



UNIVERSITÀ DI PARMA

Report progetto di Tecnologie Internet

Weather App: “Un servizio web meteo al vostro servizio”

Elena Barzaghi – 318860

Francesco Zoni - 323085

Indice

Amministratore.....	4
Analisi e specifica dei requisiti	3
API.....	5
Base dati	11
Conclusione	16
Condomino	4
Configurazione del middleware	5
Configurazione di express-session	5
Configurazione e Dipendenze	5
Connessione al database	6
Decisioni progettuali	5
Diagramma delle classi	15
Endpoint API	5
Endpoint per la gestione delle riunioni e delle pagine	6
Framework.....	5
Funzioni per il recupero dei dati dal database	6
Gestione dei file statici	5
Introduzione e requisiti informali.....	3
Obiettivi	3
Processo di sviluppo	16
Progettazione	5
Specifica dei requisiti funzionali.....	4
Specifica dei requisiti non funzionali.....	4
Stake holders	3
Tecnologie utilizzate.....	10

Introduzione e requisiti informali

In Italia, la digitalizzazione è stata spesso trascurata, ma negli ultimi tempi abbiamo assistito a un crescente interesse verso applicazioni, siti web e soluzioni digitali che non solo intrattengono, ma offrono anche funzionalità pratiche. È in questo contesto che abbiamo deciso di sviluppare un sito web, con la prospettiva di espanderlo in futuro in un'applicazione mobile. In pratica, ci siamo dedicati alla creazione di una piattaforma web su misura per la gestione condominiale, mirando a migliorare l'esperienza e la qualità della vita all'interno dei condomini."

Obiettivi

Il nostro principale obiettivo con il progetto di gestione condominiale è quello di creare un ambiente digitale che semplifichi e ottimizzi le comunicazioni tra i condomini e l'amministratore del condominio. L'obiettivo fondamentale è quello di fornire una soluzione completa e intuitiva che consenta a tutti i membri della comunità condominiale di interagire in modo efficace, accedendo a un sistema di notifiche ben strutturato ed efficiente. Ci impegniamo a garantire che ogni aspetto delle comunicazioni all'interno del condominio, dalle informazioni cruciali alle segnalazioni e alle discussioni, avvenga in modo trasparente e con facilità. Inoltre, ci siamo posti come scopo, la ricezione e l'accessibilità di documenti come i verbali delle riunioni o i bilanci degli anni passati e la gestione autonoma dei pagamenti condominiali. In sintesi, miriamo a migliorare la qualità della vita condominiale attraverso una gestione delle notifiche all'avanguardia, consentendo a tutti i condomini di partecipare attivamente e di essere sempre informati su ciò che accade nella loro comunità

Analisi e specifica dei requisiti

In questa sezione, vengono definite le proprietà dell'applicazione. I requisiti rappresentano le caratteristiche, le proprietà e i comportamenti che l'applicazione deve possedere al termine dello sviluppo. Per condurre l'analisi dei requisiti, si individuano i casi d'uso, gli attori e le relazioni che si verificano tra di essi.

I casi d'uso descrivono le interazioni tra il sistema e l'utente, mentre gli attori rappresentano i ruoli degli utenti che interagiscono con il sistema; questi attori possono anche essere considerati sottosistemi. Un attore può essere coinvolto in uno o più casi d'uso, e allo stesso modo, un caso d'uso può coinvolgere uno o più attori. Tuttavia, ogni caso d'uso ha un attore principale designato.

Stake holders

Il gestionale che abbiamo creato prevede due tipi di utenti:

- **il condomino**, colui che vive all'interno del condominio o ha una casa di proprietà nello stabile. L'utente, che a seguito del login, riesce ad interagire con una parte di tutto il gestionale, ossia quello che gli è permesso visualizzare
- **l'amministratore di condominio**, la persona preposta ad occuparsi dell'effettiva gestione condominiale. Questo utente, a seguito del login, può, invece, visualizzare la completezza delle funzionalità che il gestionale fornisce.

Specifica dei requisiti funzionali

Per ogni attore, di seguito è riportato un elenco completo dei requisiti funzionali.

Condomino

- Registrarsi e loggarsi al sito
- Effettuare, modificare ed eliminare pagamenti
- Visualizzare i progetti futuri previsti per il condominio
- Visualizzare i pdf dei bilanci che sono stati caricati
- Creare notifiche come lamentele, comunicazioni o altro
- Visualizzare le riunioni condominiali

Amministratore

- Registrarsi e loggarsi al sito
- Effettuare, modificare ed eliminare pagamenti
- Crea ed eliminare i progetti
- Visualizzare i progetti futuri previsti per il condominio
- Caricare i pdf dei bilanci e visualizzarli
- Visualizzare le notifiche che gli arrivano
- Programmare le nuove riunioni condominiali e visualizzarle

Specifica dei requisiti non funzionali

Alcuni requisiti non funzionali possono essere l'implementazione di una grafica minimalista ma efficace per permettere a tutti gli utenti di potersi avvicinare al sito e una miglior gestione della sicurezza dei dati sensibili.

Decisioni progettuali

Framework

La scelta di non utilizzare un framework è stata condivisa per favorire innanzitutto un'applicazione meno pesante e che ci permettesse di includere solo quello di cui avevamo bisogno. Questo ci consentiva, inoltre, di avere una maggiore flessibilità durante la progettazione e un controllo completo del codice.

API

La scelta di usare degli endpoint API invece che un'architettura client-server tradizionale per rendere flessibile l'interfaccia utente, come detto inizialmente, l'idea è di poter sviluppare la nostra web app, in un futuro, come app mobile. L'utilizzo delle API, infatti permette anche un'interoperabilità tra diversi linguaggi e piattaforme e la gestione divisa tra back-end e front-end.

Progettazione

Endpoint API

Il file `server.js` contiene la configurazione e la definizione degli endpoint per un server Node.js che fornisce un'API REST per un'applicazione web. Di seguito, il ragionamento generale dietro agli endpoint e alle API nel file:

Configurazione e Dipendenze

Inizialmente, il file dichiara le dipendenze necessarie come Express.js, bcrypt, jwt, mysql2, multer, ecc. Queste dipendenze sono utilizzate per gestire diverse funzionalità dell'applicazione, come l'autenticazione, la gestione dei file, il routing, e la connessione al database.

Configurazione di express-session

Viene configurata la gestione delle sessioni per gestire l'autenticazione degli utenti e memorizzare le informazioni di sessione.

Configurazione del middleware

Vengono configurati i middleware di Express, come `express.urlencoded` e `express.json`, per il parsing dei dati delle richieste in formato URL-encoded e JSON.

Gestione dei file statici

Viene configurata la gestione dei file statici per le directory `/pdf` e `/images` per servire i file PDF e le immagini.

Endpoint API

Vengono definiti vari endpoint API tramite il metodo `app.get` e `app.post`. Questi endpoint gestiscono richieste HTTP per diverse funzionalità dell'applicazione, come ottenere dati, autenticazione, gestione dei file, inserimento, aggiornamento e eliminazione di dati nel database.

Connessione al database

Viene stabilita la connessione al database MySQL utilizzando i dettagli di configurazione forniti nel file `config.json`.

Funzioni per il recupero dei dati dal database

Sono presenti diverse funzioni che eseguono query al database per recuperare dati come condomini, riunioni, pagamenti, notifiche, progetti futuri e utenti. Questi dati vengono poi restituiti come risposta alle richieste API.

Endpoint per la gestione delle riunioni e delle pagine

Ci sono endpoint specifici per gestire le riunioni, come la creazione, l'aggiornamento e l'eliminazione delle riunioni. Inoltre, ci sono endpoint per ottenere i dati delle pagine web corrispondenti a ciascuna funzionalità dell'applicazione.

In sintesi, il file `server.js` fornisce un server web che gestisce diverse funzionalità di un'applicazione, tra cui autenticazione, gestione dei file, recupero e gestione dei dati dal database e routing delle pagine web. Gli endpoint API vengono utilizzati per comunicare con il frontend dell'applicazione.

App.get('/home', (req, res) => {...})	
descrizione	Questo endpoint gestisce le richieste GET provenienti dal client e invia un file HTML specifico al client in risposta.
Utilizzo	Questo endpoint viene utilizzato per fornire al client il contenuto di un file HTML situato in una directory specifica. Quando un client invia una richiesta GET a /home, il server risponderà inviando il file HTML desiderato.
Parametri	<p>req: L'oggetto di richiesta (request) rappresentante la richiesta del client.</p> <p>res: L'oggetto di risposta (response) utilizzato per inviare dati al client.</p> <p>directory: Una variabile che rappresenta il percorso della directory contenente il file HTML desiderato.</p>
Risposta	L'endpoint /home risponde inviando il file HTML specificato tramite la funzione res.sendFile(). Il file viene inviato come risposta al client.

Questo è l'algoritmo sviluppato per la fase di login:

App.get('/login, (req, res) => {...}	
descrizione	L'endpoint /login è una parte critica di un server Node.js che utilizza il framework Express.js e si occupa dell'autenticazione degli utenti. Questo endpoint gestisce le richieste POST per l'accesso degli utenti al sistema, verificando le loro credenziali rispetto ai dati memorizzati nel database.
Utilizzo	Questo endpoint viene utilizzato per consentire agli utenti di accedere al sistema. Gli utenti inviano una richiesta POST contenente le loro credenziali (email e password) al server. Il server verifica queste credenziali rispetto ai dati memorizzati nel database e risponde con una pagina appropriata in base al ruolo dell'utente (Utente o Altro).
Parametri	<p>req: L'oggetto di richiesta (request) contenente le credenziali inviate dall'utente.</p> <p>res: L'oggetto di risposta (response) utilizzato per inviare dati al client.</p> <p>email: L'indirizzo email inviato dall'utente per l'autenticazione.</p> <p>password: La password inviata dall'utente per l'autenticazione.</p> <p>db: Riferimento al database utilizzato per eseguire query.</p>
Risposta	L'endpoint /login risponde in base all'esito dell'autenticazione dell'utente: Se l'utente è autenticato con successo, viene reindirizzato a una pagina appropriata in base al ruolo. Se le credenziali sono invalide, viene restituito un errore con stato 401 (Non

	<p>autorizzato) indicando "Credenziali non valide."</p> <p>In caso di errori durante l'autenticazione o l'accesso al database, viene restituito un errore con stato 500 (Errore del server) e un messaggio "Errore durante il login."</p>
--	---

Invece un esempio di endpoint con metodo POST:

App.post('/insert_payment, (req, res) => {...}	
descrizione	L'endpoint /login è una parte critica di un server Node.js che utilizza il framework Express.js e si occupa dell'autenticazione degli utenti. Questo endpoint gestisce le richieste POST per l'accesso degli utenti al sistema, verificando le loro credenziali rispetto ai dati memorizzati nel database.
Utilizzo	Questo endpoint viene utilizzato per consentire agli utenti di inserire nuovi pagamenti nel sistema. Il client invia una richiesta POST contenente i dati del pagamento, come la data, la descrizione, il codice identificativo e l'importo. Il server verifica e registra questi dati nel database.
Parametri	<p>req: L'oggetto di richiesta (request) contenente i dati del pagamento inviati dal client.</p> <p>res: L'oggetto di risposta (response) utilizzato per inviare dati al client.</p> <p>data: La data del pagamento.</p> <p>descrizione: Una breve descrizione del pagamento.</p> <p>codice_identificativo: Un codice identificativo univoco per il pagamento.</p> <p>prezzo: L'importo del pagamento.</p> <p>condominio_id: L'ID del condominio a cui è associato il pagamento (nel tuo esempio, è impostato su 1).</p> <p>utente_id: L'ID dell'utente che effettua il pagamento (nel tuo esempio, è impostato su 1).</p>

Risposta	<p>L'endpoint /insert_payment risponde in base all'esito dell'inserimento del pagamento:</p> <p>Se il pagamento viene inserito con successo, il server restituisce uno stato 201 (Creato) e un messaggio "Pagamento inserito con successo."</p> <p>In caso di errori durante l'inserimento o l'accesso al database, il server restituirà uno stato 500 (Errore del server) e un messaggio "Errore durante l'inserimento del pagamento."</p>
-----------------	---

L'algoritmo usato molteplici volte per avere dati nel client dal database:

App.post('/insert_', (req, res) => {...}	
descrizione	L'endpoint /login è una parte critica di un server Node.js che utilizza il framework Express.js e si occupa dell'autenticazione degli utenti. Questo endpoint gestisce le richieste POST per l'accesso degli utenti al sistema, verificando le loro credenziali rispetto ai dati memorizzati nel database.
Utilizzo	Questo endpoint viene utilizzato per consentire agli utenti di inserire nuovi pagamenti nel sistema. Il client invia una richiesta POST contenente i dati del pagamento, come la data, la descrizione, il codice identificativo e l'importo. Il server verifica e registra questi dati nel database.
Parametri	<p>req: L'oggetto di richiesta (request) contenente i dati del pagamento inviati dal client.</p> <p>res: L'oggetto di risposta (response) utilizzato per inviare dati al client.</p> <p>data: La data del pagamento.</p> <p>descrizione: Una breve descrizione del pagamento.</p> <p>codice_identificativo: Un codice identificativo univoco per il pagamento.</p> <p>prezzo: L'importo del pagamento.</p>

	condominio_id: L'ID del condominio a cui è associato il pagamento (nel tuo esempio, è impostato su 1). utente_id: L'ID dell'utente che effettua il pagamento (nel tuo esempio, è impostato su 1).
Risposta	L'endpoint /insert_payment risponde in base all'esito dell'inserimento del pagamento: Se il pagamento viene inserito con successo, il server restituisce uno stato 201 (Creato) e un messaggio "Pagamento inserito con successo." In caso di errori durante l'inserimento o l'accesso al database, il server restituirà uno stato 500 (Errore del server) e un messaggio "Errore durante l'inserimento del pagamento."

Tecnologie utilizzate

Per lo sviluppo dell'applicativo è stato utilizzato l'editor di testo Visual Studio Code integrato con plug-in e varie estensioni che si è verificato la scelta migliore per come è stato affrontato il progetto. Per migliorare la gestione dei commit, in seguito ad alcuni confronti coi colleghi, è stato integrato nel progetto ed utilizzato GitKraken per visualizzare l'interezza dei cambiamenti apportati. Come piattaforma di hosting è stato scelto GitHub che ha facilitato la collaborazione tra i due partner.

Come già specificato, non si è voluto utilizzare un framework specifico, ma per la parte grafica è stata implementata con Html e CSS per avere un pieno controllo sul codice.

Per assicurare la massima sicurezza delle password archiviate nell'applicazione, è stata adottata la tecnologia open-source Bcrypt. Questa soluzione ha dimostrato di offrire un'efficace crittografia e decrittografia delle password, garantendo un elevato livello di protezione per i dati sensibili.

Per quanto concerne la comunicazione tra il client e il database, è stata scelta l'implementazione utilizzando NodeJS. Questa decisione è stata presa perché NodeJS ha agevolato l'implementazione di una comunicazione fluida ed efficiente tra le due entità, semplificando il flusso dei dati e garantendo un'esperienza utente ottimale.

L'adozione di queste tecnologie e strumenti ha giocato un ruolo cruciale nell'assicurare un processo di sviluppo efficiente e nella creazione di una robusta

infrastruttura per l'applicazione, il che ha permesso di conseguire un prodotto finale caratterizzato da una notevole qualità e un insieme di funzionalità estremamente avanzate.

Base dati

Il database, che è stato creato per gestire quest'applicativo web, è stato pensato con sette tabelle tutte esplicative, tranne Associazione. Quest'ultima è stata creata per poter gestire una relazione n-n tra i condomini e gli utenti e ci ha permesso di implementare la pagina Famiglie a cui si può accedere.

```
DROP DATABASE IF EXISTS database_condominium;
```

```
CREATE DATABASE database_condominium;
```

```
USE database_condominium;
```

```
CREATE TABLE Utente (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    nome VARCHAR(255) NOT NULL,  
    cognome VARCHAR(255) NOT NULL,  
    email VARCHAR(255) NOT NULL,  
    password VARCHAR(255) NOT NULL,  
    p_iva VARCHAR(255),  
    ruolo ENUM('Amministratore','Utente')  
);
```

```
CREATE TABLE Condominio (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    nome VARCHAR(255) NOT NULL,  
    indirizzo VARCHAR(255) NOT NULL,  
    n_piani int NOT NULL,  
    n_appartamenti int NOT NULL,  
    cantine boolean,
```

```
garage boolean,  
utente_id INT NOT NULL,  
FOREIGN KEY (utente_id) REFERENCES Utente(id)  
);
```

```
CREATE TABLE Associazione (  
    ass_id INT PRIMARY KEY AUTO_INCREMENT,  
    condominio_id INT NOT NULL,  
    utente_id INT NOT NULL,  
    FOREIGN KEY(utente_id) REFERENCES Utente(id),  
    FOREIGN KEY (condominio_id) REFERENCES Condominio(id)  
);
```

```
CREATE TABLE Pagamento (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    data DATE NOT NULL,  
    descrizione VARCHAR(255) NOT NULL,  
    codice_identificativo int(3) NOT NULL,  
    importo DECIMAL(10, 2) NOT NULL,  
    condominio_id INT,  
    utente_id INT,  
    FOREIGN KEY (condominio_id) REFERENCES Condominio(id),  
    FOREIGN KEY (utente_id) REFERENCES Utente(id)  
);
```











```
CREATE TABLE Notifica (  
    id INT PRIMARY KEY AUTO_INCREMENT,
```










```
tipologia ENUM('Lamentela','Revisione','Altro') NOT NULL,  
data DATE NOT NULL,  
descrizione VARCHAR(255) NOT NULL,  
utente_id INT,  
FOREIGN KEY (utente_id) REFERENCES Utente(id)  
);
```









```
CREATE TABLE Riunione (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    data DATE NOT NULL,  
    ora TIME,  
    titolo VARCHAR(255) NOT NULL,  
    descrizione VARCHAR(255) NOT NULL,  
    utente_id INT NOT NULL,  
    FOREIGN KEY (utente_id) REFERENCES Utente(id)  
);
```









```
CREATE TABLE Progetto_Futuro (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    data_inizio DATE NOT NULL,  
    data_fine DATE NOT NULL,  
    nome VARCHAR(255) NOT NULL,  
    descrizione VARCHAR(255) NOT NULL,  
    condominio_id INT NOT NULL,  
    FOREIGN KEY (condominio_id) REFERENCES Condominio(id)  
);
```










Diagramma delle classi






 	database_condominium Condominio
	id : int(11)
	nome : varchar(255)
	indirizzo : varchar(255)
	n_piani : int(11)
	n_appartamenti : int(11)
	cantine : tinyint(1)
	garage : tinyint(1)
	utente_id : int(11)








 	database_condominium Pagamento
	id : int(11)
	data : date
	descrizione : varchar(255)
	codice_identificativo : int(3)
	importo : decimal(10,2)
	condominio_id : int(11)
	utente_id : int(11)

 	database_condominium Riunione
	id : int(11)
	data : date
	ora : time
	titolo : varchar(255)
	descrizione : varchar(255)
	utente_id : int(11)

 	database_condominium Progetto_Futuro
	id : int(11)
	data_inizio : date
	data_fine : date
	nome : varchar(255)
	descrizione : varchar(255)
	condominio_id : int(11)

 	database_condominium Utente
	id : int(11)
	nome : varchar(255)
	cognome : varchar(255)
	email : varchar(255)
	password : varchar(255)
	p_iva : varchar(255)
	ruolo : enum('Amministratore','Utente')

 	database_condominium Associazione
	ass_id : int(11)
	condominio_id : int(11)
	utente_id : int(11)

 	database_condominium Notifica
	id : int(11)
	tipologia : enum('Lamentela','Revisione','Altro')
	data : date
	descrizione : varchar(255)
	utente_id : int(11)

Processo di sviluppo

Il processo di sviluppo è stato sviluppato in coppia con una pair programming. Inizialmente, si è scelto il progetto da sviluppare, dopodiché sono stati valutati i requisiti da sviluppare, la creazione di un database completo ed efficiente, per poi proseguire con lo sviluppo vero e proprio del codice.

La programmazione dello sviluppo è stata di tipo progressivo, inserendo di giorno in giorno nuove idee, oltre a quelle che erano state prefissate.

La scrittura del codice ha affrontato alcune difficoltà superate con determinazione e in alcuni casi con l'aiuto di alcuni colleghi di corso per comprendere a pieno l'errore e trovare la soluzione migliore.

L'intero percorso di sviluppo e programmazione ha portato ad un applicativo web minimalista e soddisfacente per le premesse che era state fatte.

Conclusione

Nel corso di questo progetto, abbiamo intrapreso un viaggio nell'ambito della digitalizzazione, cercando di colmare il divario tra la gestione condominiale tradizionale e le potenzialità offerte dalle soluzioni digitali. Abbiamo sviluppato un sito web personalizzato con l'obiettivo di semplificare e migliorare le comunicazioni tra i condomini e l'amministratore del condominio, rendendo l'esperienza condominiale più efficiente e trasparente.

I nostri sforzi si sono concentrati su una serie di obiettivi chiave, tra cui la creazione di un sistema di notifiche ben strutturato, l'accesso facilitato a documenti importanti come i verbali delle riunioni e i bilanci passati, e la gestione autonoma dei pagamenti condominiali. Abbiamo lavorato per garantire che ogni membro della comunità condominiale possa partecipare attivamente e rimanere sempre informato su ciò che accade nella propria comunità.

In definitiva, siamo orgogliosi del risultato raggiunto con questo progetto e riteniamo che il nostro sito web possa contribuire in modo significativo a migliorare la gestione condominiale e l'esperienza dei condomini. Guardiamo al futuro con la prospettiva di espandere questa piattaforma in un'applicazione mobile, continuando a innovare e adattarci alle esigenze della nostra comunità.

Ringraziamo i nostri compagni di corso per aver testato e averci supportato nelle nostre decisioni.