24
Figura 11 Rappresentazione grafica del pattern MVC .....	26
Figura 12 Schema Entità-Relazione .....	31
Figura 13 Schema Entità-Relazione ristrutturato .....	40
Figura 14 Schema relazionale.....	41
Figura 15 Modello Ruby on Rails .....	43
Figura 16 Vista Ruby on Rails .....	44
Figura 17 Controller Ruby on Rails.....	44
Figura 18 Diagramma a classi dei modelli .....	46
Figura 19 Diagramma delle classi Condomini e Condomini .....	48
Figura 20 Diagramma delle classi Admin .....	49
Figura 21 Metodo per utilizzare api di Google Drive.....	50
Figura 22 Diagramma delle classi Cartelle condominio e Cartelle Condomino.....	50
Figura 23 Gerarchia ruoli .....	51
Figura 24 Diagramma attività: autenticazione utente registrato .....	54
Figura 25 Diagramma a stati di un condomino.....	55
Figura 26 Diagramma casi d'uso .....	56
Figura 27 Diagramma attività: creazione di un gruppo condominio.....	56
Figura 28 Red-green-refactor loop.....	64
Figura 29 Home page, Sign up e Log in .....	69
Figura 30 Bacheca di un condominio.....	70
Figura 31 Pagina di iscrizione a un "gruppo condominio" .....	71
Figura 32 cartella Google Drive di un "gruppo condominio".....	71

## Indice delle tabelle

Tabella 1 Dizionario dei dati: entità .....	32
Tabella 2 Dizionario dei dati: relazioni.....	33
Tabella 3 Dizionario dei dati: attributi.....	34
Tabella 4 Dizionario dei dati: Vincoli di cardinalità .....	36
Tabella 5 Dizionario dei dati: Vincoli esterni.....	37
Tabella 6 Vincoli esterni.....	42
Tabella 7 Permessi e ruoli .....	53

# 1. Introduzione

## 1.1. Scopo della tesi

Lo scopo di questa attività è stato quello di ideare e realizzare un'applicazione informatica che fornisca, in un unico applicativo, le informazioni di interesse di un condominio. La realizzazione della stessa ha permesso: creazione di eventi, condivisione di documenti d'interesse del condominio e scambio di informazioni tramite e-mail o bacheca, tutto questo sfruttando anche i servizi informatici messi a disposizione da Google come Google Drive, Google Calendar e Gmail.

Nella tesi verranno illustrate le metodologie utilizzate durante lo sviluppo e le scelte fatte durante il progetto.

Nello sviluppo dell'applicazione è stata usata una delle metodologie Agile, la metodologia *Xtreme Programming*, nel quale lo sviluppatore perfeziona continuamente un prototipo funzionante, ma incompleto, fino a quando il cliente non è soddisfatto del risultato, questa tipologia di sviluppo si contrappone a quella tradizionale<sup>1</sup>.

Questo metodo influisce in maniera considerevole sul normale flusso di sviluppo del software, soprattutto nella strutturazione del testing. Per tale ragione nel capitolo finale vedremo come sono stati applicati gli approcci TDD (Test Driven Development) e BDD (Behaviour Driven Development) all'applicativo d'interesse.

Il sito web è stato realizzato in Ruby on Rails, un framework che ha stravolto le metodologie standard per lo sviluppo web. Il successo di questo framework indubbiamente è da attribuire al linguaggio in cui è stato scritto, Ruby, un linguaggio di alto livello orientato agli oggetti, che agevola lo sviluppatore con la sua sintassi molto semplice, molto simile a un linguaggio famosissimo come Python.

---

<sup>1</sup> Tra le metodologie di sviluppo tradizionali, che non verranno approfondite in questa tesi, troviamo: Waterfall, Spiral e Rational Unified Process (RUP).



## 1.2. Architettura di riferimento

L'architettura che prenderemo come riferimento è quella denominata *architettura three-tier* ("a tre strati"), in cui l'applicazione informatica si organizza in 3-tier di calcolo logici e fisici, che sono: *presentation layer* dedicato alla interfaccia utente, *application logic layer* dedicato alla logica funzionale e *resource management layer* dedicato all'archiviazione e alla gestione dei dati associati all'applicazione.

Questa struttura porta numerosi vantaggi, il principale è sicuramente il fatto che i tre strati possono essere eseguiti su infrastrutture differenti, distribuiti su nodi della rete diversi tra loro, in una rete anche eterogenea., ognuno può essere modificato e gestito da enti differenti, così che qualsiasi intervento su uno dei tre strati non vada ad influire sugli altri due.

Il *presentation layer* rappresenta l'interfaccia dell'utente, permette all'utente di interagire con le applicazioni. Il fine di questo strato è quello di visualizzare le informazioni e raccogliere le informazioni eventualmente inserite dall'utente. Normalmente viene eseguito da un browser web e sviluppato utilizzando HTML, CSS e Javascript.

Il *application logic layer* è la parte centrale dell'applicazione in cui vengono raccolte ed elaborate le informazioni raccolte dallo strato precedente secondo una certa logica, durante l'elaborazione può interagire con lo strato inferiore che ha accesso ai dati salvati dall'applicazioni. Ci sono svariati linguaggi per sviluppare questo strato come Java e PHP, nel nostro caso è stato utilizzato ruby.

Il *resource management layer* è lo strato in cui vengono gestite e memorizzate le informazioni elaborate dallo strato precedente, tramite un sistema di gestione di basi di dati che può essere relazione o non relazionale, nel nostro caso specifico è stato usato un DBMS relazionale, nello specifico PostgreSQL.

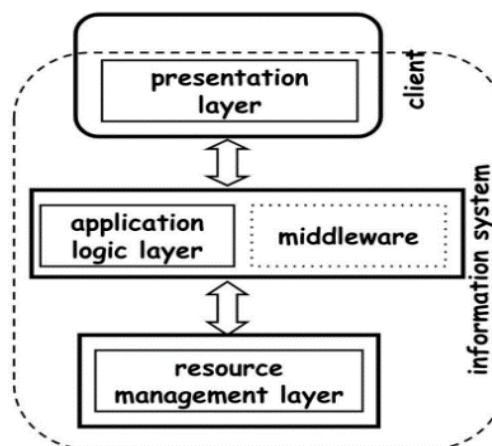


Figura 1 Architettura three-tier

Lo schema appena descritto (illustrato dalla *Figura 1*) rappresenta la base della maggior parte degli applicativi contemporanei, ma può essere ampliato anche con più strati prendendo il nome di *n-tier* o *multi-tier*, ma ciò non aggiunge benefici, anzi porta a una maggiore complessità, che non determina spesso la scelta di questi tipi di architetture. (IBM Cloud Education, 20 luglio 2022)

L'architettura mostrata ha molte similitudini con un altro pattern diffusissimo, il Model-View-Controller, che verrà analizzato nel seguito della trattazione, svolgendo una parte cruciale nello sviluppo di applicazioni Web, in cui i tre strati prendono un significato più specifico, legato al contesto: lo strato di presentazione è rappresentato da un Web server, lo strato della logica dell'applicazione è rappresentato da un application server, mentre lo strato della gestione delle risorse è rappresentato da un RDBMS (sistema di gestione di basi di dati relazionale).

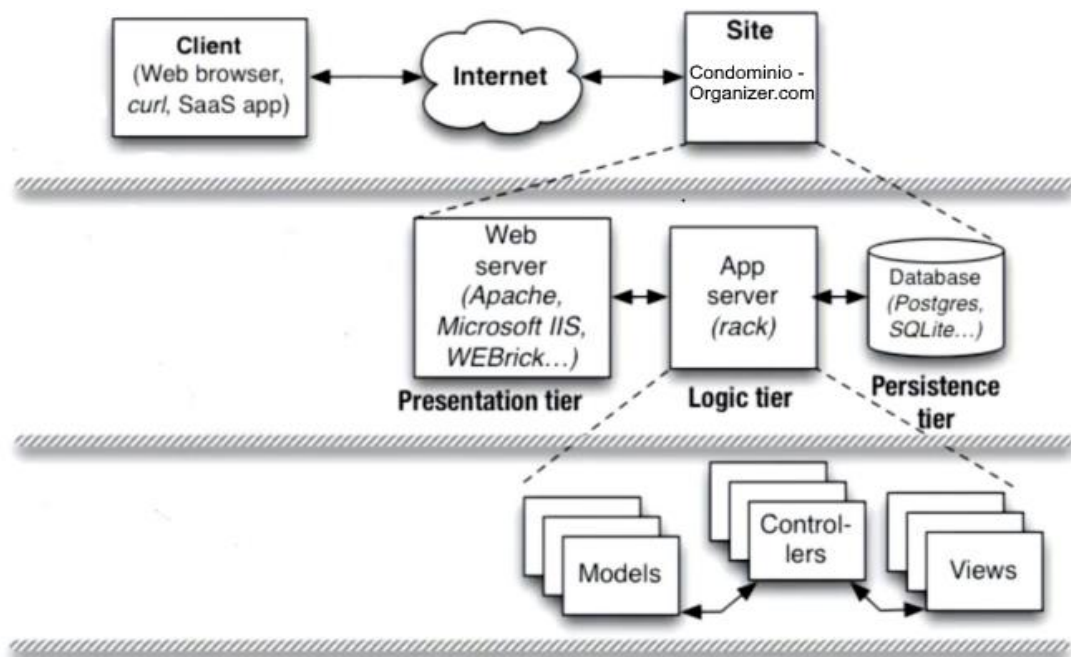


Figura 2 Architettura three-tier in Ruby on rails

### 1.3. Organizzazione della tesi

Prima di entrare nel vivo della trattazione delle scelte progettuali dell'applicazione informatica, dopo la breve introduzione scritta, descriviamo l'organizzazione della tesi.

Nel prossimo capitolo, *Raccolta ed analisi dei requisiti*, verranno analizzati in dettaglio i requisiti richiesti dall'applicazione web avvalendosi di User stories e mock-up, per aiutare il lettore alla comprensione degli obiettivi che si sono voluti raggiungere.

Nel capitolo 3, *Tecnologie e Metodologie Utilizzate*, vengono descritti in maniera approfondita i metodi *agili* che sono stati accennati all'inizio della tesi, il framework *Ruby on Rails* che è il cuore di tutta l'applicazione e tutte le altre tecnologie che sono state utilizzate nell'implementazione del sistema.

Nel capitolo 4, *Analisi e progettazione concettuale*, verranno analizzati i dati che troviamo nel sistema informatico. Inoltre, si parlerà della progettazione concettuale, e come si elabora, di conseguenza lo schema concettuale. Lo schema concettuale verrà prodotto dai requisiti raccolti e mostrati nel capitolo 2 della discussione.

Nel capitolo 5, *Progettazione del sistema*, entreremo nel vivo dell'architettura dell'applicazione. Partendo dall'illustrazione del diagramma ER (entità-relazione) verranno prodotte le tabelle corrispondenti nel modello relazione. Successivamente seguendo il modello MVC (Model-view-controller) con *Ruby on rails*, ad ogni tabella che abbiamo prodotto sarà affiancato un modello che al seguito di un comando di un controller, conseguenza di un'iterazione dell'utente con il sistema, effettuerà un'operazione CRUD<sup>2</sup>.

Il capitolo sarà diviso in due parti principali: la prima parte prevederà la progettazione logica, in cui verrà elaborato lo schema ER, per arrivare ad uno schema direttamente traducibile nello schema relazionale, cioè lo schema logico, mentre nella seconda parte si useranno i diagrammi UML per mappare tutte le componenti della nostra applicazione nello schema relazionale elaborato precedentemente.

---

<sup>2</sup> CRUD è un acronimo che deriva dai nomi delle operazioni di primaria importanza per interagire con un database: Create, Retrieve, Update, Delete.

Nel capitolo 6, *Realizzazione del sistema*, si mostrerà da un punto di vista di alto livello, di un caso d'utilizzo dell'applicazione, in particolare si concentrerà l'attenzione sul processo di creazione di un "gruppo condominio" e sui partecipanti dei gruppi. Ovviamente queste operazioni possono essere effettuate solamente da utenti autenticati, e verranno approfondite le varie abilità di gestione dei Leader condominio e dei semplici condòmini.

Infine, nel capitolo 7, *Validazione e dispiegamento*, verranno descritti i test implementati, uno strumento utile per individuare eventuali difetti nell'applicazione, e le tecnologie utilizzate per effettuarli. Infine, indicazioni per distribuire l'applicazione e come un utente può eseguire l'applicazione software, a partire dalla code base su GitHub.

## 2. Raccolta e analisi dei requisiti

### 2.1. Descrizione della realtà d'interesse

L'idea dietro alla creazione di questa applicazione informatica nasce dal bisogno dei condòmini di avere una comunicazione più efficiente con gli amministratori di condominio, che spesso deve seguire un iter molto lungo prima di poter giungere correttamente al destinatario. Molte volte per mettersi in comunicazione con il proprio amministratore, che il condomino potrebbe non avere mai visto o sapere la sua identità, deve effettuare una ricerca online o addirittura attraverso un passa parola con i propri vicini. Riportiamo un estratto di questo sito web che mette in risalto codesto problema, sia dal punto di vista pratico che da quello legislativo:

«conoscere il nome dell'amministratore di condominio non sempre è così facile, o quanto meno avere certezza ufficiale della persona investita dell'incarico non è cosa semplice...non esiste un'anagrafe pubblica dei condomini e delle relative persone che lo rappresentano...se voglio sapere chi è l'amministratore del condominio Alfa dove abita Tizio devo andare davanti al portone d'ingresso del condominio Alfa. Motivo ai sensi dell'articolo 1129, quinto comma, c.c.:

*Sul luogo di accesso al condominio o di maggior uso comune, accessibile anche ai terzi, è affissa l'indicazione delle generalità, del domicilio e dei recapiti, anche telefonici, dell'amministratore.*

Si tratta della così detta targa dell'amministratore. Il problema è che all'inadempimento di quanto prescritto da questa norma non è ricollegata alcuna sanzione, ergo: da quanto la norma è entrata in vigore non si contano le segnalazioni di inadempimento.

Lo stesso discorso deve farsi per il condominio che non ha un vero e proprio amministratore ma un così detto facente funzione.

Il sesto comma dell'articolo 1129 specifica che *"in mancanza dell'amministratore, sul luogo di accesso al condominio o di maggior uso comune, accessibile anche ai terzi, è affissa l'indicazione delle generalità e dei recapiti, anche telefonici, della persona che svolge funzioni analoghe a quelle dell'amministratore"*.

Qualcuno potrebbe pensare: in mancanza di queste targhe conoscendo il codice fiscale del condominio ci si può rivolgere all'Agenzia delle entrate. A livello telematico l'ente consente di verificare solamente l'esistenza del codice fiscale (<https://goo.gl/2dOWsc>) e non anche la persona cui è associato.

Da quanto ci risulta gli uffici non sono obbligati a fornire questa informazione, che, comunque, non fornisce garanzia assoluta di attualità, posto che il cambio di intestazione del codice fiscale è onere dell'amministratore subentrante.

Ed allora? In mancanza di targa identificativa (che per le medesime ragioni appena esposte non dà alcuna certezza, in fondo l'amministratore potrebbe essere stato revocato e la targa lasciata lì per dimenticanza), come fare a sapere chi è l'amministratore di condominio?

Semplice (si fa per dire): basta recarsi presso l'edificio ed iniziare a suonare ai citofoni chiedendo informazioni. La signora Maria ed il signor Mario di turno, alle volte, sono più informati degli uffici preposti (che in questo caso non esistono).

È evidente che l'importanza dell'informazione in esame sia fin troppo sottovalutata e sarebbe auspicabile un più incisivo quadro normativo che consenta di tenere costantemente aggiornata l'anagrafe dei legali rappresentanti dei condomini. ».<sup>3</sup>

Dopo aver letto questo breve articolo è chiaro al lettore che c'è un problema di fondo, ed è qui che il suddetto progetto vuole intervenire fornendo ad un nuovo condomino un modo chiaro e veloce per entrare in comunicazione con l'amministratore.

La piattaforma che è stata realizzata non vuole solo agevolare il condomino, ma anche aiutare nel suo lavoro l'amministratore che tra i suoi compiti deve indire le assemblee condominiali, condividere bollette ed altri file di interesse, e questa applicazione, che sarà presentata nel corso della stesura del progetto, viene incontro a queste necessità. Infatti, la comunicazione dell'amministratore sarà più veloce ed immediata perché potrà viaggiare attraverso Google e le sue applicazioni, come ad esempio la creazione di eventi su Google Calendar e la condivisione di file di qualsiasi natura con Google Drive.

Oltre a questi compiti appena citati, l'amministratore nella realtà quotidiana deve mandare comunicazioni sia di carattere personale a singoli condòmini, che di carattere generale a tutti gli appartenenti di un condominio. Inoltre, non sempre l'amministratore agisce singolarmente, ma spesso delega i suoi compiti ad un terzo, e rispetto ai decenni scorsi, grazie alle nuove normative un amministratore può vivere anche a centinaia di chilometri dai condòmini amministrati, nasce quindi il bisogno di nuove tecnologie per lavorare a distanza o da remoto.

Mentre dall'altra prospettiva, quella del condomino, l'applicazione è interessante perché permette di creare una *community*, che agevola la comunicazione, la rende fruibile in tempo reale, pensando anche a quei condòmini che sono fuorisede e agli stessi amministratori di condominio, non sempre appartenenti al comune di riferimento in cui è collocato lo stabile da amministrare.

---

<sup>3</sup> Fonte: <https://www.condominioweb.com/come-conoscere-il-nome-dellamministratore.13654#2>

## 2.2. Funzionalità offerte

L'applicazione informatica che stiamo progettando punta ad aggregare tutte le informazioni di interesse di un condominio, con un sistema di "gruppi condominio" gestiti da un Leader Condominio forniti di bacheca dove inserire comunicazioni condominiali. Permette di organizzare l'invio di eventi "pagamento" tramite e-mail, tenere traccia di eventi come assemblee, tramite un utilizzo simbiotico della bacheca locale fornita al gruppo, l'invio di e-mail ai membri del gruppo e Google Calendar. Fornisce inoltre un Drive privato ad ogni nuovo condominio con la creazione di una nuova cartella per ogni condòmino, così da semplificare la condivisione di documenti tra l'amministratore ed i condòmini.

Quindi ora c'è bisogno di entrare nello specifico di tutte le funzionalità offerte, e per farlo divideremo il capitolo in diverse sezioni, nelle quali verranno approfondite le funzionalità richieste facendo uso di diversi strumenti come *User Stories* e *mock-up*.

### 2.2.1. Sezione utenti non registrati

Un utente che accede per la prima volta al sito è un *utente non registrato*, e sono disponibili per lui solo alcune funzionalità limitate. Per poter sfruttare le funzionalità del sito web ha bisogno di una pagina per registrarsi, i dati richiesti per registrarsi sono:

NOME E COGNOME: fondamentali per essere facilmente riconoscibili nei vari gruppi condominio, così che gli altri membri sanno esattamente con chi si stanno rivolgendo.

E-MAIL: campo univoco per ogni membro, che lo distingue da ogni altro utente del sito, e fondamentale per tutte le funzionalità del sito dalle basilari e-mail, alle cartelle drive e gli eventi su calendar.

PASSWORD: per permettere all'utente di accedere in sicurezza all'account, essa deve essere almeno di 8 caratteri.

Inoltre, la registrazione può essere effettuata sia inserendo i dati richiesti direttamente nel sito, ma anche tramite OAuth 2<sup>4</sup>, in particolare tramite Google, in cui l'utente dà il consenso al trattamento dei dati del proprio account.

Altra funzione disponibile, è quella del *Contact Us*, che permette di inviare recensioni o problematiche dell'applicazione direttamente ai proprietari del suddetto.

---

4 "OAuth 2 è un protocollo standard aperto che consente alle applicazioni di accedere alle risorse protette di un servizio per conto dell'utente. OAuth 2.0 definisce flussi di autorizzazione per applicazioni native, applicazioni web e dispositivi mobili"

Fonte: <https://www.teranet.it/introduzione-ad-oauth-2>.

Una volta creato un profilo, l'utente diventa un *utente registrato*. Da quanto appena detto, sono state definite per gli *utenti non registrati* le seguenti user stories:

1. *Come Utente non registrato, in modo da diventare un utente registrato voglio potermi iscrivere con la mia e-mail, nome e cognome.*
2. *Come Utente non registrato, in modo da diventare un utente registrato voglio poter accedere con il mio account Google.*
3. *Come Utente non registrato, in modo da ottenere informazioni sull'applicazione voglio poter accedere ad una pagina Homepage.*
4. *Come Utente non registrato, in modo da ottenere le informazioni per contattare gli sviluppatori, voglio avere una pagina Contact Us.*

Questi sono invece sono dei mock-up per chiarire le funzionalità che si aspetta l'utente graficamente:

The mockup shows a web browser window titled 'A Web Page' with the URL 'https://condominio-organizer/signup'. The page content is titled 'ISCRIVITI' and contains the following elements:

- Input fields for 'Username', 'E-mail', 'Password', and 'Ripeti password'.
- A 'Crea account' button.
- Text: '...Oppure accedi con Google'.
- A 'Iscriviti con Google' button.
- Links: 'Torna alla home' and 'Vai alla pagina di Login'.

Figura 3 Mockup relativa alla user stories 1 e 2

The mockup shows a web browser window titled 'A Web Page' with the URL 'https://condominio-organizer'. The page layout includes:

- A header with a 'Logo Sito' on the left and 'Login | Iscriviti' on the right.
- A large central area titled 'Descrizione' containing several lines of placeholder text.
- A 'Contact Us' link at the bottom.

Figura 4 Mockup relativa alla user story 3



### 2.2.2. Sezione utente registrato

Dopo aver definito gli *utenti non registrati*, passiamo agli *utenti registrati*. Questi una volta effettuata la registrazione, possono vedere i condomini vicini a loro tramite le API<sup>5</sup> di Geocoder, ricercare manualmente tra una lista di condomini nel suo comune ed effettuare una richiesta d'accesso per essi o iscriversi direttamente con un codice d'invito fornito da un amministratore, diventando un *condomino* del condominio selezionato. Nel caso in cui non esistesse il suo condominio, l'utente registrato potrà creare il suo gruppo, il che comporterà automaticamente il passaggio a *Leader Condominio* per quest'ultimo. Inoltre, un utente registrato può visualizzare le informazioni del proprio account e modificarle e può visualizzare la lista dei condomini a cui partecipa, amministra o per cui ha fatto richiesta. Ogni volta che vorrà accedere a codeste funzionalità, l'utente dovrà accedere al sito attraverso una pagina di login, per proteggere i propri dati da malintenzionati, usando la e-mail e password inserite durante la registrazione (o eventualmente accedere tramite il proprio account Google).

Ora definiamo le specifiche che devono essere rispettate durante la creazione di un condominio:

NOME: in questo campo devono essere inserito un nome che identifica facilmente il condominio.

COMUNE: in questo campo deve essere inserito il comune in cui è localizzato il condominio, così che sarà visibile durante la ricerca per comune.

INDIRIZZO: in questo campo deve essere inserito l'indirizzo del comune, secondo il formato italiano (Via Tiburtina 214), necessari anche questo per permettere alle API di Geocoder di generare le coordinate del condominio.

BANNER: in questo campo deve essere caricata un'immagine che diventerà il banner del condominio, è opzionale, nel caso di mancato inserimento verrà usata un'immagine di default.

Da quanto appena detto, sono state definite per gli *utenti registrati* le seguenti user stories:

1. *Come Utente registrato, in modo da poter selezionare un condominio vicino a me e diventare un condomino, voglio poter visualizzare la lista dei condomini in uno specifico comune.*

---

5 "Le API (Application Programming interface) sono meccanismi che consentono a due componenti software di comunicare tra loro usando una serie di definizioni e protocolli."  
Fonte: <https://aws.amazon.com/it/what-is/api/>

2. Come Utente registrato, in modo da accedere ad un condominio e diventare un condomino, voglio poter fare richiesta di accesso ad un condominio da me scelto.
3. Come Utente registrato, in modo da avere un accesso rapido ad un condominio e diventare un condomino, voglio poter utilizzare un codice d'invito.
4. Come Utente registrato, in modo da avere accesso al mio account, voglio una pagina per effettuare il login.
5. Come Utente registrato, in modo da terminare la mia sessione, voglio poter effettuare il logout.
6. Come Utente registrato, in modo da cancellare l'iscrizione al sito, voglio poter eliminare il mio account.
7. Come Utente registrato, in modo da personalizzare il mio account, voglio poter cambiare la mia immagine di profilo.
8. Come Utente registrato, in modo da avere una sicurezza maggiore, voglio poter cambiare la mia password.
9. Come Utente registrato, in modo da poter ricevere informazioni alla casella di posta corretta, voglio poter cambiare la mia e-mail.
10. Come Utente registrato, in modo da avere un resoconto delle informazioni, voglio una pagina per vedere le informazioni del mio account.
11. Come Utente registrato, in modo da recuperare la mia password, voglio una pagina dedicata accessibile dalla pagina di login.
12. Come Utente registrato, in modo da aggiungere un condominio mancante e diventare Leader condominio, voglio poter creare un gruppo condominio.
13. Come Utente registrato, in modo da avere una overview generale dei condomini di cui faccio parte, amministro o di cui ho fatto richiesta di partecipazione, voglio una pagina che me li elenchi.

Mostriamo una serie di mock-up che spiegano meglio le user stories più significative:

The image displays two side-by-side mock-up screenshots of a web application interface for condominium management.

The left screenshot, titled "A Web Page" with the URL "https://condominio-organizer/login", shows a "LOG IN" section. It includes input fields for "E-mail" and "Password", an "Accedi" button, and a link for "...Oppure accedi con Google" with a corresponding "accedi con Google" button. At the bottom, there are links for "Torna alla home" and "Recupera la tua password".

The right screenshot, titled "A Web Page" with the URL "https://condominio-organizer/edit-user", shows an "Edit User" section. It features an "Avatar" field with a placeholder image and a "Scegli file" button. Below this are fields for "Username" (pre-filled with "Mario Rossi"), "E-mail", "Nuova Password", "Ripeti Password", and "Current Password" (highlighted in red). There is an "Aggiorna" button. At the bottom, there is a section titled "Cancella il mio account" with a "Sei insoddisfatto?" prompt, a "Cancella il mio account" button, and a "Logout" button.

Figura 5 Mockup relativa alle user story 4-5-6-7-8-9-10-11

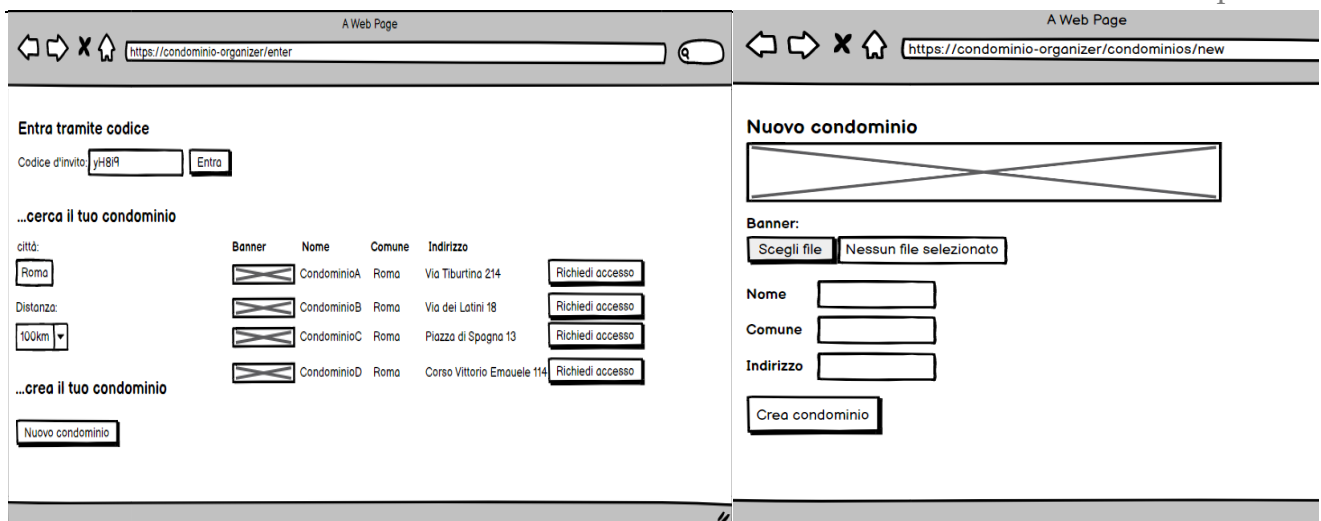


Figura 6 Mokup relativa alle user stories 1-2-3-12

### 2.2.3. Sezione condomino

Iniziamo ora a parlare dei *condòmini*, utenti che sono entrati in un condominio presente nel sito con una delle modalità descritte precedentemente. Essi possono iscriversi ad altri condomini, scrivere post e commenti nei gruppi a cui partecipano. Possono visualizzare e caricare file tramite Google Drive, su cui dispongono di una cartella privata con il loro username, all' interno di un'altra cartella legata al condominio di cui ha i permessi di lettura e scrittura sono del *Leader condominio*. Saranno sempre in contatto, per eventuali problemi, con l'amministratore tramite e-mail o la bacheca del condominio. Tutti gli eventi che devono essere notificati avviano automaticamente una comunicazione tramite e-mail con file ics, e se possibile, si genera un evento (con eventuali remainder, se posto ad una certa distanza temporale) sul Google calendar dell'utente.

Essendo dei semplici condòmini non hanno la possibilità di modificare o eliminare il condominio, possono solo eliminare post e commenti che hanno scritto, ma non quelli degli altri utenti, inoltre non possono creare eventi, ma solo visualizzarli.

Hanno accesso alla lista dei membri del condominio così da vedere chi sono gli amministratori e gli altri condòmini, ma anche qui è limitato nelle interazioni che può effettuare.

Di seguito la lista delle funzioni richieste da un *condomino*:

1. Come condomino, in modo da avere una gestione dei miei gruppi, voglio poter uscire da un gruppo condominio.
2. Come condomino, in modo da essere notificato da eventi, che includono pagamenti e i loro remainder, assemblee di condominio e altri eventi "generici", voglio ricevere avvisi tramite e-mail, con file evento ICS e creazione, se autorizzato, di un evento sul mio Google Calendar.

3. Come condomino, in modo da vedere le notizie rapidamente, voglio vedere la bacheca del gruppo condominio
4. Come condomino, in modo da sapere a chi rivolgermi in caso di necessità, voglio poter vedere chi sono i Leader condominio.
5. Come condomino, in modo da sapere chi sono gli altri condomini, voglio vedere chi sono i partecipanti del gruppo.
6. Come condomino, in modo da comunicare rapidamente con tutti gli altri condomini, voglio creare un post nella bacheca.
7. Come condomino, in modo da esprimere la mia opinione o dubbio su una comunicazione condominiale, voglio poter commentare un post nella bacheca.
8. Come condomino, in modo da vedere le opinioni degli altri condomini, voglio vedere i commenti di un determinato post.
9. Come condomino, in modo da gestire le mie comunicazioni con gli altri condomini, voglio poter eliminare un post che ho creato.
10. Come condomino, in modo da gestire le mie opinioni sulle comunicazioni condominiali, voglio poter eliminare un commento sotto un post che ho creato.
11. Come condomino, in modo da poter condividere documenti con i soli Leader condominio, voglio poter accedere ad una cartella Google Drive dedicata tramite l'applicazione.

Spieghiamo meglio l'ultima user story: ogni condomino ha una propria sottocartella nella cartella drive del condominio. La cartella e i permessi vengono creati quando un utente si unisce al condominio; nel caso l'accesso fosse stato effettuato senza account Google, l'utente può solo accedere alla cartella tramite link ma non condividere file. Il sito svolge il ruolo di interfaccia verso questa cartella Drive, permettendo di accedere direttamente ai contenuti di questa da esso.

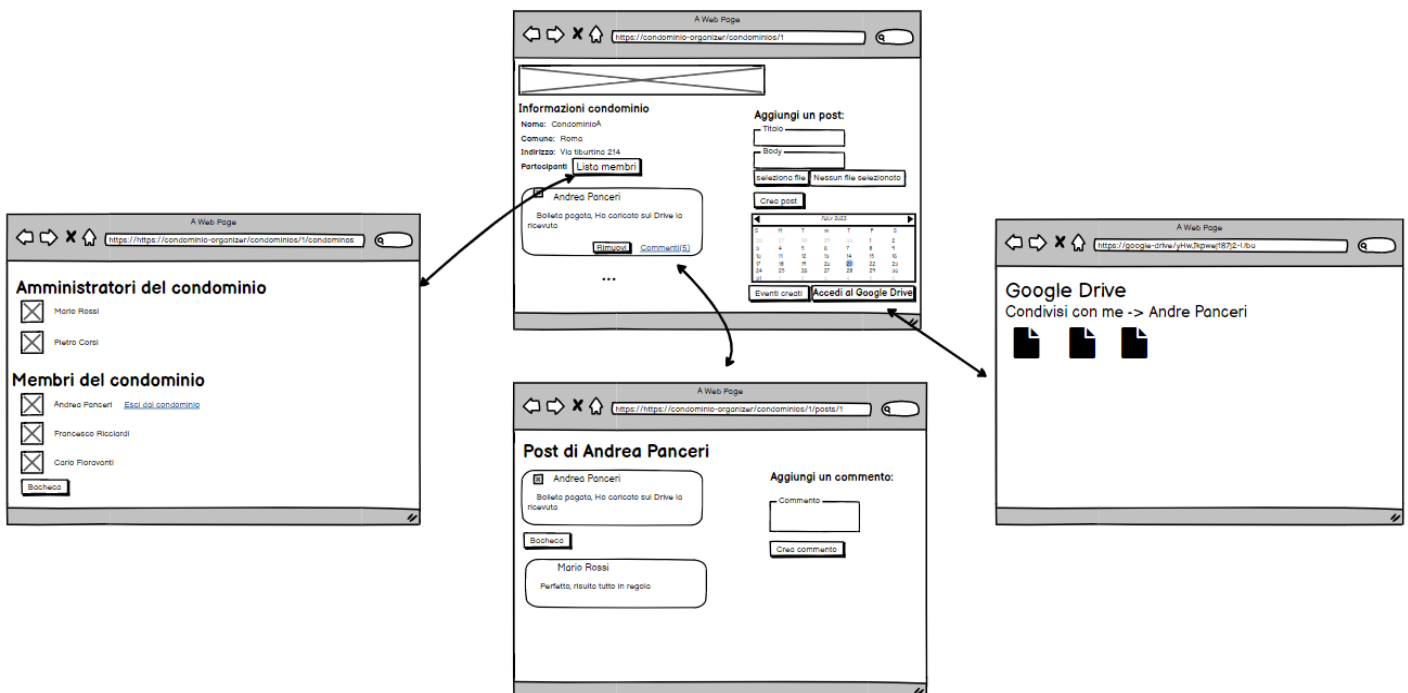


Figura 7 Mockup delle funzioni di un condomino

#### 2.2.4. Sezione Leader condominio

Il *Leader Condominio* attende a tutte le funzioni possibili del condomino, ma riceve anche dei privilegi extra relativi alla gestione del condominio di cui è Leader: può creare condomini, cancellare qualsiasi post e commento dal gruppo che gestisce, espellere i condòmini, condividere il codice d'accesso del condominio e accettare le richieste di partecipazione. Inoltre, può elevare a Leader Condominio un altro membro (Ma non ridurre i privilegi di altri Leader), modificare informazioni del gruppo, impostare un "evento", generico o appartenente a una delle due categorie speciali "Pagamento" o "Assemblea". Tale evento attiva automaticamente l'invio di comunicazioni, relative allo stesso, a tutti i membri di un determinato condominio, tramite l'invio di e-mail contenenti file evento ICS autogenerati, e se autorizzati dall'utente, generazione di eventi sul Google Calendar dello stesso; può accedere ad ogni cartella del Google Drive del gruppo e inserirne file all'intero, ed inoltre contattare lo staff di amministrazione del sito tramite e-mail, per segnalare comportamenti scorretti. Un condomino può essere automaticamente promosso a Leader Condominio nel caso questo crea un gruppo condominio.

Analizziamo ora le caratteristiche di un evento:

**CATEGORIA:** ci sono tre categorie di evento disponibili generico, assemblea e pagamento. L'evento generico, se l'amministratore è autenticato con un account Google, genera semplicemente un evento sul Calendar, mentre con l'evento assemblea manderà le richieste di partecipazione a tutti i membri del condominio, con annesso un reminder un'ora prima della riunione. Infine, l'evento pagamento, oltre quanto detto per l'assemblea, ripeterà l'evento ogni settimana.

**TITOLO:** questo campo serve come descrizione per l'evento che è stato creato, e quindi far capire al ricevente dell'e-mail di cosa tratta.

**DATA EVENTO:** campo fondamentale che fissa la data dell'evento.

Il sistema informatico inoltre cambierà i permessi delle cartelle drive in ogni situazione che avviene durante l'uso da parte dell'amministratore del gruppo. Ad esempio, se un utente viene espulso verrà eliminata di conseguenza la sua cartella, o se un utente viene elevato a Leader, esso avrà i permessi di lettura e scrittura sulla cartella associata al condominio, ovviamente nel caso viene ceduto il ruolo da Leader vengono revocati i permessi sulla cartella. Inoltre, quando un amministratore scrive un post sulla bacheca può allegare un file e scegliere in quali cartelle, associate ai condòmini, verrà caricato.

Come fatto finora riassumiamo sotto forma di user stories le caratteristiche di un *Leader condominio*:

1. *Come Leader condominio, in modo da invitare altri partecipanti, voglio poter condividere il codice d'invito del gruppo.*
2. *Come Leader condominio, in modo da amministrare flessibilmente il condominio, voglio*

- elevare un condomino a Leader condominio.*
3. *Come Leader condominio, in modo da gestire eventuali problemi che necessitano accesso superiore, voglio avere un canale di comunicazione via e-mail con gli admin.*
  4. *Come Leader condominio, in modo da dimettermi dal ruolo di Leader, voglio poter cedere il ruolo Leader condominio ad un altro membro del gruppo.*
  5. *Come Leader condominio, in modo da comunicare con i condòmini, voglio mandare una mail a uno o tutti i membri del gruppo che amministro con Gmail o con altri provider.*
  6. *Come Leader condominio, in modo da comunicare con i condomini, voglio poter scrivere sulla bacheca.*
  7. *Come Leader condominio, in modo da poter moderare i contenuti condivisi sulla bacheca voglio poter eliminare un qualsiasi post nella bacheca.*
  8. *Come Leader condominio, in modo da gestire i membri dei gruppi, voglio poter rimuovere un membro del gruppo.*
  9. *Come Leader condominio, in modo da gestire le comunicazioni tra condòmini, voglio poter eliminare un qualsiasi commento da un post.*
  10. *Come Leader condominio, in modo da gestire i contenuti della cartella Drive condominiale, voglio poter caricare file nelle cartelle dei condòmini.*
  11. *Come Leader condominio, in modo da gestire le comunicazioni con i condòmini, voglio poter creare eventi "Assemblea" (con avvisi che vengono automaticamente inviati via e-mail a tutti i condòmini, quest'ultimi sono inoltre invitati all'evento su Google Calendar, con annesso file ICS).*
  12. *Come Leader condominio, in modo da avvertire i condòmini di eventuali pagamenti, voglio poter creare eventi "Pagamento" (con avvisi che vengono automaticamente inviati via e-mail a tutti i condòmini, con remainder a cadenza settimanale, con annesso file ICS).*
  13. *Come Leader condominio, in modo da personalizzare il condominio, voglio cambiare il banner del condominio.*
  14. *Come Leader condominio, in modo da permettere l'accesso al condominio ad un utente, voglio poter autorizzare gli utenti che hanno creato una richiesta di accesso al mio condominio, utilizzando una pagina apposita.*

Mostriamo uno schema che illustra cosa si aspetta un amministratore (Esempio: Mario Rossi) dall' applicazione:

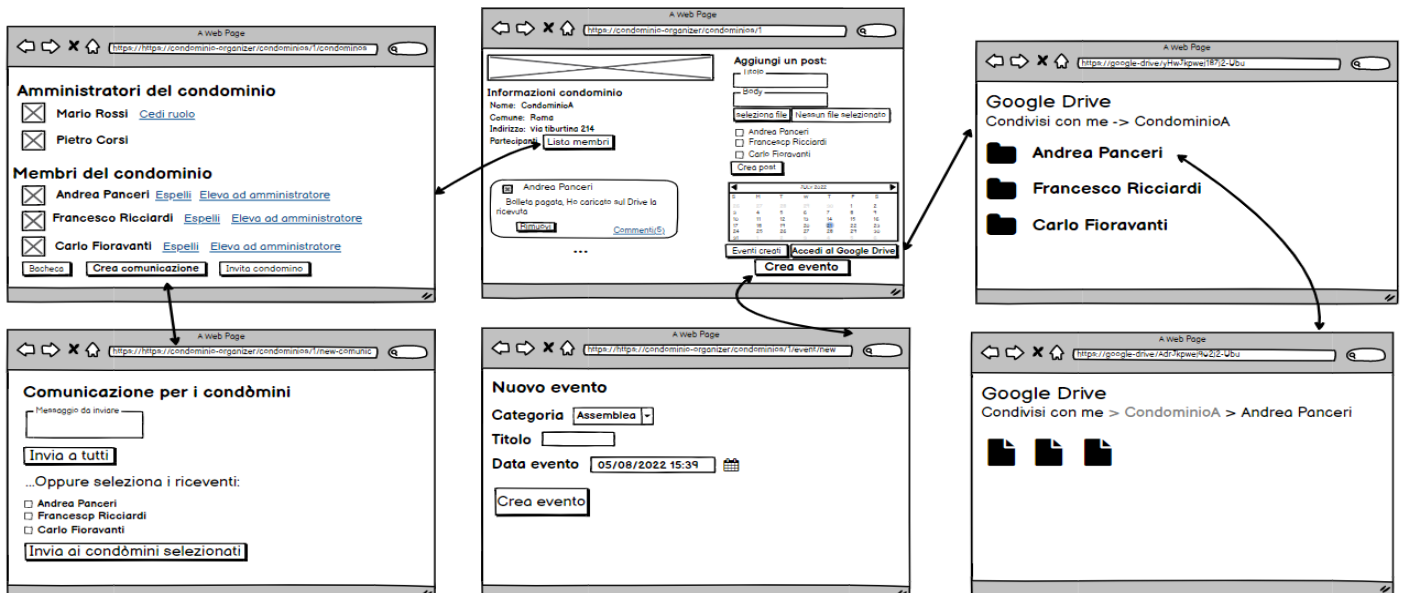


Figura 8 Mockup delle funzioni di un Leader condominio

### 2.2.5. Sezione Admin

L'*admin* è il ruolo riservato allo staff di amministrazione del sito. Ha accesso a tutte le informazioni, non sensibili, degli iscritti al sito ed inoltre può cancellare commenti e post da qualsiasi gruppo, espellere membri da un qualsiasi condominio, revocare e donare privilegi di amministrazione di un gruppo, cancellare gruppi non conformi, visualizzare informazioni e statistiche sul funzionamento del sito. Esiste un solo account Admin "root" inizialmente, da questo sarà poi possibile elevare altri account ad Admin.

Le user stories degli *admin* sono le seguenti:

1. Come admin, in modo da gestire gli utenti, voglio poter vedere tutti gli utenti del sito.
2. Come admin, in modo da gestire gli utenti, voglio poter eliminare un utente dal sito.
3. Come admin, in modo da gestire i condomini, voglio poter eliminare un condominio dal sito.
4. Come admin, in modo da gestire i condomini, voglio poter vedere tutti i condomini del sito.
5. Come admin, in modo da gestire i gruppi, voglio poter eliminare commenti o post da un gruppo.
6. Come admin, in modo da gestire i gruppi, voglio poter espellere un utente da un gruppo.
7. Come admin, in modo da gestire il sito, voglio poter visualizzare le informazioni del sito come il numero di utenti, il numero dei condomini ed altre ancora.

8. Come admin, in modo da gestire i gruppi, voglio rendere Leader Condominio un membro del gruppo.
9. Come admin, in modo da prevenire “abusi di potere” nei gruppi, voglio privare dei privilegi amministrativi un Leader condominio.
10. Come admin, in modo da avere una comunicazione più efficiente, voglio avere un canale di comunicazione via e-mail con i Leader Condominio.
11. Come admin, in modo da delegare le funzioni di amministrazione, voglio avere la possibilità di elevare altri utenti ad Admin.

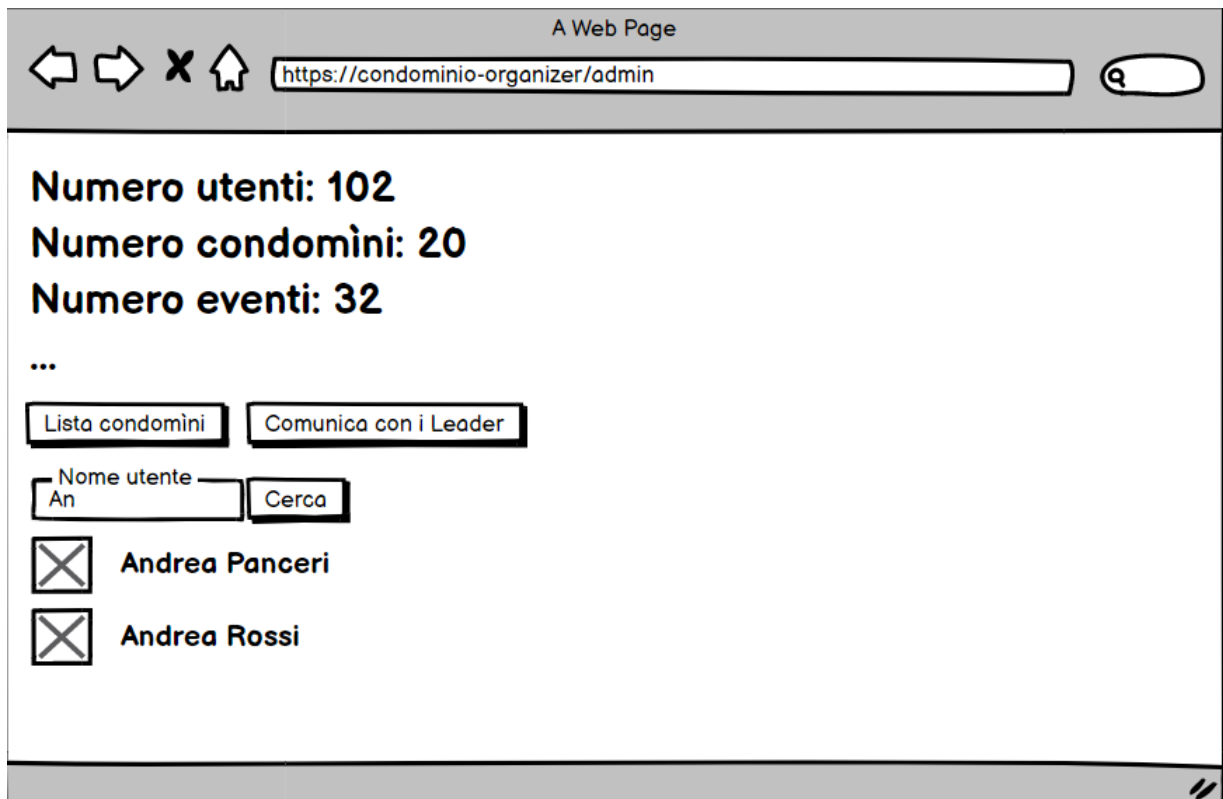


Figura 9 dashboard admin



### 3. Tecnologie e metodologie utilizzate

In questo capitolo l'attenzione verterà sulle tecnologie e metodologie adottate durante la progettazione e la realizzazione dell'applicazione informatica. Rispettivamente divideremo il capitolo in due parti: nella prima illustreremo le metodologie utilizzate, mentre nella seconda tratteremo le tecnologie e strumenti usati.

#### 3.1. I metodi agili

La nascita dei *metodi agili* è stata la conseguenza dei problemi riscontrati con quelli più antichi come *waterfal* e *spiral*. Tali metodi, rispetto a quelli più antichi, puntano l'attenzione sullo sviluppo del software stesso, meno sul design e la documentazione del software, permettendo tempi di produzione molto più rapidi.

I *metodi agili* per rispondere ai bisogni degli applicativi moderni, che subiscono molto cambiamenti durante il loro sviluppo, e hanno bisogno di tempi di risposta rapidi, danno allo sviluppatore dei principi da seguire, i quali riassumiamo qui:

##### **I principi sottostanti al metodo agile:**

1. La nostra massima priorità è soddisfare il cliente rilasciando software di valore, fin da subito e in maniera continua.
2. Accogliamo i cambiamenti nei requisiti, anche a stadi avanzati dello sviluppo. I processi agili sfruttano il cambiamento, a favore del vantaggio competitivo del cliente.
3. Consegniamo frequentemente software funzionante, con cadenza variabile da un paio di settimane a un paio di mesi, preferendo i periodi brevi.
4. Committenti e sviluppatori devono lavorare insieme quotidianamente per tutta la durata del progetto.
5. Fondiamo i progetti su individui motivati. Diamo loro l'ambiente e il supporto di cui hanno bisogno e confidiamo nella loro capacità di portare il lavoro a termine.
6. Una conversazione faccia a faccia è il modo più efficiente e più efficace per comunicare con il gruppo ed all'interno del gruppo.
7. Il software funzionante è il principale metro di misura di progresso.
8. I processi agili promuovono uno sviluppo sostenibile. Gli sponsor, gli sviluppatori e gli utenti dovrebbero essere in grado di mantenere indefinitamente un ritmo costante.
9. La continua attenzione all'eccellenza tecnica e alla buona progettazione esalta l'agilità.
10. La semplicità - l'arte di massimizzare la quantità di lavoro non svolto - è essenziale.
11. Le architetture, i requisiti e la progettazione migliori emergono da gruppi che si auto-organizzano.

12. A intervalli regolari il gruppo riflette su come diventare più efficace, dopodiché regola e adatta il proprio comportamento di conseguenza.

(Beck K., Beedle M., et altr. 2001, <http://agilemanifesto.org/iso/it/principles.html>, ultima consultazione 21 luglio 2022).

### 3.1.1. La Programmazione Estrema e la sua implementazione

La programmazione estrema (XP, Extreme programming) è uno dei metodi agili più popolari, che ha portato grandi risultati in molte compagnie di diverse dimensioni e importanza.

XP enfatizza il lavoro di gruppo, in cui il manager, clienti e sviluppatori sono entità alla pari. Tale metodo organizza un ambiente semplice ma efficace che consente al gruppo di diventare molto efficace. Il gruppo si auto-organizza attorno al problema per risolverlo nel modo più efficiente possibile. La programmazione estrema ottimizza lo sviluppo di un applicativo informatico in cinque modi essenziali: comunicazione, semplicità, feedback, rispetto e coraggio. I programmatori “*estremi*” comunicano costantemente con i loro clienti e colleghi. Mantengono il design desiderato semplice e pulito. Ricevono riscontri testando il loro software a partire dal primo giorno. Forniscono l’applicativo ai clienti il prima possibile e implementano le modifiche secondo i suggerimenti. Ogni piccolo risultato rafforza il loro rispetto per i contributi unici di ogni singolo membro del gruppo. Con questa base i programmatori Extreme sono in grado di rispondere con coraggio alle mutevoli esigenze e tecnologie. (Don Wells, 2013)

Ciascuna funzionalità descritta nel capitolo precedente segue questo metodo. Infatti, per ciascuna funzionalità, sono state definite le user stories con i relativi mockup. Seguendo un processo incrementale dalle user stories e i mockup realizzati consultando l’utente finale, fino ad arrivare alla realizzazione dell’applicazione.

Nel corso dello sviluppo dell’applicativo è stata seguita rigorosamente la programmazione estrema, in particolare si è lavorato in gruppo di quattro persone. La collaborazione ha portato numerosi vantaggi, infatti sono stati risolti problemi che se approcciati singolarmente avrebbero sicuramente rallentato di molto il lavoro. Inoltre, si è fatto uso di strumenti di *version control*, in particolare si è utilizzato GitHub, che ha permesso a tutti i membri del gruppo di lavorare sullo stesso codice sorgente e rimanere aggiornati su ogni cambiamento di esso. Particolarmente utile è stata l’integrazione di *visual studio code* con GitHub, questo ha facilitato ancora di più lo sviluppo del *software* e la coordinazione del gruppo.

Per quanto riguarda il mio contributo nel gruppo, mi sono occupato dell'integrazione delle api di Google Drive, Google Calendar e Gmail, con la relativa gestione di tutte le operazioni di *back-end*. Inoltre, con la collaborazione di un altro membro, ho realizzato tutti i test che verranno approfonditi nell'ultimo capitolo. Nella parte grafica il mio lavoro è stato quello di realizzare la pagina per accedere e cercare condomini, e relativa bacheca. In aggiunta, ho lavorato personalmente alla gestione dei ruoli e delle funzionalità di ognuno.



Figura 10 Schema programmazione estrema

### 3.2. Panoramica sul framework *Ruby on Rails*

Nella realizzazione dell'applicazione informatica si è fatto uso di uno dei framework più famosi nell'ambito dello sviluppo web, cioè Ruby on Rails.

La definizione del dizionario di framework è: la struttura di base all'origine di un sistema, non lontano dal concetto che si utilizza nello sviluppo di un software. Dà il supporto e la "guida" di base dell'applicativo in sviluppo. Un framework nella programmazione è uno strumento che fornisce componenti o soluzioni già pronte che sono personalizzate per accelerare lo sviluppo di un software. Solitamente nella pratica è una raccolta di classi di oggetti, le quali possono essere estese aggiungendone altre che ereditano le operazioni delle classi fornite dal framework.

Per spiegare cos'è Ruby on Rails ci avvaleremo della documentazione ufficiale del framework:

"Rails è un framework per lo sviluppo di applicazioni web scritto nel linguaggio di programmazione Ruby. È progettato per semplificare la programmazione delle applicazioni Web fornendo allo sviluppatore ciò di cui ha bisogno per iniziare. Ti consente di scrivere meno codice, rispetto ad altri framework o linguaggi. Rails parte dal

presupposto che esiste sempre un modo "migliore" per fare le cose, ed è progettato per portarti verso quel modo. La filosofia Rails comprende due principali principi guida:

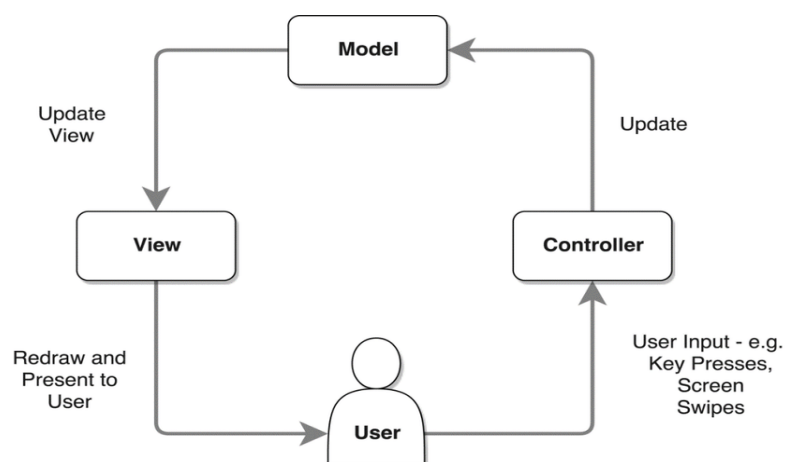
- Don't Repeat Yourself: DRY è un principio di sviluppo software che afferma che "Ogni frammento di conoscenza deve avere una rappresentazione unica, non ambigua e autorevole all'interno di un sistema". Non scrivendo le stesse informazioni più e più volte, il nostro codice è più mantenibile, più estensibile e con meno errori.
- Convention Over Configuration: Rails ha delle opinioni sul modo migliore per fare molte cose in un'applicazione Web e utilizza per impostazione predefinita questo insieme di convenzioni, piuttosto che richiedere di specificare sottigliezze attraverso infiniti file di configurazione."

(David Heinemeier Hansson, [https://guides.rubyonrails.org/getting\\_started.html](https://guides.rubyonrails.org/getting_started.html), ultima consultazione 22 luglio 2022)

L'architettura di Ruby on Rails si basa sul paradigma MVC (Model-View-Controller), che riprendendo l'architettura mostrata al capitolo 1, si posiziona nell' *application logic layer* in una struttura three-tier, e sui principi dell'architettura REST (REpresentational State Transfer), definiti per la progettazione di un sistema distribuito.

MVC si riferisce soprattutto alla realizzazione della GUI (Guest user interface), specifica che un'applicazione informatica consiste in un modello dei dati (Model), una presentazione delle informazioni (View) e un controllo di tali dati (Controller). Questa architettura richiede che ciascuno di questi deve essere in oggetti separati.

Tutto ciò porta semplificazioni importanti. I dati posso essere presentati in modi diversi e si possono fare interazioni separate su ciascuna presentazione. I dati se vengono modificati in una delle svariate presentazioni, vengono aggiornate tutte le altre.



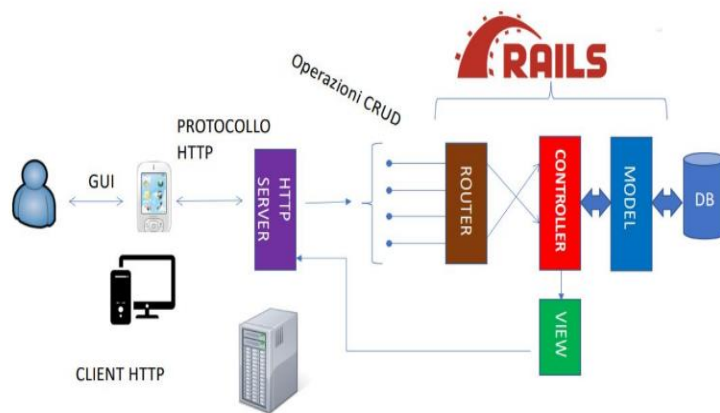


Figura 11 Rappresentazione grafica del pattern MVC

REST fornisce uno schema architetturale per software, spiega come un servizio deve essere costruito, e per estensione, come dovrebbero apparire le sue API:

- Rappresenta le varie entità manipolate da una web app come risorse.
- Costruisce *routes* (percorsi) così che ogni richiesta HTTP contenga tutte le informazioni necessarie a identificare sia la risorsa che l'azione da eseguire su di essa.

Solitamente, affiancato a REST troviamo l'acronimo già citato, CRUD, che si riferisce alle 4 funzioni che devono essere implementate per considerare completa un'applicazione REST, cioè: creazione, lettura, aggiornamento, cancellazione.

Ad ognuna delle funzioni corrisponde un'istruzione equivalente nel linguaggio SQL per interagire con una base di dati, rispettivamente: INSERT, SELECT, UPDATE, DELETE. Ad ogni operazione corrisponde un'equivalente istruzione in linguaggio SQL.

### 3.2.1. Pacchetti software aggiuntivi

Durante la realizzazione dell'applicazione sono stati utilizzati altri framework e servizi, tra cui troviamo:

- **Google-api-client.** Una delle "gemme"<sup>6</sup> fondamentali dell'applicazione realizzata, ufficialmente supportata da Google e costantemente aggiornata, permette di interagire in modo semplice e chiaro con tutti i servizi di Google, nel caso in questione con Gmail, Google Drive e Google Calendar.
- **Devise.** è una delle "gemme" più famose di Ruby, gestisce l'autenticazione dell'utente nel modo migliore e sicuro possibile, semplificando di molto il lavoro del programmatore, inoltre con pochi accorgimenti è estendibile per gestire

l'autenticazione con OAuth. Si può utilizzare anche per limitare l'accesso alle risorse solo agli utenti autenticati.

- **Bootstrap.** *Bootstrap* è uno dei framework lato client più utilizzati nell'ambito web. Offre ai programmatori degli strumenti per creare facilmente design di pagine web moderni, completamente conforme agli standard del CSS (Cascading Style Sheets). Forse la cosa più importante di questo framework è il fatto che permette agli elementi delle pagine web di essere completamente adattabili a qualsiasi dimensione dello schermo, fondamentale nei tempi moderni in cui la fruizione del web avviene tramite dispositivi di tutte le dimensioni.
- **Cancancan.** È una libreria per gestire le autorizzazioni d'accesso alle risorse nelle applicazioni realizzate con Ruby on Rails, la quale limita le risorse a cui l'utente può accedere. Tutti i permessi sono definiti in uno o più file *ability*, semplificando e organizzando in modo ottimale la gestione delle autorizzazioni.

---

<sup>6</sup> Le "gemme" in Ruby sono delle librerie, insieme di funzioni, che aiutano il programmatore nella realizzazione dell'applicazione desiderata.<sup>6</sup> Le "gemme" in Ruby sono delle librerie, insieme di funzioni, che aiutano il programmatore nella realizzazione dell'applicazione desiderata.

## 4. Analisi e progettazione concettuale

Nel corso di questo nuovo capitolo ci sarà la progettazione della base di dati, per fare ciò ne bisognerà definire la struttura, la composizione e il suo contenuto. Per raggiungere un risultato ottimale bisognerà seguire tre fasi di progettazione: concettuale, logica e fisica. Come esplicitato nel titolo questo capitolo verterà sulla progettazione concettuale.

### 4.1. Analisi dei requisiti

Per iniziare la progettazione della base di dati c'è bisogno della raccolta ed analisi dei requisiti. In questa fase si vuole descrivere in linguaggio naturale il contesto che si vuole modellare, sia i dati di interesse che le operazioni che si vogliono operare su di essi. Una volta che sono stati raccolti tutti i requisiti della base di dati, questi vanno analizzati per ottenere una descrizione chiara che permetta di isolare l'entità di interesse e le relazioni esistenti tra loro.

Diamo una specifica dei dati di interesse nel nostro sistema:

GLI UTENTI. L'applicazione deve salvare le informazioni che sono inerenti ad un utente. Come si evince da quanto detto nella trattazione precedente, emerge che per ogni utente è importante: il proprio username, l'e-mail che sarà identificativa, la password, l'immagine di profilo (Avatar dell'utente), se l'utente si è autenticato con *Oauth* e se è uno degli *Admin* del sito. Inoltre, interessa sapere se l'utente ha fatto delle richieste d'accesso per un condominio, e anche se è un partecipante di qualche condominio (o amministratore).

I CONDÒMINI. L'applicazione deve immagazzinare le informazioni sui vari partecipanti di un condominio, conseguentemente per ogni condomino interessa sapere a quale utente del sito si riferisce, e a quale condominio partecipa, e se eventualmente esso è Leader di quel gruppo. Inoltre, interessa sapere se il condomino ha scritto eventualmente qualche post o commento, e qual è la cartella che ha associata nel Drive del condominio, con i permessi su di essa.

I CONDOMINI. L'applicazione deve salvare le informazioni associate ai condomini, che sono: il nome, il comune in cui è situato con il suo indirizzo, e per permettere funzioni di calcolo delle distanze nel sito, anche latitudine e longitudine. Inoltre, è d'interesse sapere il codice per permettere l'accesso diretto al condominio. A questo si aggiunge il bisogno di conoscere chi sono i condòmini, le eventuali richieste d'accesso che sono

state effettuate da essi, i post e i commenti eventualmente scritti nella bacheca, gli eventi che potrebbero essere creati dagli amministratori e per ultimo la cartella Drive associata al condominio.

**LE RICHIESTE.** L'applicazione deve salvare le richieste fatte dagli utenti per entrare nei condomini. Quindi occorre conoscere l'utente che ha effettuato la richiesta e per quale condominio.

**GLI EVENTI.** L'applicazione deve immagazzinare le informazioni sugli eventi creati in un condominio, di cui interessa: il titolo, la data in cui avverrà e la categoria. Inoltre, bisogna capire a che condominio è legato l'evento, ed eventualmente se chi l'ha creato ha effettuato l'accesso con Google, l'id del Google Calendar. Esistono tre categorie di eventi cioè: generico, assemblea e pagamento, ognuno rappresentato dal parametro di categoria.

**I POSTS.** Il sito deve salvare le informazioni sui post scritti nelle varie bacheche, di cui è necessario sapere per ognuno il titolo e il contenuto (body), da che condomino è stato scritto e in quale condominio è stato inserito. Inoltre, bisogna vedere se ci sono dei commenti relativi a un post, ed inoltre può essere allegato un file su di esso.

**I COMMENTI.** L'applicazione deve immagazzinare le informazioni sui commenti scritti sotto ai posts nelle bacheche, di cui è d'interesse sapere il contenuto. Inoltre, per ogni commento, si vuole sapere da quale utente è stato scritto, in quale condominio e a quale post si riferisce.

**LE CARTELLE CONDOMINIO.** L'applicazione deve salvare le informazioni sulle cartelle Drive di ogni condominio, per fare ciò interessa: l'id della cartella nel Drive, il condominio a cui si riferisce, e se eventualmente ha delle cartelle utente al suo interno.

**LE CARTELLE DEGLI UTENTI.** L'applicazione deve salvare le informazioni sulle cartelle Drive di ogni condomino, per fare ciò interessa: l'id della cartella utente nel Drive, il condomino a cui si riferisce, e se eventualmente ha dei files al suo interno.

**I FILES.** L'applicazione deve salvare le informazioni sui files nelle cartelle Drive di ogni utente, per fare ciò interessa: l'id del file nel Drive e la cartella utente in cui si trova.



## 4.2. La progettazione concettuale

La progettazione concettuale ha come fine quello di dare forma ai requisiti raccolti della base di dati in uno schema concettuale, che deve essere indipendente dal tipo di RDBMS che verrà scelto. Solitamente si utilizza il modello ER (Entità-relazione) durante questa fase progettuale per definire gli aspetti delle entità del sistema, cioè i dati d'interesse. Esso è un diagramma che descrive le entità da modellare (ad esempio, nel nostro caso di studio, i condomini e le altre citate precedentemente), gli attributi delle entità (ad esempio, username per l'utente), le relazioni che connettono le entità (ad esempio la relazione "ha effettuato" che lega utente alla richiesta), e le cardinalità delle relazioni (nel nostro caso un condomino afferisce ad esattamente un condominio).

## 4.3. Lo schema Entità-Relazione

Grazie alla raccolta ed analisi dei requisiti effettuata precedentemente si è riusciti ad individuare dei dati fondamentali nel sistema che stiamo investigando. Creando uno schema di partenza che contiene tutte le informazioni di base, integrandolo e approfondendolo, si giunge al risultato finale, illustrato nella Figura 12.

Per ottenere un diagramma ER completo va integrato con una documentazione adeguata, che permetta di comprendere alla perfezione ogni componente al suo interno ed esprima il concetto legato al nome di ogni entità e relazione.

Oltre al diagramma ER, lo schema concettuale è costituito dal così detto dizionario dei dati, in cui troviamo le tabelle delle entità, delle relazioni, degli attributi ed eventualmente dei vincoli esterni.

Nella Tabella 1 troviamo l'elenco delle entità con la descrizione per ognuna, gli attributi e gli identificatori.

Nella Tabella 2 troviamo l'elenco delle relazioni con la descrizione per ciascuna di esse, le componenti che le costituiscono, gli attributi e gli identificatori.

Nella Tabella 3 troviamo l'elenco degli attributi con l'entità/relazione a cui si riferiscono, il dominio e una descrizione.

Nella Tabella 4 troviamo per ogni relazione la spiegazione dei vincoli di cardinalità.

Per prima cosa riportiamo in figura lo schema concettuale, e poi di seguito le tabelle appena citate:



Entità	Descrizione	Attributi	Identificatori
<b>Utente</b>	Utente che si è registrato nel sito.	Username, Avatar, E-mail, Password, From_oauth	{E-mail}
<b>Admin</b>	Uno degli utenti registrati che ha i privilegi di amministrazione.		
<b>Richiesta</b>	Rappresentazione della richiesta d'accesso a un condominio.		
<b>Condomino</b>	Un membro iscritto a un gruppo condominio.	Id_permesso	
<b>Condominio</b>	Rappresentazione di un condominio registrato nel sito.	Nome, Comune, Banner, Indirizzo, Codice, Latitudine, Longitudine	{Codice}
<b>Leader Condominio</b>	Uno dei condòmini di un condominio, che ne è l'amministratore.		
<b>Post</b>	Messaggio scritto da uno dei membri di un condominio.	Titolo, Body	
<b>Commento</b>	Commento scritto da un condomino sotto un post.	Body	
<b>Evento</b>	Rappresentazione di un evento fissato in un condominio	Titolo, Data, Id_calendar	
<b>Generico</b>	Evento generico che determina l'invio di e-mail con file ics.		
<b>Assemblea</b>	Evento assemblea che genera invio di e-mail con richiesta di partecipazione e file ics.		
<b>Pagamento</b>	Evento pagamento che si comporta come l'evento assemblea, ma con un remainder a cadenza settimanale.		
<b>Cartella condominio</b>	Rappresentazione della cartella Drive di un condominio.	Id_cartella	{Id_cartella}
<b>Cartella utente</b>	Rappresentazione della cartella Drive di un condomino.	Id_cartella	{Id_cartella}
<b>File</b>	File che è stato allegato in un post.	Id_file	{Id_file}

Tabella 1 Dizionario dei dati: entità

Relazione	Descrizione	Componenti	Attributi	Identificatori
<b>Effettua</b>	Ogni utente può fare delle richieste d'accesso per un condominio.	Utente, Richiesta		
<b>È un</b>	Ogni utente può entrare in un condominio e diventare un condomino.	Utente, Condomino		
<b>Indirizzata</b>	Ogni condominio può ricevere delle richieste d'accesso.	Richiesta, Condominio		
<b>Appartiene</b>	Ogni condominio ha almeno un condomino, almeno un Leader.	Condomino, Condominio		
<b>Scrive</b>	Ogni condomino può scrivere dei posts.	Condomino, Post		
<b>Contiene</b>	Sotto a ogni post possono essere pubblicati dei commenti.	Post, Commento		
<b>Raccoglie</b>	Le bacheche dei condomini possono raccogliere dei posts.	Condominio, Post		
<b>Aggiunge</b>	Ogni condomino può aggiungere un commento sotto un post.	Condomino, Commento		
<b>Legato</b>	In ogni post può essere allegato un file.	Post, File		
<b>Ha cartella</b>	Ogni condomino ha una cartella Drive nel condominio.	Condomino, Cartella utente		
<b>Afferisce</b>	Ogni file afferisce a una specifica cartella Drive di un condomino.	File, Cartella utente		
<b>Ha sottocartella</b>	Le cartelle Drive condominio possono contenere delle sottocartelle per gli eventuali condomini.	Cartella condominio, Cartella utente		
<b>Associazione</b>	Ogni condominio ha associata una cartella nel Drive.	Condominio, Cartella condominio		
<b>Gestisce</b>	In ogni condominio possono essere fissati degli eventi	Condominio, Evento		

Tabella 2 Dizionario dei dati: relazioni

Attributo	Entità/Relazione	Dominio	Descrizione
<b>Username</b>	Utente	String	Username dell'utente.
<b>Password</b>	Utente	String	Password dell'utente.
<b>E-mail</b>	Utente	String	E-mail dell'utente
<b>Avatar</b>	Utente	BLOB	Foto profilo dell'utente
<b>From_oauth</b>	Utente	Boolean	Dice se l'utente ha effettuato l'accesso con Google.
<b>Id_permesso</b>	Condominio	String	Codice che rappresenta il permesso sulla cartella Drive.
<b>Nome</b>	Condominio	String	Nome del condominio.
<b>Comune</b>	Condominio	String	Comune in cui è situato il condominio.
<b>Indirizzo</b>	Condominio	String	Indirizzo in cui si trova il condominio.
<b>Latitudine</b>	Condominio	Float	Latitudine della sede del condominio.
<b>Longitudine</b>	Condominio	Float	Longitudine della sede del condominio.
<b>Codice</b>	Condominio	String	Codice per accedere ad un condominio.
<b>Banner</b>	Condominio	BLOB	Copertina del condominio.
<b>Titolo</b>	Post	Text	Titolo di un post.
<b>Body</b>	Post	Text	Contenuto di un post.
<b>Body</b>	Commento	Text	Contenuto di un commento.
<b>Titolo</b>	Evento	Text	Titolo di un evento.
<b>Data</b>	Evento	Date	Data in cui è stato fissato un evento.
<b>Id_calendar</b>	Evento	String	Codice che rappresenta l'evento nel Google Calendar.
<b>Id_cartella</b>	Cartella condominio	String	Codice che rappresenta la cartella di un condominio nel Google Drive del sito.
<b>Id_cartella</b>	Cartella utente	String	Codice che rappresenta la cartella di un utente nel Google Drive del sito.
<b>Id_file</b>	File	String	Codice che rappresenta un file nel Google Drive del sito.

Tabella 3 Dizionario dei dati: attributi

Relazione	Componenti	Vincoli di cardinalità
<b>Effettua</b>	Utente	Ogni utente può effettuare zero o più richieste.
	Richiesta	Ogni richiesta può essere effettuata da uno ed un solo utente.
<b>È un</b>	Utente	Ogni utente può diventare zero o più volte condomino.
	Condomino	Ogni condomino è uno ed un solo utente.
<b>Indirizzata</b>	Richiesta	Ogni richiesta è indirizzata a uno ed un solo condominio.
	Condominio	Ogni condominio può ricevere zero o più richieste.
<b>Appartiene</b>	Condomino	Ogni condomino appartiene a uno ed un solo condominio.
	Condominio	Ogni condominio può avere uno o più condomini.
<b>Scrive</b>	Condomino	Ogni condomino può scrivere zero o più posts.
	Post	Ogni post è scritto da uno ed un solo condomino
<b>Contiene</b>	Post	Ogni post può contenere zero o più commenti.
	Commento	Ogni commento è legato a uno e un solo post.
<b>Raccoglie</b>	Condominio	Ogni condominio può raccogliere zero o più posts.
	Post	Ogni post è legato a uno ed un solo condominio.
<b>Aggiunge</b>	Condomino	Ogni condomino può aggiungere zero o più commenti.
	Commento	Ogni commento è aggiunto da uno e un solo condomino.

Relazione	Componenti	Vincoli di cardinalità
<b>Legato</b>	Post	Ogni post è legato a zero o un file.
	File	Ogni file è legato a uno ed un solo post.
<b>Ha cartella</b>	Condominio	Ogni condominio ha una ed una sola cartella utente.
	Cartella utente	Ogni cartella utente è collegata ad uno ed un solo condominio.
<b>Afferisce</b>	File	Ogni file afferisce a una ed una sola cartella utente.
	Cartella utente	Ogni cartella utente può contenere zero o più file.
<b>Ha sottocartella</b>	Cartella condominio	Ogni cartella condominio può contenere zero o più cartelle utente.
	Cartella utente	Ogni cartella utente fa parte di una ed una sola cartella condominio.
<b>Associazione</b>	Condominio	Ogni condominio ha una ed una sola cartella condominio.
	Cartella condominio	Ogni cartella condominio è collegata a uno ed un solo condominio.
<b>Gestisce</b>	Condominio	Ogni condominio deve gestire zero o più eventi.
	Evento	Ogni evento è gestito da uno ed un solo condominio.

Tabella 4 Dizionario dei dati: Vincoli di cardinalità

#### 4.4. I Vincoli d'integrità esterni

Il modello Entità-Relazione ha un'espressività limitata, con il solo schema concettuale, non è possibile esprimere tutte le informazioni e vincoli di una realtà d'interesse. Per questo nella documentazione appena fornita, va aggiunta un'altra tabella, che esprime in

linguaggio naturale i vincoli non esprimibili dal modello ER. Ad esempio, nella nostra realtà d'interesse non è esprimibile il fatto che l'immagine del profilo dell'utente deve essere nel formato PNG o JPEG. La tabella Vincoli d'integrità esterni elenca questi tipi di vincoli non esprimibili.

<b>Vincoli d'integrità esterni</b>	
(1)	Per ogni istanza di utente il campo avatar deve essere nel formato JPEG o PNG.
(2)	Per ogni istanza di condominio il campo banner deve essere nel formato JPEG o PNG.
(3)	Per ogni istanza di condominio il campo longitudine non può essere definito se non è definito il campo latitudine, e viceversa.
(4)	Per ogni istanza di evento il campo data deve essere maggiore della data in cui viene creato.
(5)	Per ogni istanza di utente il campo password deve essere una stringa di almeno otto caratteri.
(6)	Per ogni istanza di utente il campo username deve essere una stringa di almeno tre caratteri.
(7)	Per ogni istanza di condominio il campo nome deve essere una stringa di almeno tre caratteri.
(8)	Per ogni istanza di post il campo body deve essere una stringa di almeno tre caratteri.
(9)	Per ogni istanza di post il campo titolo deve essere una stringa di almeno tre caratteri.
(10)	Per ogni istanza di commento il campo body deve essere una stringa di almeno tre caratteri.
(11)	Per ogni istanza di evento il campo titolo deve essere una stringa di almeno tre caratteri.

Tabella 5 Dizionario dei dati: Vincoli esterni



## 5. Progettazione del sistema

Dalla documentazione ottenuta nel capitolo precedente, continuiamo la progettazione del sistema. Ad ogni entità che troviamo nel modello ER sarà associata una tabella nel modello relazionale. Visto che utilizziamo il pattern di design MVC, ad ogni tabella verrà legato un modello che eseguirà operazioni CRUD, seguendo le indicazioni del relativo controller, a seguito di un'iterazione di un utente nella view.

Dividiamo la trattazione del capitolo in due parti: nella prima ci sarà la progettazione logica, mentre nella seconda dallo schema logico ottenuto ci sarà la traduzione nello schema relazionale.

### 5.1. Progettazione logica del Data Layer

Durante la progettazione logica c'è la traduzione dello schema concettuale nello schema logico, come prima questo schema è indipendente dal RDBMS scelto. L'obiettivo di tale fase è quello di realizzare uno schema logico fedele ed efficiente alle informazioni elaborate nel capitolo precedente. Per far ciò bisogna ristrutturare il modello ER e poi tradurlo nello schema logico. Questa traduzione deve essere guidata dall'obiettivo di ottimizzare le prestazioni.

#### 5.1.1. Ristrutturazione dello schema ER

La ristrutturazione dello schema ER avviene in sette passi:

1. Preparazione dello schema ER: vengono ridisegnato il diagramma aggiungendo tutti i vincoli rivelanti che non erano stati esplicitati precedentemente.
2. Analisi delle ridondanze: si decide di eliminare eventuali ridondanze per avere una maggiore efficienza.
3. Eliminazione degli attributi multi-valore: attributi con cardinalità massima maggiore di uno vengono rimodellati, perché non direttamente traducibile nel modello relazionale.
4. Eliminazione degli attributi composti: come per quelli multi-valore, gli attributi composti devono essere divisi per essere traducibili.
5. Eliminazione delle ISA e delle generalizzazioni.
6. Scelta degli identificatori principali di entità e relazioni.
7. Specifica degli ulteriori vincoli esterni.

Effettuiamo ora una trattazione scritta, senza ridisegnare lo schema per ogni passo, illustrando direttamente il prodotto finale al termine. I primi quattro passi non aggiungono modifiche al nostro schema, poiché non si sono attributi multi-valore o composti e non si evince nessuna ridondanza.

Nel passo cinque bisogna eliminare le ISA che sussistono tra utente ed admin, e poi tra condomino e Leader condominio. Inoltre, la generalizzazione che sussiste tra evento, evento generico, evento pagamento e evento assemblea va sostituita con tre relazioni binarie tra evento ed evento generico, evento ed evento assemblea e per finire tra evento ed evento pagamento. Tale semplificazione introduce un nuovo vincolo esterno:

VINCOLO ESTERNO: Ogni istanza di evento partecipa esattamente ad una delle ISA-E-G, ISA-E-A ed ISA-E-P.

Il passo sei è quello di definire gli identificatori principali per ciascuna entità e relazione dello schema ER. A tal proposito, si aggiungono gli attributi ID (Identificatore) a tutte le entità.

Lo schema seguente mostra lo schema concettuale ristrutturato, e invece la figura successiva mostra lo schema concettuale tradotto.





## 5.1.2. Vincoli esterni

Seguendo il punto sette della ristrutturazione dobbiamo aggiornare i vincoli esterni, nel nostro caso abbiamo dovuto sostituire la generalizzazione che sussisteva tra evento, e le sue tre categorie. La seguente tabella riassume i vincoli che dovranno essere implementati nella realizzazione della base di dati:

<b>Vincoli esterni</b>	
(1)	Per ogni istanza di utente il campo avatar deve essere nel formato JPEG o PNG.
(2)	Per ogni istanza di condominio il campo banner deve essere nel formato JPEG o PNG.
(3)	Per ogni istanza di condominio il campo longitudine non può essere definito se non è definito il campo latitudine, e viceversa.
(4)	Per ogni istanza di evento il campo data deve essere maggiore della data in cui viene creato.
(5)	Per ogni istanza di utente il campo password deve essere una stringa di almeno otto caratteri.
(6)	Per ogni istanza di utente il campo username deve essere una stringa di almeno tre caratteri.
(7)	Per ogni istanza di condominio il campo nome deve essere una stringa di almeno tre caratteri.
(8)	Per ogni istanza di post il campo body deve essere una stringa di almeno tre caratteri.
(9)	Per ogni istanza di post il campo titolo deve essere una stringa di almeno tre caratteri.
(10)	Per ogni istanza di commento il campo body deve essere una stringa di almeno tre caratteri.
(11)	Per ogni istanza di evento il campo titolo deve essere una stringa di almeno tre caratteri.
(12)	Ogni istanza di evento partecipa esattamente ad una delle ISA-E-G, ISA-E-A ed ISA-E-P
(13)	Data un'istanza di condominio, essa deve partecipare alla relazione partecipa con un Leader Condominio almeno una volta.

Tabella 6 Vincoli esterni

## 5.2. Architettura dell'applicativo software

Come già detto nei primi capitoli, le applicazioni informatiche realizzate con il framework Ruby on Rails fanno uso del pattern MVC che permette di organizzare l'applicazione web in una struttura modulare e molto più efficiente rispetto allo sviluppo normale. L'applicativo viene diviso in tre componenti: modello, vista e controller, permettendo di dividere il codice che gestisce la parte grafica, la logica e la gestione dei dati. Questa architettura consente di aggiungere facilmente nuove componenti al codice, e di lavorare su diverse parti dell'applicazione senza dover comprendere o modificare l'intera logica.

Il modello mantiene la relazione tra gli oggetti e i dati nel database. Inoltre, gestisce la validazione, l'associazione, la transazione ed altre operazioni tra i dati del database. Rails usa il modulo *ActiveRecord* per gestire la logica e la comunicazione con il database.

Qui sotto troviamo un esempio di modello:

```
app > models >  condominium.rb
1 class Condominio < ApplicationRecord
2   before_validation :create_code
3   validates :comune, length: {minimum: 1, maximum: 25}, allow_blank: false
4   validates :nome, length: {minimum: 1, maximum: 25}, allow_blank: false
5   validates :indirizzo, allow_blank: false, format: { with: %r{(via|corso|viale|piazza|Via|Corso|Viale|Piazza)[ ]?[a-zA-Z](?![ ])|[a-zA-Z]){1,}[ ]?[0-9]{1,}\z}i ,message: 'invalido, formato: Via Tiburtina 214'}
6
7   def create_code
8     self.flat_code = ['a'..'z'], ['0'..'9'].shuffle[0,5].join
9   end
10
11   has_many :condominos, dependent: :destroy
12   has_many :events, dependent: :delete_all
13   has_many :users, through: :condominos
14
15   accepts_nested_attributes_for :condominos
16
17   has_one :gdrive_condo_item, dependent: :delete
18
19   has_many :posts, dependent: :destroy
20   has_many :comments
```

Figura 15 Modello Ruby on Rails

Possiamo vedere come il modello si occupa della validazione dei dati con *validates* e della relazione con gli altri modelli con *has\_many*.

La vista contiene solamente la logica della presentazione, il codice della vista deve compiere azioni che si riferiscono alla visualizzazione delle pagine dell'applicazione. In nessun modo la vista deve occuparsi della logica dell'applicazione e dell'interazione con i dati del database. Potremmo avere diverse viste inerenti allo stesso modello grazie a questa struttura. Come vediamo nel codice sottostante, nella vista c'è solo la visualizzazione del contenuto della pagina:

```

app > views > welcome > index.html.erb
21 <a id="navRegister" class="nav-link active" href="users/sign_up">Iscriviti</a>
22 </li>
23 </ul>
24 </div>
25 </div>
26 </nav>
27 <div class="container-fluid welcome-container">
28 <div class="welcome-banner">
29 <% flash.each do |key, message| %>
30 <% if message != true %>
31 <p class="welcome-messages alert alert-<%= key %>"><%= message %></p>
32 <% end %>
33 <% end %>
34 <br>
35 <h1 class="welcome-h1">Condominio Organizer</h1>
36 <p class="welcome-p">Un sito che punta a fornire un'applicazione unica dove aggregare tutte le informazioni di interesse
di un condominio, con un sistema di "gruppi condominio" gestiti da un Leader Condominio forniti di bacheca dove inserire
comunicazioni condominiali. Clicca su Iscriviti o Login, per usufruire di tutte le funzioni.</p>
37 </div>
38 </div>
39 <footer class="welcome-footer">
40 <%= link_to "Contact us", "contacts/new" %>
41 </footer>

```

Figura 16 Vista Ruby on Rails

Il controller invece deve gestire la logica dell'applicazione, facendo da intermezzo tra il presentation layer e il *resource management layer*, deve gestire le interazioni degli utenti con l'applicazione, relazionandosi con i modelli e mostrando la vista corretta all'utente. Come fatto prima vediamo un'implementazione del controller:

```

app > controllers > comments_controller.rb
1 class CommentsController < ApplicationController
2   before_action :set_comment, only: [:show, :destroy]
3   before_action :authenticate_user!
4
5   # GET /comments or /comments.json
6   def index
7     @comments = Comment.all
8   end
9
10  # GET /comments/1 or /comments/1.json
11  def show
12  end
13
14  # GET /comments/new
15  def new
16    @comment = Comment.new
17  end
18
19
20  # POST /comments or /comments.json
21  def create
22    authorize! :create, Comment
23    @post = Post.find(params[:post_id])
24    @comment = @post.comments.create(comment_params)
25    @comment.user_id = current_user.id
26    @comment.condominio_id = @post.condominio_id

```

Figura 17 Controller Ruby on Rails

### 5.2.1. Gestione dei dati

Nell'progettazione del nostro sistema vogliamo che i dati dell'applicazione informatica siano immagazzinati in un database relazionale (RDMS) per mantenere la coerenza del sistema. I database relazionali consentono di ottenere grandi risultati, assicurando che più istanze di un database contengano sempre gli stessi dati.

Come detto prima in Ruby on Rails il responsabile della rappresentazione dei dati e della loro logica è *Active Record*, il quale facilita la creazione e l'utilizzo di oggetti i cui dati richiedono l'archiviazione permanente in un database. "In Active Record, gli oggetti trasportano sia dati persistenti che comportamenti che operano su quei dati"<sup>7</sup>. Dato che le operazioni facilmente esprimibili in termini relazionali sono difficilmente codificabili in oggetti e viceversa, Ruby on Rails fa affidamento su un Object Relational Mapping, comunemente indicato come la sua abbreviazione ORM, che è una tecnica che collega gli oggetti ricchi di un'applicazione alle tabelle in un sistema di gestione di database relazionali. Utilizzando un ORM, le proprietà e le relazioni degli oggetti in un'applicazione possono essere facilmente archiviate e recuperate da un database senza scrivere lunghe istruzioni in SQL, ma con un codice di accesso al database meno complesso. Active Record ci offre diverse funzionalità, tra le quali troviamo:

- Rappresentazione dei modelli e i loro dati.
- Rappresentazione delle associazioni tra i modelli.
- Convalidazione dei modelli prima che vengano resi persistenti nel database.
- Esecuzione delle operazioni nel database in modo orientato agli oggetti.

Durante il normale funzionamento di un'applicazione web in Rails, gli oggetti possono essere creati, aggiornati e distrutti. Active Record fornisce dei controlli aggiuntivi in questo ciclo di vita dell'oggetto. Il framework mette a disposizione i *callbacks*, che sono metodi che vengono chiamati in determinati momenti del ciclo di vita di un oggetto. Con i callbacks è possibile scrivere codice che verrà eseguito ogni volta che un oggetto Active Record viene creato, salvato, aggiornato, eliminato, convalidato o caricato dal database. Questa funzionalità è utile nel nostro caso per esprimere i vincoli d'integrità descritti nel capitolo precedente.

Inoltre, Active Record permette di convalidare lo stato di un modello prima che venga scritto nel database, attraverso i *validations*. Esistono diversi metodi per controllare i modelli e per convalidare che un valore di un attributo non sia vuoto, sia univoco, non sia un duplicato, abbia un formato specifico ed altro ancora.

---

<sup>7</sup> [Active Record è descritto da Martin Fowler](#) nel suo libro *Patterns of Enterprise Application Architecture*



La figura sottostante mostra il diagramma delle classi definite nella nostra applicazione dove per ciascuna tabella nel database è stato definito un modello.

(David Heinemeier Hansson, [https://guides.rubyonrails.org/active\\_record\\_basics.html](https://guides.rubyonrails.org/active_record_basics.html), ultima consultazione 30 luglio 2022)

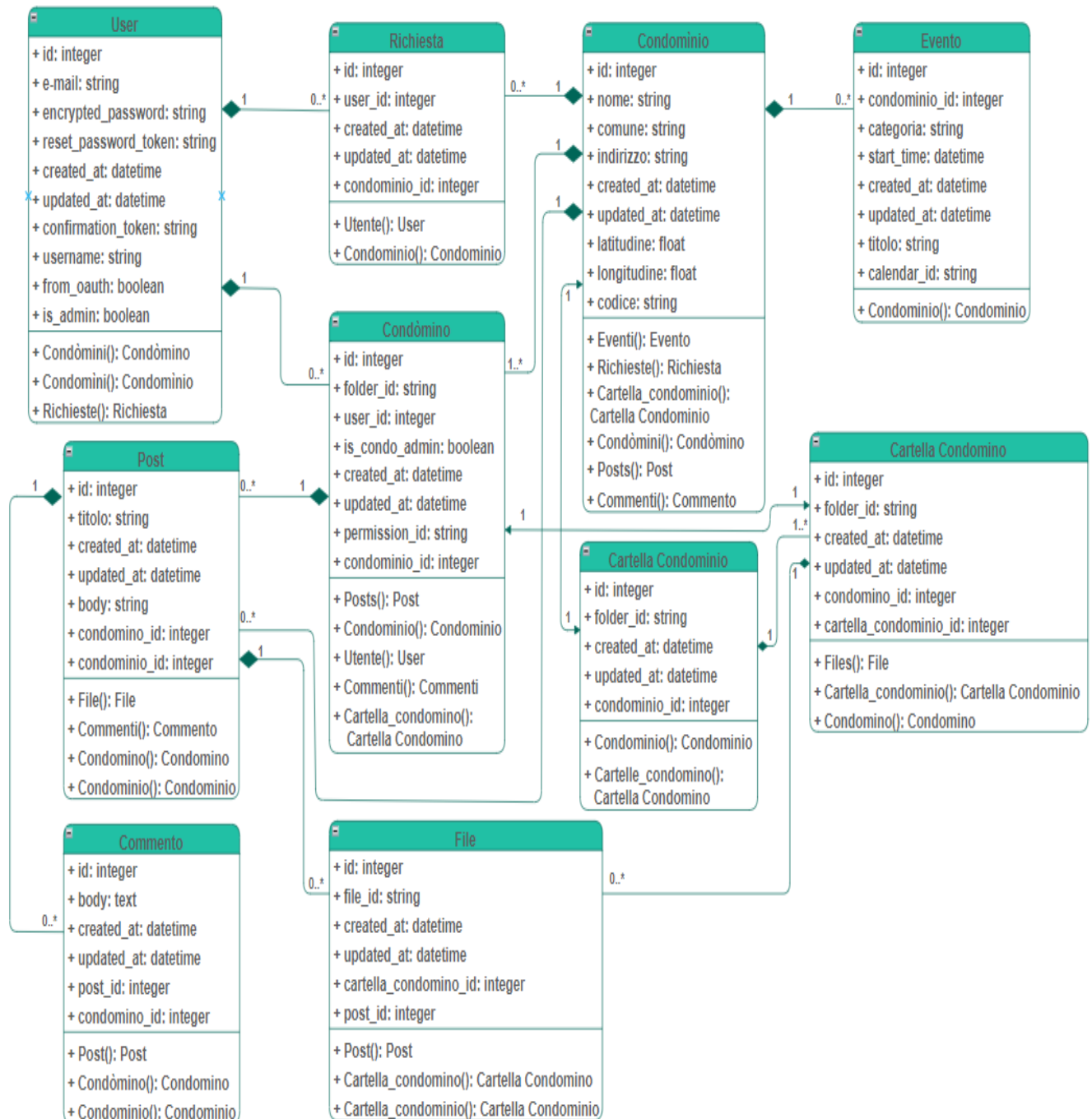


Figura 18 Diagramma a classi dei modelli

### 5.2.2. Logica dell'applicazione e presentazione

Continuando nella trattazione del pattern MVC, il controller ha la responsabilità di dare un senso alla richiesta dell'utente e produrre l'output corretto. Ruby on Rails utilizza un Action Controller per fare la maggior parte del lavoro di base, lasciando al programmatore meno lavoro possibile.

In linea generale, il controller riceverà richieste, elaborerà o salverà i dati dal modello, e userà le viste per creare un output. Può essere pensato come un intermediario tra i modelli e le viste, facendo in modo di rendere disponibili i dati del modello alla vista. Un controller è una classe Ruby che eredita le sue caratteristiche dall' *ApplicationController*, e ha dei metodi come qualsiasi altra classe. Quando l'applicazione riceve una richiesta, il *routing* determinerà quale controller e azione da lanciare, in seguito nascondendo il processo al programmatore, Rails creerà un'istanza del controller e lancerà il metodo con lo stesso nome dell'azione. Ovviamente Rails è in grado di gestire qualsiasi tipologia di errore, in caso l'azione desiderata non esiste nel controller.

Mentre il responsabile della compilazione della risposta è *ActionView*. I *Templates* di *ActionView* sono scritti utilizzando codice Ruby incorporati in tag HTML, questo facilita di molto la creazione dei layout delle pagine e soprattutto la gestione dinamica di esse. Per ogni controller c'è una cartella nel percorso *app/views*, che contiene i file *template*, che costituiscono le viste associate a quel controller, tali file vengono utilizzati per mostrare l'output di ciascuna azione di un controller.<sup>8</sup>

Di seguito verranno esibite le viste come elementi del diagramma delle classi, ma nella realtà esse corrispondono a strutture molto più articolate. Inoltre, verrà suddiviso il diagramma in diverse sezioni e analizzati gli aspetti d'interesse per ognuna, per non appesantire la trattazione verranno mostrate solo le sezioni più rilevanti dell'applicazione.

Nella la figura sottostante viene mostrato il diagramma delle classi dei controller inerenti ai condomini e ai condòmini.

Possiamo notare come ogni controller abbia associato diverse viste, ognuna delle quali si riferisce a uno dei metodi del controller, inoltre si evince che ogni controller deve accedere ai dati dei modelli per poter realizzare le viste correttamente.

---

<sup>8</sup> David Heinemeier Hansson, [https://guides.rubyonrails.org/action\\_controller\\_overview.html#what-does-a-controller-do-questionmark](https://guides.rubyonrails.org/action_controller_overview.html#what-does-a-controller-do-questionmark), ultima consultazione 14 agosto 202

Un aspetto particolare che viene mostrato dal diagramma sottostante è l'uso di metodi "custom", cioè metodi che non sono direttamente forniti da Ruby on Rails alla creazione del controller come: index, show, new, create, edit, update e destroy. Qui, ad esempio, notiamo create\_comunicazione\_for\_admin come metodo non standard.

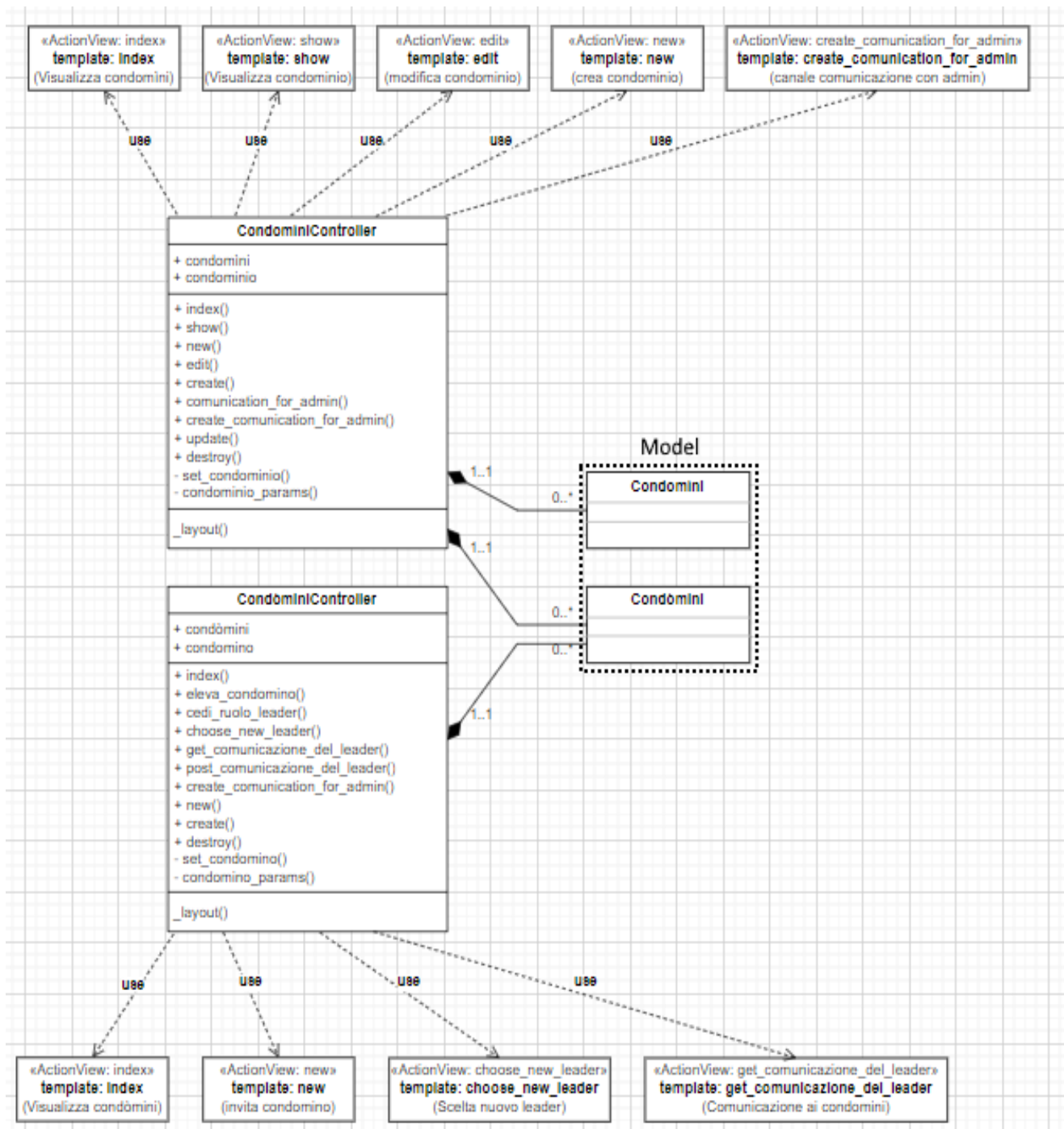


Figura 19 Diagramma delle classi Condomini e Condomini

La figura sottostante mostra il diagramma delle classi per il controller dell'admin, che è legato solo a due viste. La prima vista è la dashboard dell'admin dove troviamo informazioni generali sul sito, ma anche la possibilità di interagire con tutti gli utenti registrati del sito e di accedere alle homepage di tutti i condomini. Invece la seconda vista riguarda il canale di comunicazione tramite e-mail tra admin e leader condominio.

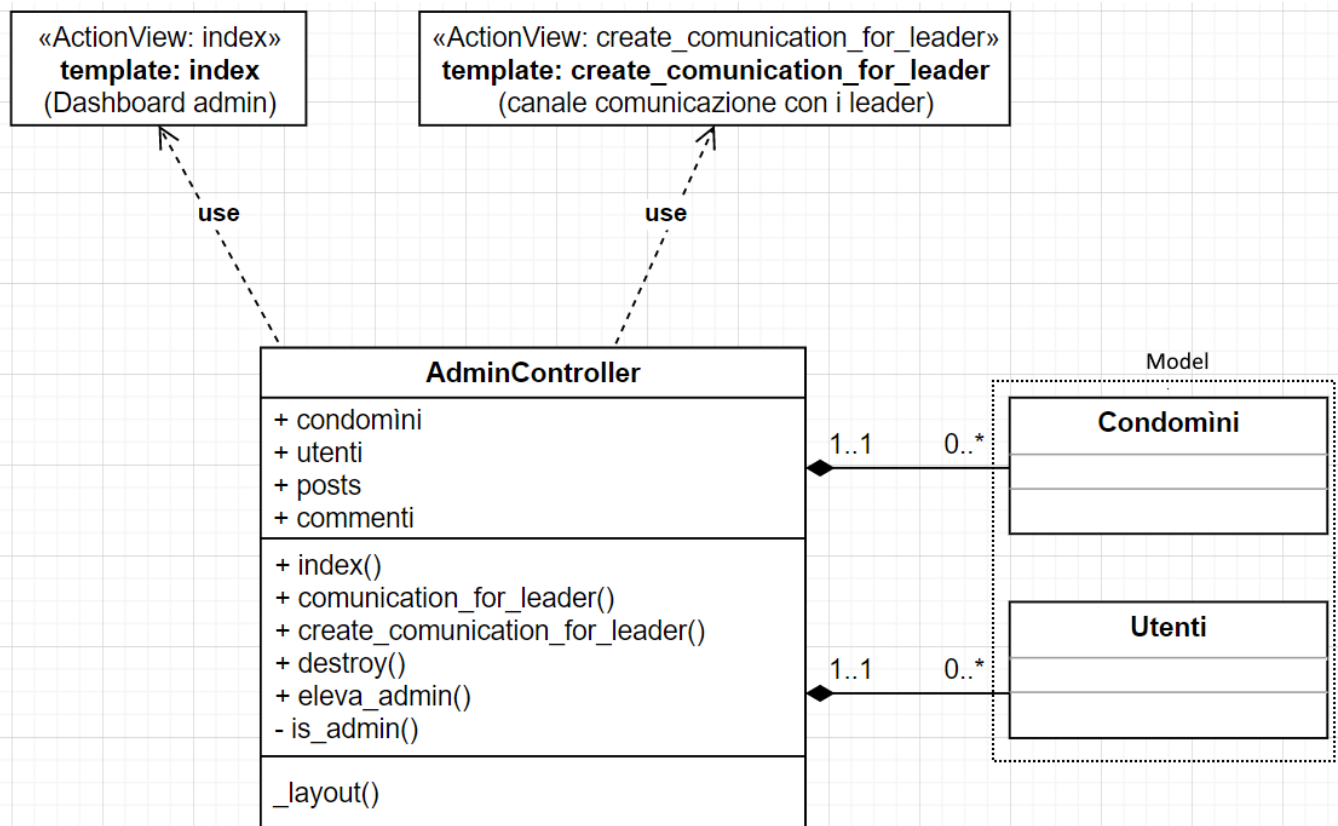


Figura 20 Diagramma delle classi Admin

L'ultimo diagramma che mostriamo è quello relativo ai controller delle cartelle condominio e delle cartelle condomino, che hanno una particolarità rispetto ai diagrammi precedenti, non sono legati ad alcuna vista, fanno solamente operazioni di "backend". Nel nostro caso questi due controller utilizzando le api di Google Drive per permettere la creazione, modifica ed eliminazione delle varie cartelle inerenti ad un condominio, inoltre vengono gestiti tutti i permessi di lettura e scrittura nelle cartelle. Il metodo principale che troviamo in entrambi i controller è "initialize\_drive\_service ()" dove vengono recuperate le credenziali dell'account Google del sito che ha i permessi per utilizzare le varie api di Google, e viene instaurato il servizio di Google Drive per poter usufruire di tutte le funzioni di cui abbiamo bisogno. Per una maggiore chiarezza prima del diagramma delle classi mostriamo la funzione appena descritta.

```

def initialize_drive_service

  file = File.read('config/google_credentials.json')

  @service = Google::Apis::DriveV3::DriveService.new
  scope = 'https://www.googleapis.com/auth/drive'
  authorizer = Google::Auth::ServiceAccountCredentials.make_creds(json_key_io: StringIO.new(file), scope: scope)
  authorizer.fetch_access_token!
  @service.authorization = authorizer

  return @service
end

```

Figura 21 Metodo per utilizzare api di Google Drive

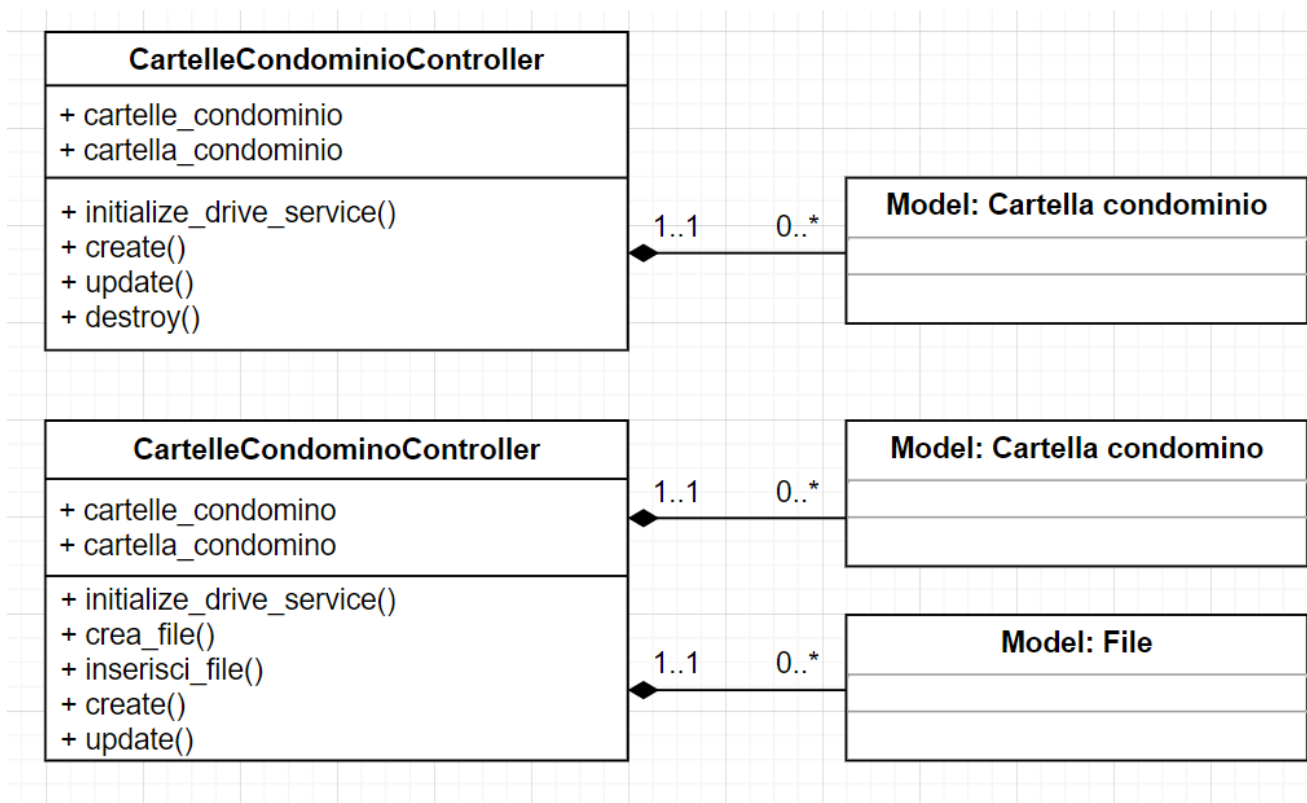


Figura 22 Diagramma delle classi Cartelle condominio e Cartelle Condomino

## 6. Realizzazione del sistema

A questo punto della trattazione verrà mostrato un caso d'uso dell'applicazione. Per fare ciò, verranno utilizzati i "diagrammi dei casi d'uso", i quali permettono attraverso sequenze di passi, di descrivere l'utilizzo del software. Come detto nel primo capitolo, focalizzeremo l'attenzione sulle azioni per creare un "gruppo condominio", premettendo che un utente deve essere autenticato nel sistema.

### 6.1. Accesso al sito web

Elenchiamo le diverse tipologie di utente, già viste nei primi capitoli:

- Utente non registrato: visitatore del sito che non ha effettuato l'accesso all'applicazione.
- Utente registrato: un utente che ha effettuato l'accesso alla piattaforma attraverso la form di login o con OAuth
- Condomino: uno dei partecipanti ad un "gruppo condominio", un utente registrato può essere condomino di più condomini.
- Leader condominio: uno degli amministratori di un "gruppo condominio", che ha funzionalità aggiuntive legate al suo ruolo.
- Admin: amministratore del sito web.

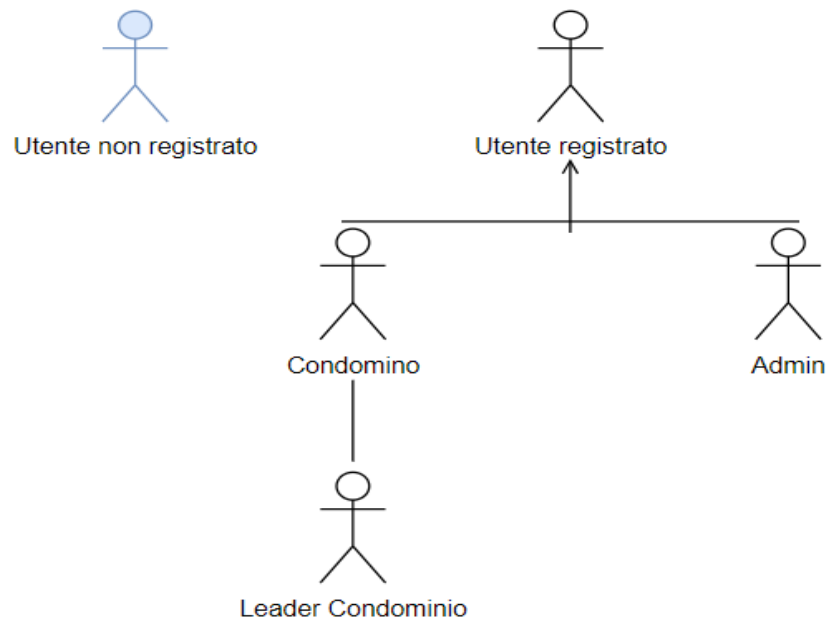


Figura 23 Gerarchia ruoli

Ogni ruolo porta con sé delle funzionalità, ogni permesso verrà gestito dalla gemma “Cancancan”, qui di seguito vengono riassunti in forma tabellare tutti i ruoli con le azioni che può compiere: (In verde ha accessi alla funzionalità, in rosso l'accesso è negato)

	Utente non registrato	Utente registrato	Condomino	Leader Condominio	Admin
Contact Us					
Registrazione account					
Login					
Modifica informazioni account					
Cancellazione account					A
Creazione di un “gruppo condominio”					
Modifica di un condominio				B	B
Eliminazione di un condominio				C	C
Creazione di un post					
Eliminazione di un post			D	D	D
Creazione di un commento					
Eliminazione di un commento			E	E	E
Visualizzazione eventi					
Creazione eventi				F	F
Modifica evento				G	G
Eliminazione evento				H	H
Eliminazione membri condominio				I	I
Elevazione membri condominio				L	L
Accesso in scrittura alla cartella drive condominio				M	M

Accesso in scrittura alla cartella drive condominio			N	N	N
Comunicazione con admin					
Condivisione codice condominio				O	O
Comunicazione per i condomini				P	P
Creazione richiesta d'accesso					
Accettazione richiesta d'accesso					
Visualizzazione membri condominio					

Tabella 7 Permessi e ruoli

Ad ogni nota nella tabella alleghiamo delle specifiche:

- A: l'admin può cancellare qualsiasi account del sito.
- B: l'admin può modificare qualsiasi condominio del sito, mentre il leader solo quelli che amministra.
- C: l'admin può eliminare qualsiasi condominio del sito, mentre il leader solo quelli che amministra.
- D: l'admin può eliminare qualsiasi post del sito, il leader qualsiasi post dei condomini che amministra, mentre il condomino solo quelli scritti da lui.
- E: l'admin può eliminare qualsiasi commento del sito, il leader qualsiasi post dei condomini che amministra, mentre il condomino solo quelli scritti da lui.
- F: l'admin può creare eventi in qualsiasi condominio, mentre il leader solo in quelli che amministra.
- G: l'admin può modificare eventi in qualsiasi condominio, mentre il leader solo in quelli che amministra.
- H: l'admin può eliminare eventi in qualsiasi condominio, mentre il leader solo in quelli che amministra.
- I: l'admin può elevare qualsiasi condomino a Leader condominio, mentre il Leader condominio può farlo solo nei condomini amministrati.



- L: l'admin può far diventare qualsiasi Leader condominio un semplice condomino, mentre il Leader condominio può cedere spontaneamente la sua carica.
- M: l'admin ha accesso a tutte le cartelle condominio del drive, mentre il leader solo alle cartelle dei condomini che amministra.
- N: l'admin ha accesso a tutte le cartelle dei condòmini del sito, il leader a tutte le cartelle dei membri dei condomini amministrati, mentre il condomino solamente alle proprie cartelle personali nei condomini a cui partecipa.
- O: l'admin può condividere il codice di qualsiasi condominio.
- P: l'admin può creare comunicazioni in qualsiasi condominio.

Ora utilizziamo le "Activity Diagrams" per mostrare nello specifico il comportamento dell'applicativo. Distinguendo le azioni eseguite dall'utente, dall'applicazione e l'output. La figura sottostante illustra l'activity diagram per l'autenticazione di un utente del sistema informatico.

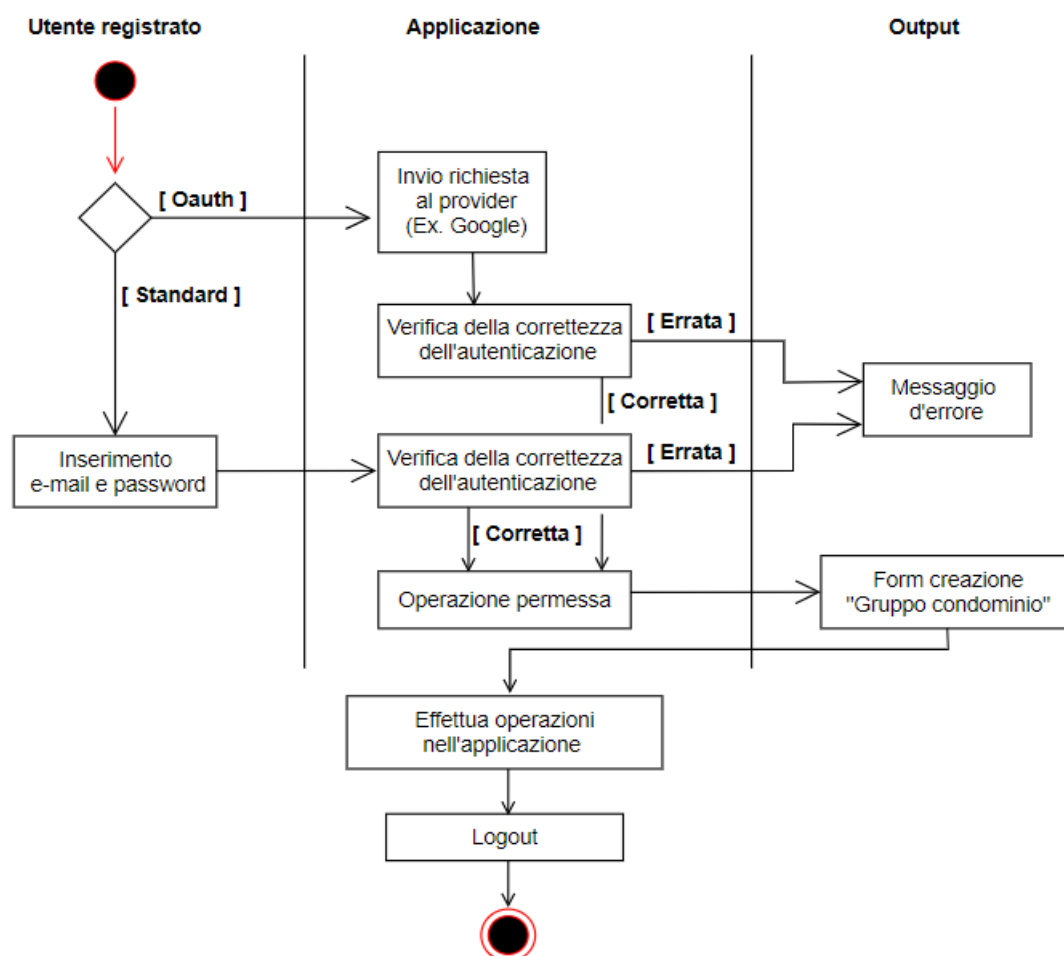


Figura 24 Diagramma attività: autenticazione utente registrato

Qui è stata presa per esempio l'accesso alla form per la creazione di un "gruppo condominio", ma questa procedura è la stessa per accedere a qualsiasi altra funzionalità del sito.

## 6.2. Ciclo di vita di un condomino

Ora parliamo dei vari cambiamenti che possono subire i condomini nel corso dell'utilizzo del sito. Innanzitutto, ci sono due modi per diventare condomino: attraverso l'utilizzo di un codice d'accesso e facendo una richiesta d'accesso, che dovrà essere accettata dal Leader condominio. Il condomino potrà essere elevato a Leader condominio da uno degli amministratori o dall'admin del sito, e avrà anche la possibilità di uscire dal condominio, cancellando la partecipazione al "gruppo condominio". Una volta diventato amministratore il condomino potrà cedere la sua carica o essere declassato dall'admin.

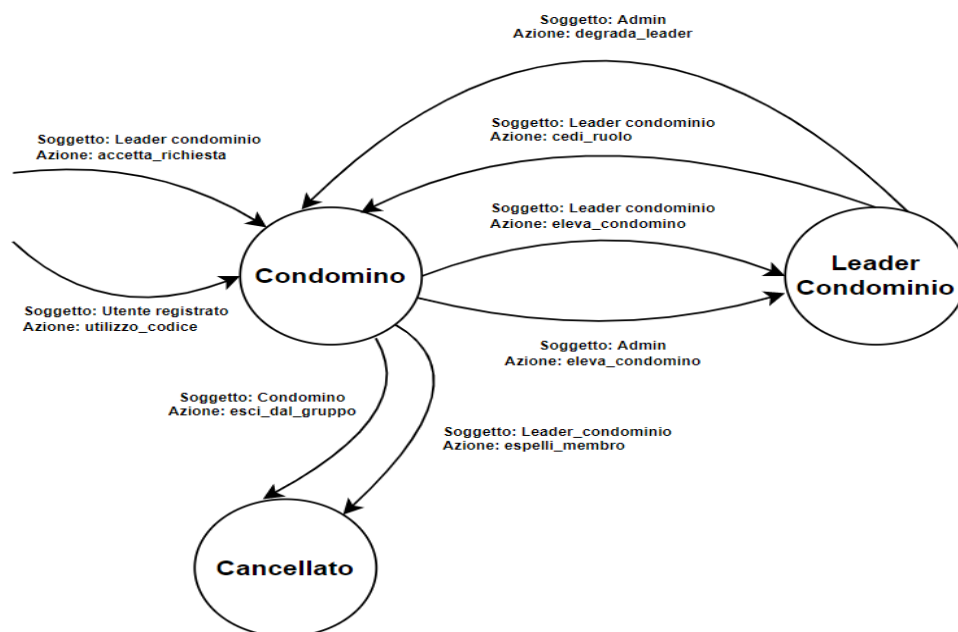


Figura 25 Diagramma a stati di un condomino

Ora utilizziamo i diagrammi dei "casi d'uso" per mostrare un caso d'uso per la creazione di un "gruppo condominio", questi diagrammi mostrano come ci si aspetta che l'utente interagisca con l'applicativo. Un "gruppo condominio" viene creato da un utente registrato inserendo i dati all'interno della form apposita, diventando l'amministratore di esso. In seguito, possono essere modificati i dati del condominio da uno dei Leader condominio o admin, e anche cancellato sempre dagli stessi.

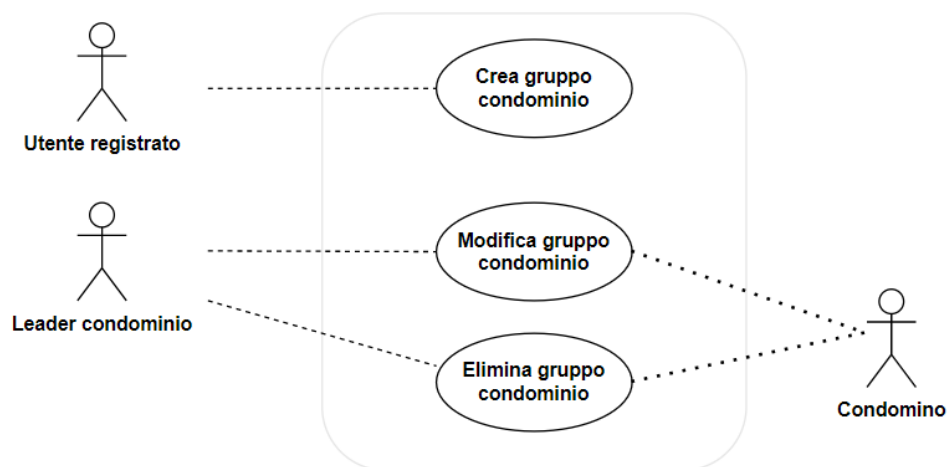


Figura 26 Diagramma casi d'uso

Ora mostriamo, riutilizzando l'activity diagram, le azioni che l'utente deve fare per creare un gruppo condominio. L'utente, come per qualsiasi altra azione nel sito, deve essere autenticato.

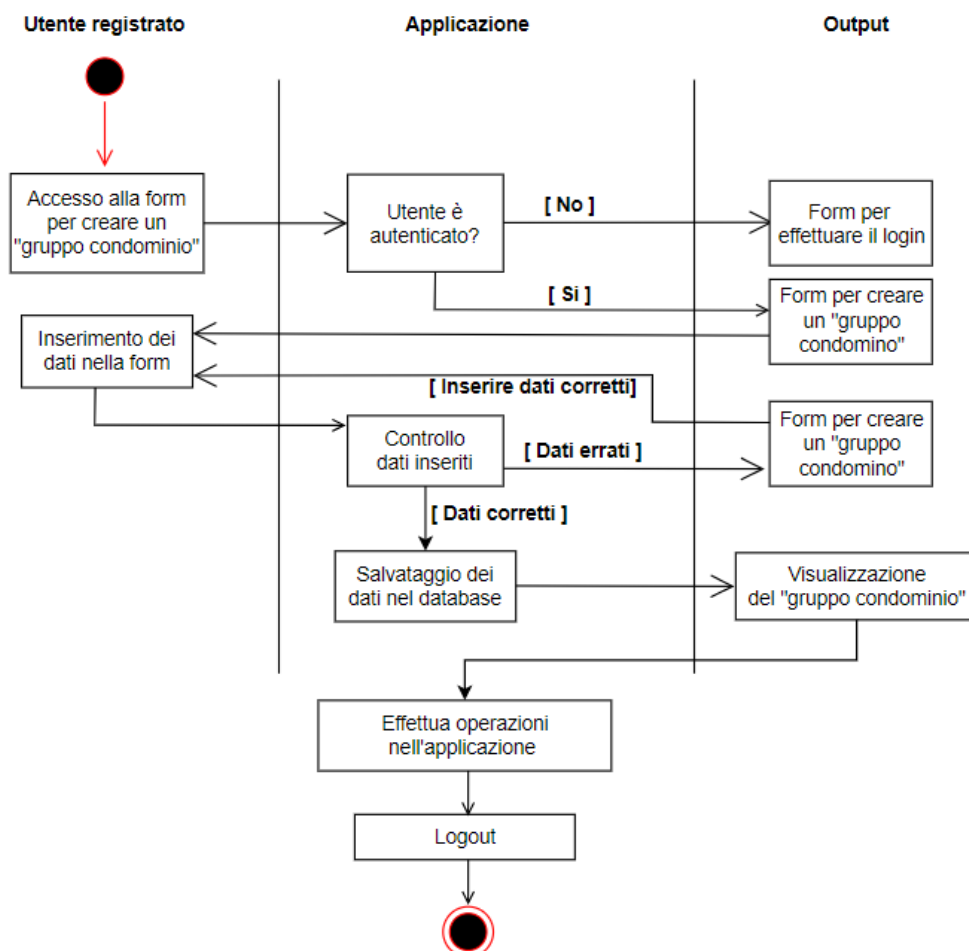


Figura 27 Diagramma attività: creazione di un gruppo condominio

## 7. Validazione e dispiegamento

La fase di “*testing del software*” è fondamentale per ottenere un prodotto finale adeguato, si cercano eventuali problemi nel codice, e, se l’applicativo si comporta come dovrebbe in qualsiasi situazione, sia affidabile in ogni sua parte.

Grazie a questa fase si prevencono bug durante il dispiegamento dell’applicazione, e in generale si hanno prestazioni nettamente migliori.

Altra fase ugualmente importante è quella di validazione, i cui si controlla se l’applicazione informatica rispetti tutti i requisiti e le specifiche predisposte all’inizio dello sviluppo. Nel nostro caso dobbiamo verificare che tutte le “*user stories*” sono state implementate correttamente.

### 7.1. Il Software Testing

Il “*Software Testing*”, dal punto di vista ingegneristico, è un processo di analisi di un prodotto software che determina se esso soddisfa le condizioni stabilite. Vengono ricercate mancate richieste, bugs ed errori generici per valutare il grado di sicurezza e prestazioni del prodotto.

Normalmente in questa fase sarebbe opportuno un gruppo apposito, diverso dal gruppo di sviluppo, ma molte volte per ragioni prettamente economiche, viene accorpata la fase di sviluppo e testing in un unico gruppo.

Nel mondo attuale lo sviluppo degli applicativi deve essere molto rapido e la distribuzione di nuove funzionalità costante nel tempo, per rimare al passo con lo sviluppo tecnologico e rispondere alle richieste del mercato sempre più frenetico ed esigente. Per permette alle aziende di mantenere questo ritmo opprimente, l’approccio allo sviluppo e al testing dei prodotti ha subito dei grandi cambiamenti, anche per mantenere le *performance* ad un livello adeguato.

Qui entrano nel discorso i metodi agili trattati all’inizio del documento, che hanno come obiettivo quello appunto di rispondere velocemente alle richieste sempre più numerose, mantenendo i costi bassi ed efficientando ogni singola fase dello sviluppo.

Questi metodi, a differenza di quelli tradizionali, si basano sul fatto che la qualità del software è il risultato di un processo, e non dipende dalla responsabilità di uno specifico gruppo. Il *testing*, parte di ogni iterazione *agile*, deve essere automatizzato il più possibile, e il lavoro di gruppo aumenta la qualità del risultato finale. Da ciò sono nati questi tipi di sviluppo guidati dai test:

- Test Driven Development (TDD): è un processo di sviluppo del software,

in cui i test vengono scritti ancora prima del codice, e quindi diventano proprio essi il cuore dello sviluppo.

- Behavior Driven Development (BDD): i test utilizza i costrutti dei linguaggi naturali, non utilizzando termini tecnici, così che i test diventino comprensibili da tutti, e la fase di testing diventa più collaborativa.

## 7.2. Behavior-driven development

BDD è un modo che permette ai gruppi di sviluppo *software* di lavorare a stretto contatto con gli altri gruppi di lavoro che non comprendono i tecnicismi dello sviluppo, per fare ciò c'è bisogno della comprensione condivisa del problema da risolvere, lavorare velocemente con piccole iterazioni così da aumentare i riscontri. Chiedere agli utenti come ci si aspetta che l'applicazione si comporti, utilizzare le *user stories* così da spiegare con linguaggio naturale e schematico le funzionalità richieste. Questo processo di sviluppo incoraggia a lavorare in rapide iterazioni, dividendo le problematiche degli utenti in piccole parti così da risolverle nel modo più rapido possibile. Solitamente si seguono tre passi:

- Ricerca: prendere una piccola modifica del sistema, una nuova *user stories*, e parlare su un esempio concreto di tale funzionalità e concordarsi su come ci si aspetta di farla.
- Formulazione: documentare quanto concordato nella prima fase, in un modo che può essere automatizzato.
- Automazione: implementare lo scenario descritto dalla documentazione, iniziando con un test automatico che guiderà lo sviluppo del codice, come in TDD.

Le *user stories* sono scritte in modo da essere comprese sia dagli umani che dai computer; infatti, i test fanno uso del linguaggio *Gherkin*, un insieme di regole che fanno in modo che il testo che scriviamo in linguaggio naturale per i test sia comprensibile dai calcolatori. Per fare ciò *Gherkin* fa uso di alcune parole chiave.

Come prima cosa in un file di test viene descritta la funzionalità da testare nel modo meno ambiguo possibile utilizzando la parola chiave *Feature*. Poi utilizzando uno *Scenario* descriviamo l'utilizzo di tale funzionalità e cosa ci si aspetta da un'esecuzione corretta. Infine, si utilizzano vari *steps* per andare ad automatizzare il test di tale funzionalità utilizzando sempre un linguaggio naturale.

Per non fare una trattazione solamente teorica, mostriamo ora un caso di utilizzo pratico, avvalendoci di una delle funzionalità mostrate nel secondo capitolo, la creazione della cartella Google drive di un condomino.

*Come Condomino, in modo da poter condividere documenti con i soli Leader condominio, voglio poter accedere ad una cartella Google Drive dedicata tramite l'applicazione*

Nel riquadro sottostante mostriamo il test automatizzato di tale funzionalità. Utilizziamo *feature* per definire il nome della funzionalità, con *Background* spieghiamo il contesto in cui ci troviamo e con *Scenario* spieghiamo cosa ci si aspetta da esecuzione corretta della funzionalità.

Gli steps utilizzati nello *Scenario* e nel *Background* che sono *When*, *Then* e *And*, permettono una traduzione diretta della user story, e di far comprendere al compilatore cosa deve fare.

**Feature:** Creazione cartella Google Drive del condomino nel condominio

**US:** Come condomino, in modo da poter condividere documenti con i soli Leader condominio, voglio poter accedere ad una cartella Google Drive dedicata tramite l'applicazione.

**Background:** l'utente ha un account, ha effettuato l'accesso ed è nella pagina di controllo

**Given** il seguente utente esiste

| test | [test@example.com](mailto:test@example.com) | Test1234@ |

**And** sono autenticato

**And** sono nella pagina di controllo

**Scenario:** Un utente registrato può entrare in un condominio, diventando membro del condominio e viene creata una cartella Google Drive dell'utente all'interno del condominio.

**When** ho usato un codice per entrare in un condominio

**Then** dovrei essere nella pagina del condominio

**And** dovrei vedere "Benvenuto nel condominio."

**And** la cartella Google Drive esiste

**And** sono un Condomino

Attraverso un'altra gemma fondamentale *Cucumber*, che è uno strumento per testare software, vengono eseguiti test di accettazione automatici con alla base sempre *Gherkin*. Tale strumento fa il parsing delle stringhe scritte in linguaggio naturale in codice Ruby presenti nel file di definizione della feature, se non viene trovato un riscontro viene lanciato un errore in cui si richiede l'implementazione della stringa.

Nel riquadro sottostante mostriamo parte del file di definizione della feature appena mostrata:

```

Given /^il seguente utente esiste: $/ do |instances|
  Create_user
end

def Create_user
  @user = FactoryBot.create(:e-mail)
end

Given /^ho usato un codice per entrare in un condominio$/ do
  @user1 = FactoryBot.create(:e-mail1)
  steps %Q{
    When I follow "Home"
    And I follow "Account"
    And premo "logout"
    Then dovrei essere nell' home page
    And dovrei vedere "Hai effettuato correttamente il logout"
    When I follow "Login"
    And I fill in the following:
      | user_email   | test1@example.com |
      | user_password | Test1234@         |
    And premo "Log in"
    When I follow "Nuovo condominio"
    Then dovrei essere nella nuova pagina condominio
    When I fill in the following:
      | condominio_nome   | test_condominio   |
      | condominio_comune | Roma              |
      | condominio_indirizzo | Via Tiburtina 214 |
    And premo "Crea Condominio"
    Then dovrei vedere "Condominio creato correttamente."
  }
  flat_code = Condominio.find(1).flat_code
  steps %Q{
    When I follow "Home"
    And I follow "Account"
    And premo "logout"
    Then dovrei essere nell' home page
    And dovrei vedere "Hai effettuato correttamente il logout"
    When I follow "Login"
    And I fill in the following:
      | user_email   | test@example.com |
      | user_password | Test1234@         |
    And premo "Log in"
    And I fill in the following:
      | codice | #{flat_code} |
    And premo "Entra"
  }
end

When /^premo "([^"]*)"$/ do |button|
  click_button(button)
end

When /^(?:|I )follow "([^"]*)"$/ do |button|
  click_link(link, match: :first)
end

```

Ora qui mostriamo come è stata testata la creazione della cartella Google Drive del condominio:

**Feature:** Creazione cartella Google Drive del condominio

**Background:** l'utente ha un account, ha effettuato l'accesso ed è nella pagina di controllo

**Given** il seguente utente esiste

test	test@example.com	Test1234@
------	------------------	-----------

**And** sono autenticato

**And** sono nella pagina di controllo

**Scenario:** Un utente registrato può creare un condominio, diventando Leader del condominio, e viene creata una cartella Google Drive del condominio. Inoltre, può modificare le informazioni e cancellare il condominio stesso.

**When** I follow "Nuovo condominio"

**Then** dovrei essere nella pagina per creare un condominio.

**When** compilo il seguente:

condominio_nome	test_condominio	
condominio_comune	Roma	
condominio_indirizzo	Via Tiburtina 214	

**And** premo "Crea condominio."

**And** sono un Leader condominio

**When** I follow "Modifica"

**And** sono un Condomino

**And** compilo il seguente:

condominio_nome	test_condominio_modifica	
condominio_comune	Avezzano	
condominio_indirizzo	Piazza Torlonia 12	

**And** premo "Modifica condominio."

**Then** dovrei vedere "Condominio è stato aggiornato."

**When** I follow "Home"

**And** I follow "Elimina"

**Then** dovrei vedere "Condominio è stato eliminato correttamente."

**Scenario:** Un utente registrato non può creare un condominio senza inserire un nome.

**When** I follow "Nuovo condominio"

**Then** dovrei essere nella pagina per creare un condominio.

**When** compilo il seguente:

condominio_comune	Roma	
condominio_indirizzo	Via Tiburtina 214	

**And** premo "Crea condominio."

**And** dovrei vedere "Nome è troppo corto (il minimo è 1 carattere)."

**Scenario:** Un utente registrato non può creare un condominio senza inserire il comune.

**When** I follow "Nuovo condominio"

**Then** dovrei essere nella pagina per creare un condominio.

**When** compilo il seguente:

condominio_nome	test_condominio	
condominio_indirizzo	Via Tiburtina 214	

**And** premo "Crea condominio."

**And** dovrei vedere "Comune è troppo corto (il minimo è 1 carattere)."

**Scenario:** Un utente registrato non può creare un condominio senza inserire un indirizzo corretto.

**When** I follow "Nuovo condominio"

**Then** dovrei essere nella pagina per creare un condominio.

**When** compilo il seguente:

condominio_nome	test_condominio_modifica	
condominio_comune	Avezzano	

**And** premo "Crea condominio."

**And** dovrei vedere "Indirizzo invalido, formato: Via Tiburtina 214."



Passiamo ora ai test sull'autenticazione dell'utente, per prima cosa illustriamo i test sulla parte di registrazione:

**Feature:** Creare un account.

**US:** Come utente non registrato, in modo da diventare un utente registrato, voglio potermi iscrivere con la mia e-mail, nome e cognome.

**Background:** l'utente non è registrato nel sito ed è nella pagina di registrazione.

**Given** non sono autenticato

**And** sono nella pagina per registrarsi

**Scenario:** Un utente non registrato può creare un account.

**When** compilo il seguente:

user_username	AndreaTest	
user_email	andreatest@example.com	
user_password	Test123#	
user_password_confirmation	Test123#	

**And** premo "Sign up."

**Then** dovrei essere nella dashboard di benvenuto dell'utente "AndreaTest"

**And** dovrei vedere "Hai completato la registrazione, benvenuto in CondominioOrganizer."

**Scenario:** Un utente non registrato non può creare un account senza inserire un username.

**When** compilo il seguente:

user_email	andreatest@example.com	
user_password	Test123#	
user_password_confirmation	Test123#	

**And** premo "Sign up"

**Then** dovrei fallire la creazione dell'account con "Username non può essere lasciato bianco"

**Scenario:** Un utente non registrato non può creare un account senza inserire un indirizzo e-mail.

**When** compilo il seguente:

user_username	AndreaTest	
user_password	Test123#	
user_password_confirmation	Test123#	

**And** premo "Sign up"

**Then** dovrei fallire la creazione dell'account con "Email non può essere lasciato bianco"

**Scenario:** Un utente non registrato non può creare un account inserendo un indirizzo e-mail non valido.

**When** compilo il seguente:

user_username	AndreaTest	
user_email	email_invalida	
user_password	Test123#	
user_password_confirmation	Test123#	

**And** premo "Sign up"

**Then** dovrei fallire la creazione dell'account con "Email non è valido"

**Scenario:** Un utente non registrato non può creare un account non inserendo una password.

**When** compilo il seguente:

user_username	AndreaTest	
user_email	andreatest@example.com	

**And** premo "Sign up"

**Then** dovrei fallire la creazione dell'account con "Password non può essere lasciato bianco"

**Scenario:** Un utente non registrato non può creare un account inserendo la password di conferma diversa da quella inserita.

**When** compilo il seguente:

user_username	AndreaTest	
user_email	andreatest@example.com	
user_password	Test123#	
user_password_confirmation	Testerrato123#	

**And** premo "Sign up"

**Then** dovrei fallire la creazione dell'account con "Password confirmation non coincide con Password"

```

Scenario: Un utente non registrato non può creare un account inserendo un indirizzo e-mail già inserito
da un altro utente.
Given il seguente utente esiste:
    | test | test@example.com | Test1234@ |
When compilo il seguente:
    | user_uname      | AndreaTest      |
    | user_email      | test@example.com |
    | user_password   | Test123#        |
    | user_password_confirmation | Test123#        |
And premo "Sign up"
Then dovrei fallire la creazione dell'account con "Email è già presente"
When I follow "Log in"
Then dovrei essere nella pagina di login
When compilo il seguente:
    | user_email | test@example.com |
    | user_password | Test1234@ |
And premo "Log in"
Then dovrei vedere "Bentornato, continua con la navigazione nel sito"

```

Per finire i test riguardanti l'accesso al proprio account, specificando che questi test insistono solo su una parte dell'applicativo, si potrebbero realizzare altri test ancora.

**Feature:** Accedere al proprio account.

US: Come utente registrato, in modo da avere accesso al mio account, voglio una pagina per effettuare login.

**Background:** L'utente ha un account ed è nella homepage

```

Given il seguente utente esiste:
    | test | test@example.com | Test1234@ |
And sono nella home page

```

**Scenario:** Un utente registrato può accedere al proprio account.

```

When I follow "Login"
Then dovrei essere nella pagina di login
When compilo il seguente:
    | user_email | test@example.com |
    | user_password | Test1234@ |
And premo "Log in"
And dovrei vedere "Bentornato, continua la navigazione nel sito."

```

**Scenario:** Un utente registrato può effettuare il logout.

```

Given ho effettuato l'autenticazione
And sono nella pagina dell'account
When premo "logout"
Then dovrei essere nella home page
And dovrei vedere "Hai effettuato correttamente il logout."

```

**Scenario:** Un utente registrato può cancellare il proprio account.

```

Given ho effettuato l'autenticazione
And sono nella pagina dell'account
When premo "Cancella il mio account"
Then dovrei essere nella home page
And dovrei vedere "Hai cancellato correttamente il tuo account."

```

**Scenario:** Un utente registrato può cambiare il proprio indirizzo e-mail.

```

Given ho effettuato l'autenticazione
And sono nella pagina dell'account
When compilo il seguente:
    | user_email      | andreatestnuovo@example.com |
    | user_current_password | Test1234@ |
And premo "Modifica"
Then dovrei essere nella pagina di benvenuto
And dovrei vedere "Hai aggiornato le tue informazioni."

```

### 7.3. Test-driven development

TDD, come BDD, è un processo di sviluppo del software che segue il principio secondo cui i test devono essere *FIRST*, cioè:

- **Fast:** i test devono essere eseguiti velocemente, visto che saranno il cuore dello sviluppo.
- **Independent:** ogni test che scriviamo deve essere indipendente dagli altri, così che possono essere eseguiti in modo distinto.
- **Repeatable:** i test ogni volta che vengono rieseguiti devono dare lo stesso risultato, permettendo quindi l'automazione di essi.
- **Self-checking:** deve essere possibile automatizzare il processo di determinare il corretto superamento del test, senza l'iterazione di una persona con il terminale.
- **Timely:** i test devono essere scritti prima del codice, seguendo i principi dei metodi *agile*.

Inoltre, questo processo porta a una migliore qualità del codice, con una migliore comprensibilità, grazie alla separazione dei problemi. Ovviamente questo permette una manutenibilità del codice migliore, velocizzando i tempi di sviluppo e riducendo i costi. TDD usa un loop di sviluppo fissato che aiuta lo sviluppatore, che viene chiamato normalmente “*red-green-refactor loop*”, come mostrato nella figura sottostante.



Figura 28 Red-green-refactor loop

Bisogna per prima cosa scrivere un test che fallirà, non avendo inserito ancora del codice, dopodiché scrivere il minimo codice che renda funzionante il test, e infine ripulire il codice e continuare con lo stesso ciclo.

Invece di Cucumber che usiamo in BDD, qui viene utilizzato RSpec un'altra gemma che permette di scrivere test in Ruby on Rails. Ciò che porta la scelta di tale tecnologia, è il fatto che possiamo scrivere dei test per singoli moduli,

quindi si singole viste, controller e modelli. Ovviamente come visto con Cucumber il calcolatore ha bisogno di parole chiave per comprendere cosa deve fare, qui si utilizza *describe* per spiegare l'azione che stiamo testando e *it* per scrivere tutti i test legati a quell'azione e cosa ci aspettiamo per far essi siano corretti, tutto come sempre in linguaggio naturale.

Mostriamo nel riquadro sottostante un caso di utilizzo pratico, in particolare andremo a fare dei test sul modello del condomino. Per testare poi qualsiasi altra unità si dovranno fare operazioni analoghe a quelle che mostriamo.

I test vanno scritti in file con il seguente formato *nomeunità\_spec.rb* all'interno della cartella spec.

```
require 'rails_helper.rb'

RSpec.describe Condomino, type: :model do
  before do
    @user = FactoryBot.create(:e-mail)
    @condominio = FactoryBot.create(:condominio)
    @condominio = FactoryBot.create(:condominio, user_id: @user.id, condominio_id: @condominio.id)
  end

  describe 'creation' do
    it 'can be created if valid' do
      expect(@condominio).to be_valid
    end

    it 'will not be created if not valid user_id' do
      @condominio.update(user_id: 243)
      @condominio.save
      expect(@condominio).to_not be_valid
    end

    it 'can create the connected drive folder' do
      expect(@condominio.gdrive_user_items).to_not be_nil
    end
  end

  describe 'update' do
    it 'can be updated (admin status)' do
      @condominio.update(is_condo_admin: true)
      @condominio.save
      expect(@condominio.reload.is_condo_admin).to match(true)
    end
  end

  describe 'destruction' do
    it 'can be destroyed' do
      @condominio.destroy
      expect{@condominio.reload}.to raise_error ActiveRecord::RecordNotFound
      expect{@condominio.reload.gdrive_user_items}.to raise_error ActiveRecord::RecordNotFound
    end
  end
end
```

Mostriamo ora altri test, iniziando con quelli relativi al modello del condominio:

```
require 'rails_helper.rb'
require 'condominio.rb'

RSpec.describe Condominio, type: :model do
  before do
    @condominio = FactoryBot.create(:condominio)
  end

  describe 'creation' do
    it 'can be created if valid' do
      expect(@condominio).to be_valid
    end

    it 'can't be created if comune is longer than 25 characters' do
      @condominio.comune = 'nomecomunetroppolungoperilsito'
      expect(@condominio).to_not be_valid
    end

    it 'can't be created if comune is shorter than 1 characters' do
      @condominio.comune = ''
      expect(@condominio).to_not be_valid
    end

    it 'can't be created if nome is longer than 25 characters' do
      @condominio.nome = 'nomecondominiotroppolungoperilsito'
      expect(@condominio).to_not be_valid
    end

    it 'can't be created if nome is shorter than 1 characters' do
      @condominio.nome = ''
      expect(@condominio).to_not be_valid
    end

    it 'can't be created if indirizzo is in the wrong format' do
      @condominio.indirizzo = 'indirizzo invalido'
      expect(@condominio).to_not be_valid
    end
  end

  describe 'update' do
    it 'can be updated' do
      @condominio.update(nome: 'Condominio di Andrea')
      @condominio.save
      expect(@condominio).to be_valid
    end

    ...
  end

  describe 'destruction' do
    it 'can be destroyed' do
      @condominio.destroy
      expect{@condominio.reload}.to raise_error ActiveRecord::RecordNotFound
      expect{@condominio.reload.gdrive_condo_items}.to raise_error ActiveRecord::RecordNotFound
    end
  end
end
```

Gli ultimi test che illustriamo sono quelli del modello degli *user*, ma come detto prima per BDD, si potrebbero realizzare ancora altri test su altre unità della nostra applicazione.

```
require 'rails_helper.rb'
require 'user.rb'

RSpec.describe User, type: :model do
  before do
    @user = FactoryBot.create(:e-mail)
    @user1 = FactoryBot.create(:e-mail1)
  end

  describe 'creation' do
    it 'can be created if valid' do
      expect(@user).to be_valid
    end

    it 'will not be created if not valid username' do
      @user.username = nil
      expect(@user).to_not be_valid
    end

    it 'will not be created if not valid e-mail' do
      @user.email = 'e-mail invalida'
      expect(@user).to_not be_valid
    end

    it 'will not be created if not valid password' do
      @user.password = 1234
      expect(@user).to_not be_valid
    end

    it 'will not be created if mail exist' do
      @user.email = 'test1@example.com'
      expect(@user).to_not be_valid
    end
  end

  describe 'update' do
    it 'will not be update if not valid password' do
      @user.save!
      @user.update(password: 'Test')
      expect(@user).to be_valid
    end

    it 'will not be update if not valid e-mail address' do
      @user.save!
      @user.update(email: 'Email invalida')
      expect(@user).to be_valid
    end

    it 'will not be update if e-mail address exist' do
      @user.save!
      @user.update(email: 'test1@example.com')
      expect(@user).to be_valid
    end
  end
end
```

```

describe 'destruction' do
  it 'will be destroyed' do
    @user.destroy
    expect{@user.reload}.to raise_error ActiveRecord::RecordNotFound
  end

  it 'will be canceled all shareholdings in condominiums' do
    @condominio = FactoryBot.create(:condominio)
    @partecipazione = FactoryBot.create(:condominio, user_id: @user.id, condominio_id:
@condominio.id)
    @user.destroy
    expect{@user.reload}.to raise_error ActiveRecord::RecordNotFound
    expect{@partecipazione.reload}.to raise_error ActiveRecord::RecordNotFound
  end
end
end

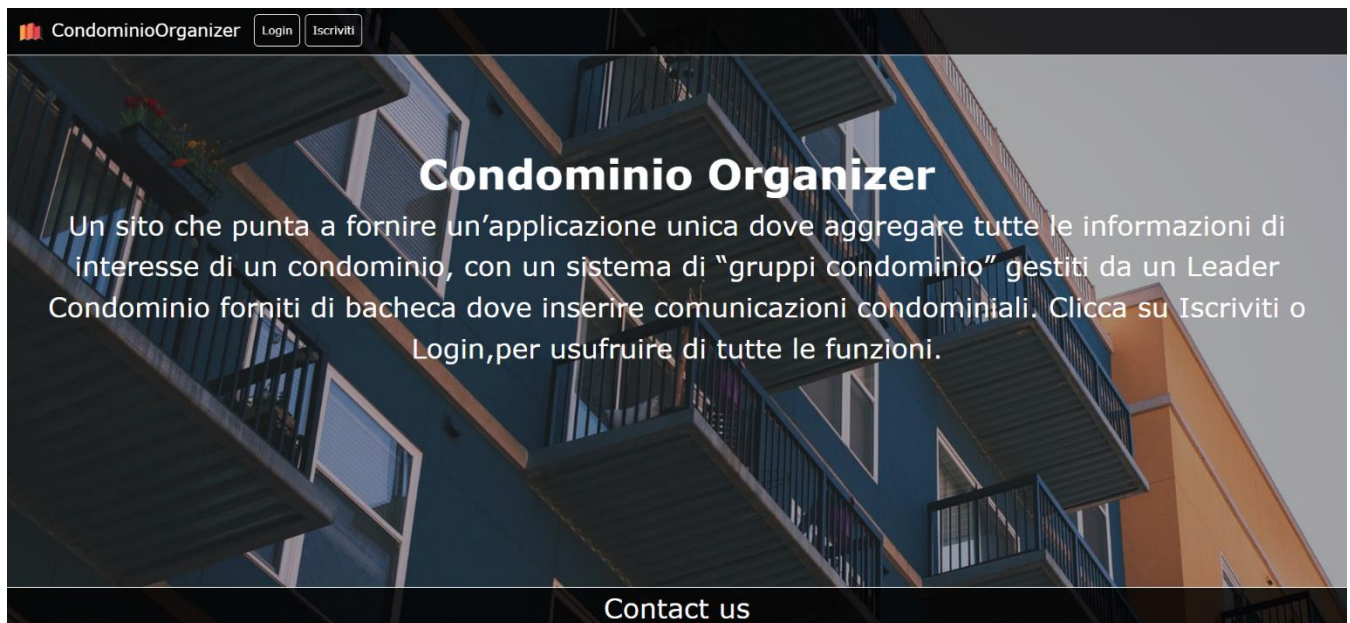
```

## 7.4. Dispiegamento

In quest'ultima parte parliamo del dispiegamento, che è la fase in cui rendiamo disponibile l'applicativo all'utente finale. Una modalità per farlo è mettere tutte le componenti del *software* su una macchina che eseguirà tutta la nostra parte di *back-hand*. Una soluzione è usare Heroku, che è una piattaforma cloud per distribuire applicativi online, particolarmente comodo è il fatto che è possibile legare l'account di Heroku con la code base di GitHub così che tutte le modifiche che vengono fatte sul repository, automaticamente vanno a riflettersi online. Quando si utilizzano queste piattaforme cloud molte volte bisogna riadattare il software per renderlo compatibile, come per esempio potrebbero non essere supportati alcuni DBMS, o solo alcune versioni del programma che stiamo utilizzando sono fruibili.

Nel caso del mio gruppo di lavoro durante il dispiegamento sulla piattaforma Heroku sono stati riscontrati diversi problemi. Il primo problema è stato il fatto che durante la fase di sviluppo come RDBMS è stato usato SQLite, che purtroppo non è supportato; quindi, è si è dovuto passare ad un altro database relazionale, cioè PostgreSQL, ma grazie all'utilizzo di Ruby on Rails i cambiamenti fatti sul codice sono stati minimi.

Altro problema è stato cambiare le impostazioni delle varie gemme utilizzate; infatti, sono stati cambiati i file di configurazione di Geocoder e icalendar, ma anche qui il lavoro non è stato eccessivo. Risolti questi piccoli problemi il dispiegamento è stato molto facile, ora per dare un riscontro visivo del risultato finale alleghiamo diverse schermate dell'applicativo funzionante online. Per prima cosa mostriamo l'homepage che viene visualizzata dagli utenti non registrati, la pagina di registrazione e quella di accesso.



## Sign up

Username

Email

Password (8 characters minimum)

Password confirmation

Sign up

[Sign in with Google](#) [Log in](#)

## Log in

Email

Password

☐ Remember me

Log in

[Sign in with Google](#) [Sign up](#)

[Forgot your password?](#)

Figura 29 Home page, Sign up e Log in

Altra parte fondamentale del sito che mostriamo di seguito, è quella che riguarda l'accesso al condominio di un utente e la bacheca di un condominio. Inoltre, mostriamo la visualizzazione della cartella Google Drive da parte di un amministratore, nel nostro caso del comune Roma, che ha al suo interno altre tre cartelle che rappresentano i tre condomini che partecipano al "gruppo condominio".



The screenshot displays a web application for managing a condominium. At the top, there is a decorative orange wave graphic. Below it, the main heading is "Informazioni condominio". Under this heading, several fields provide details: "Nome: comune roma", "Comune: Roma", "Indirizzo: via dei latini 18", "Partecipanti:" followed by a "Lista membri" button, and "Codice: gv6Q".

Below the information section, there are three buttons: "Comunica con Admin", "Modifica", and "Home".

The "Bacheca dei post:" section shows a post by a user named "Topolino" with the text "ciao". Below the text are buttons for "Commenti" and "Simuovi post".

The "Aggiungi un Post:" section contains a form with fields for "Title" and "Body", a "File" selection button (labeled "Scagli file"), and a note: "Non selezionando nessuno, il file non verrà caricato, ma pubblicato solo il post". Below this, there are checkboxes for users "andrea pancari" and "Topolino". A "Crea Post" button is at the bottom of this section.

Below the post creation section is a calendar for "agosto 2022". The calendar has columns for days of the week (lun, mar, mer, gio, ven, sab, dom) and rows for weeks. The date "31" is highlighted in yellow.

At the bottom of the page, there are buttons for "Crea evento", "Eventi creati", and "Accedi al Google Drive".



lun	mar	mer	gio	ven	sab	dom
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	1	2	3	4

Figura 30 Bacheca di un condominio

## Entra tramite codice...

Codice d'invito:  [Entra](#)

## ...cerca il tuo condominio

Città:	<input type="text" value="Roma"/>	<b>Banner</b>	<b>Nome</b>	<b>Comune</b>	<b>Indirizzo</b>	
Distanza (in chilometri):	<input type="text" value="100"/>		comune roma	Roma	via dei latini 18	<a href="#">Richiedi accesso</a>
<a href="#">Cerca</a>			Mio Condo	Roma	Via Milano 123	<a href="#">Richiedi accesso</a>

## ...crea il tuo condominio

[Nuovo condominio](#)[Home](#)

Figura 31 Pagina di iscrizione a un "gruppo condominio"

Condivisi con me > comune roma ▾ 

### Cartelle




 andrea panceri	 andreaaa	 Topolino
--	--	--

Figura 32 cartella Google Drive di un "gruppo condominio"

## Bibliografia

- [1] "Introduzione ad Oauth 2.0", <https://www.teranet.it/introduzione-ad-oauth-2>
- [2] "Cos'è un' API?", <https://aws.amazon.com/it/what-is/api/>
- [3] Beck K., Beedle M., Van Bennekum A., Cockburn A., Cunningham W., Fowler M., Grenning J., Highsmith J., Hunt A., Jeffries R., Kern J., Marick B., Martin R., Mellor S. Schwaber K., Sutherland J., Thomas D., Manifesto for Agile software development, 2001.
- [4] Don Wells, Extreme Programming: A gentle introduction, 2013. (Ultima modifica ottobre 8, 2013)
- [5] David Heinemeier Hansson, [https://guides.rubyonrails.org/getting\\_started.html](https://guides.rubyonrails.org/getting_started.html)
- [6] Fowler M., Patterns of Enterprise Application Architecture, 2002
- [7] David Heinemeier Hansson, [https://guides.rubyonrails.org/active\\_record\\_basics.html](https://guides.rubyonrails.org/active_record_basics.html)
- [8] David Heinemeier Hansson, [https://guides.rubyonrails.org/action\\_controller\\_overview.html#what-does-a-controller-do-questionmark](https://guides.rubyonrails.org/action_controller_overview.html#what-does-a-controller-do-questionmark)