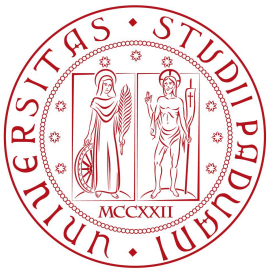
Università degli Studi di Padova
Dipartimento di Scienze Statistiche
Corso di Laurea Triennale in

Statistica per le Tecnologie e le Scienze



Progettazione della base di dati e sviluppo in Laravel di un gestionale con API per Sistemisti Informatici

Relatore: Prof. Marco Dussin
Dipartimento di Scienze Statistiche

Laureando: Giacomo Cappella
Matricola n. 1150316

Anno Accademico 2023/2024

Ai miei genitori,
con profonda gratitudine

Indice

Introduzione

I contratti di prestazione occasionale sono spesso il punto di partenza di chi vuole iniziare un proprio percorso di lavoro autonomo.

In questo elaborato viene preso in esame il caso di due giovani, entrambi sistemisti informatici attualmente impiegati come dipendenti presso un'Istituzione Scolastica della provincia di Treviso, i quali hanno deciso di avviare una collaborazione e di iniziare a seguire più clienti dal punto di vista informatico dando vita a "EdGiTech", prototipo di una futura società i cui clienti sono principalmente piccole-medie aziende ed esercizi commerciali.

Da qui nasce la necessità di adottare un *software* cucito su misura che permetta il controllo funzionale dei *ticket* di lavorazione, la gestione delle anagrafiche clienti e la creazione automatica delle ricevute di prestazione occasionale a seguito del termine di una lavorazione. Dopo opportune ricerche e test di applicativi già sul mercato, è stata ritenuta necessaria la costruzione di un gestionale personalizzato *end-to-end*, in quanto gli applicativi già sviluppati non supportano tutte le funzionalità richieste, oltre a presentare un costo di gestione mensile per ora insostenibile.

Questo elaborato quindi descrive la progettazione e presenta alcuni dettagli salienti relativi allo sviluppo di EdGiGest, un *software* gestionale attraverso il framework PHP gratuito "Laravel" a cui viene integrato, attraverso delle REST API, il *software* gratuito "Clockify" per la gestione dei *ticket* di assistenza. Tale scelta è stata dettata dalla possibilità di alleggerire il codice sfruttando le potenzialità di un *software* web già collaudato, funzionale per le esigenze di EdGiTech e gratuito nella sua versione base.

L'elaborato segue la struttura dettata da quanto progettato e implementato per la creazione del gestionale, nello specifico:

- nel primo capitolo viene effettuata l'analisi dei requisiti, necessaria per descrivere gli obiettivi e le richieste da parte del committente;
- nel secondo capitolo sono descritte le tecnologie utilizzate nel progetto per la creazione del *software*;
- nel terzo capitolo viene esposta la struttura della base di dati progettata e implementata, necessaria per il funzionamento del gestionale;
- nel quarto capitolo viene presentato il *software* costruito in tutte le sue funzionalità e viene approfondita una sezione significativa di cui sono discussi i meccanismi di gestione delle informazioni.

Capitolo 1

Analisi dei requisiti

L'analisi dei requisiti è una fase fondamentale nel processo di sviluppo di un *software* [5]. Questa attività consiste nella raccolta di tutte le richieste da parte del committente, che vengono successivamente analizzate e riorganizzate in un documento dettagliato. Tale documento descrive tutte le funzionalità richieste, insieme ai vincoli imposti.

Questa fase è cruciale perché in essa vengono definiti tutti gli obiettivi del progetto, garantendo che il risultato finale soddisfi le richieste iniziali del mandante.

È importante sottolineare che tale fase è soggetta a continui aggiornamenti durante l'intero ciclo di sviluppo del progetto [6], poiché i requisiti possono variare sia da parte del committente sia da parte degli sviluppatori. Ciò può includere, ad esempio, l'aggiunta di nuove funzionalità, l'implementazione di sistemi di sicurezza o l'incremento di prestazioni rispetto a quanto inizialmente previsto.

La seguente analisi dei requisiti comprende inizialmente un glossario in cui vengono definiti i termini tecnici utilizzati in tutto l'elaborato e successivamente una lista di requisiti ritenuti fondamentali per la completa funzionalità del gestionale. Infine è presente una rassegna di tutti i casi d'uso del gestionale, in cui si esplorano le funzionalità richieste nel dettaglio.

1.1 Glossario dei termini

- *Framework*: raccolta di componenti *software* che rendono più efficiente lo sviluppo di nuove applicazioni. I *framework* possono anche definire e applicare determinate regole di architettura *software* o processi aziendali, in modo che sia possibile sviluppare nuove applicazioni standardizzate[1].

- ORM (Object Relational Mapping): tecnica per convertire i dati tra database e oggetti in memoria quando si utilizzano linguaggi di programmazione orientati agli oggetti [1].

- API (*Application Programming Interface*): insieme di regole per la comunicazione tra computer, *software* o tra diversi componenti di *software*. Consentono agli sviluppatori di utilizzare la pratica del riuso del codice risparmiando tempo e risorse[2].

- *Ticket*: lavorazione associata ad un cliente che può comprendere una o più attività. In figura 1.1, ad esempio, è possibile vedere lo *screenshot* di una interfaccia per la creazione di un *ticket*, che si riferisce ad una richiesta di installazione di un nuovo componente *hardware*.

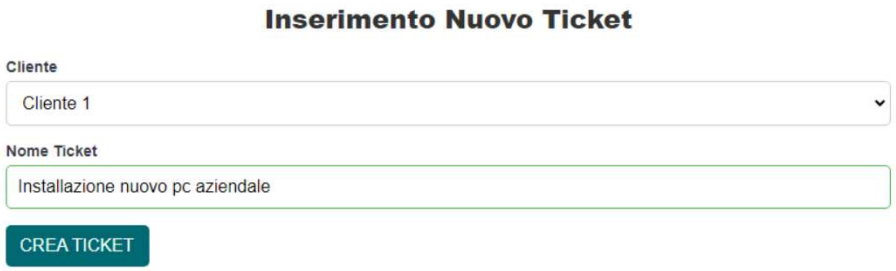


Figura 1.1: Creazione di un *ticket*

[1] https://aws.amazon.com/it/what-is/framework/

[2] https://www.treccani.it/enciclopedia/api_(Lessico-del-XXI-Secolo)/

- Attività: lavorazione specifica appartenente ad un *ticket*. Per ogni attività sono associati i tempi della lavorazione utili ai fini del pagamento della prestazione.

- Clockify: applicazione web gratuita che consente di tenere traccia delle ore di lavoro associandole a progetti e clienti.

- Prestazione occasionale: tipo di contratto di lavoro caratterizzato dalla saltuarietà e dalla limitata entità economica, il quale dà la possibilità di poter svolgere, a fianco di un lavoro subordinato, delle prestazioni lavorative fino a 5.000,00 euro lordi annui, offrendo delle agevolazioni rispetto il pagamento delle tasse dovute allo Stato e senza la necessità di dover aprire una Partita IVA[3].

1.2 Attori

- **Utente non autenticato**: utente registrato nel sistema che non ha ancora eseguito l'accesso. A seconda del tipo di utente profilato, verrà dato l'accesso a sezioni specifiche del gestionale.

- **Utente autenticato**: utente generico che ha completato la procedura di autenticazione nel sistema. Può visualizzare e modificare il proprio profilo.

- **Sistemista**: utente che possiede il ruolo di amministratore, ovvero con i permessi più elevati. Può visualizzare una *dashboard* contenente tutte le informazioni primarie, aggiungere e modificare le anagrafiche cliente, aggiungere e modificare i *tickets*. Può inoltre gestire in modo completo i passaggi di stato di un *ticket* (descritti al paragrafo 1.3) e aggiungere, modificare ed eliminare le attività associate. Infine può gestire la creazione e l'invio delle ricevute di prestazione occasionale. Il compito del sistemista informatico consiste

[3] `https://www.lavoro.gov.it/temi-e-priorita/rapporti-di-lavoro-e-relazioni-industriali/focus-on/disciplina-rapporto-lavoro/pagine/prestazioni-occasionali/`

nella progettazione e gestione dell'intera infrastruttura informatica dell'azienda nonché della sicurezza della stessa, attraverso l'utilizzo di *hardware* e *software* dedicati come *firewall* e *antivirus*. Inoltre, garantisce il corretto funzionamento e aggiornamento dei dispositivi di rete attraverso il loro monitoraggio e la sicurezza degli archivi digitali aziendali, comprendendo adeguati permessi, meccanismi di *backup* e di *disaster recovery*.

- **Cliente**: utente che ha la possibilità di visualizzare una *dashboard* personalizzata con la sola visualizzazione di tutti i *tickets* ad esso associati.

1.3 Descrizione del progetto

Il gestionale deve permettere ai tecnici sistemisti che lo utilizzano di poter avere una gestione completa dei *tickets* di lavorazione. Per ogni cliente possono essere effettuate più lavorazioni, come nuove installazioni di dispositivi o manutenzioni programmate, al termine delle quali è necessario rendicontarne quanto effettivamente svolto in dettaglio tecnico e in termini di tempo. Al termine di una lavorazione, ai fini del pagamento della prestazione, viene prodotto un documento denominato "Ricevuta di prestazione occasionale" nella quale viene riportata, oltre ai dati del prestatore occasionale e del cliente, la quantità di ore effettivamente svolte con il relativo importo lordo e netto da corrispondere.

Il primo passo spetta nella maggior parte di casi al cliente: quando sorge un problema informatico o vi è la necessità di un intervento tecnico per una nuova installazione, viene contattato il supporto tecnico di EdGiTech, che riceve la chiamata e cerca di comprendere nel dettaglio le esigenze del cliente. Successivamente il supporto tecnico deve poter accedere al gestionale per creare un nuovo *ticket* e assegnarlo al cliente in questione. Quindi, vengono programmati e creati a sistema uno o più interventi tecnici, ovvero le attività, che possono essere eseguiti fisicamente dal cliente oppure da remoto. Una volta terminata l'attività

il sistemista è tenuto a compilare un'apposita area del gestionale in cui inserisce la data e il range orario in cui questa è stata svolta, assieme ad una descrizione tecnica di tutto ciò che è stato fatto. Questa è fondamentale per due motivi principali, in primo luogo per dare un riscontro al cliente dell'avvenuta lavorazione e per tenere traccia delle problematiche risolte in quanto potrebbero tornare utili per altre lavorazioni simili.

Come si può notare dalla figura 1.2, la vita di un *ticket* è racchiusa entro tre stati: "aperto", "sospeso" e "chiuso". Nel momento in cui un *ticket* viene creato, di default viene contrassegnato come "aperto" e rimane tale fino a che non vengono concluse tutte le attività ad esso associate. Un *ticket* potrebbe essere segnato come "sospeso", invece, quando non si può procedere immediatamente alla sua risoluzione e non si può chiudere perché il problema non è risolto. Nello specifico potrebbe accadere di essere in attesa di un *feedback* del cliente sulla risoluzione del problema, oppure potrebbe dipendere dalla risoluzione di un altro *ticket*.

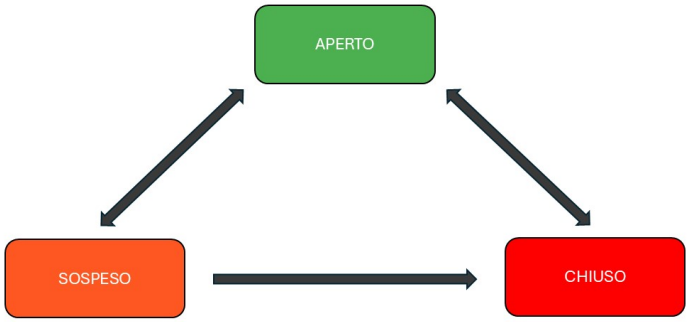


Figura 1.2: Passaggi di stato di un *ticket*

Quando tutte le lavorazioni sono terminate il sistemista può procedere con la chiusura del *ticket* di lavorazione. Prima del passaggio allo stato di "chiuso" il sistema deve dare la possibilità di far decidere al tecnico se inviare una email automatizzata con il resoconto delle attività svolte al cliente.

L'iter di un *ticket* termina con la costruzione del documento fiscale necessario ai fini del pagamento della prestazione ovvero, nell'esempio trattato, la ricevuta di prestazione occasionale.

Il gestionale deve poter gestire anche questa funzionalità, facendo scegliere al tecnico uno o più *tickets* da inserire nella ricevuta per un dato cliente selezionato. Successivamente, il programma calcolerà il totale delle ore dei *tickets* selezionati e costruirà il documento contenente tutti i dati del tecnico e del cliente, nonché degli importi lordi e netti, che verrà trasmesso al cliente attraverso una funzione automatizzata.

Infine, il sistema deve dare la possibilità di gestire l'anagrafica dei clienti, essenziale per l'invio delle email automatizzate, delle ricevute della prestazione occasionale e per l'associazione dei *tickets* di lavorazione.

1.4 Elenco dei requisiti

Di seguito viene fornito l'elenco di tutti i requisiti dell'applicazione, distinti tra requisiti funzionali, ovvero le operazioni che il gestionale deve compiere attraverso l'input umano, e non funzionali, incentrati su come il sistema deve essere implementato.

Inoltre, ad ogni requisito viene attribuita una diversa necessità:

- obbligatorio se il requisito è fondamentale per la funzionalità del sistema e deve essere implementato;
- facoltativo se il requisito migliora l'esperienza di utilizzo e completa le funzionalità del gestionale, ma non è indispensabile.

Login

Requisito	Funzionale	Obbligatorio
Il gestionale deve poter permettere all'utente di autenticarsi e poter visualizzare il proprio profilo	Sì	No
Il gestionale deve poter far accedere gli utenti in maniera differenziata tra sistemisti e clienti	No	No
Il gestionale deve dare la possibilità agli utenti registrati di poter resettare la loro password inviando una email di reimpostazione all'indirizzo associato	No	No
Il gestionale deve offrire un sistema di doppia autenticazione (2FA) richiedendo agli utenti, oltre alla password, un codice di verifica temporaneo inviato tramite SMS o generato da un'app di autenticazione dedicata	No	No

Tabella 1.1: Requisiti *login*.

Dashboard

Requisito	**Funzionale**	**Obbligatorio**
I sistemisti visualizzeranno una *dashboard* che include tutte le funzionalità del gestionale mentre i clienti avranno la possibilità di visualizzare tutti i *tickets* a loro associati	No	No
Il gestionale deve dare la possibilità di visualizzare un menu principale che permetta una navigazione rapida alle sue diverse funzioni	Sì	Sì
Il gestionale deve mostrare in modo chiaro tutti i *tickets* aperti e sospesi per ogni cliente, accompagnati dai dettagli più importanti come il loro codice identificativo e la durata in ore	Sì	Sì
Tutte le operazioni principali devono essere accessibili in massimo 3 click dalla *dashboard*	No	Sì

Tabella 1.2: Requisiti *dashboard*.

Ticket

Requisito	**Funzionale**	**Obbligatorio**
Il gestionale deve permettere l'inserimento di un nuovo *ticket*, associandolo ad un cliente	Sì	Sì
Ogni *ticket* creato deve essere presente su Clockify e deve essere associato ad un cliente	Sì	Sì
Il gestionale deve permette la modifica dei dettagli di un *ticket*, i quali dovranno essere sincronizzati sul corrispondente di Clockify	Sì	Sì
Il gestionale deve dare la possibilità di modificare lo stato di un *ticket* da aperto, sospeso a chiuso e viceversa, sincronizzandolo con Clockify	Sì	Sì
Il gestionale deve permettere al sistemista che cambia lo stato del *ticket* in "chiuso" di scegliere se inviare una email automatica al cliente con il *report* dettagliato di tutte le attività ad esso associate	Sì	Sì

Tabella 1.3: Requisiti gestione *ticket*.

Attività

Requisito	Funzionale	Obbligatorio
Il gestionale deve permettere di visualizzare una lista dettagliata di tutte le attività collegate ad un *ticket*	Sì	Sì
Il gestionale deve permettere l'inserimento di una nuova attività associata ad un *ticket*, sincronizzandola su Clockify	Sì	Sì
Il gestionale deve permettere la modifica dei dettagli di un'attività, i quali dovranno essere sincronizzati sulla corrispondente di Clockify	Sì	Sì
Il gestionale deve permettere la cancellazione di un'attività	Sì	Sì

Tabella 1.4: Requisiti gestione attività.

Cliente

Requisito	Funzionale	Obbligatorio
Il gestionale deve dare la possibilità di aggiungere un nuovo cliente	Sì	Sì
Ogni cliente creato deve essere presente anche su Clockify	Sì	Sì
Il gestionale deve permettere la modifica dei dettagli di un cliente, i quali dovranno essere sincronizzati su Clockify	Sì	Sì

Tabella 1.5: Requisiti gestione cliente.

Ricevute

Requisito	Funzionale	Obbligatorio
Il gestionale deve dare la possibilità di avere un'area dedicata alla costruzione e all'invio delle ricevute di prestazione occasionale	Sì	Sì
Il gestionale deve permettere di scegliere un cliente al quale emettere la ricevuta	Sì	Sì
Il gestionale deve mostrare per ogni cliente una lista di *tickets* chiusi selezionabili ai fini della costruzione della ricevuta	Sì	Sì
Dopo aver selezionato i *tickets* oggetto di ricevuta il gestionale deve permettere la creazione del documento di ricevuta, mostrarlo in anteprima e dare la possibilità al sistemista di inviarlo tramite email automatica al cliente, con i propri riferimenti bancari al fine del pagamento della prestazione	Sì	Sì

Tabella 1.6: Requisiti gestione ricevute.

Compatibilità

Requisito	Funzionale	Obbligatorio
Il gestionale deve essere compatibile con i principali *browser* (es. Google Chrome, Firefox, Safari, Microsoft Edge)	No	No
Il gestionale deve garantire il corretto funzionamento su versioni desktop e mobile dei browser supportati	No	No
Deve essere possibile aggiungere nuovi moduli o funzionalità senza la necessità di modificare profondamente l'architettura esistente	No	No
Il gestionale deve supportare la connessione HTTPS per garantire la sicurezza dei dati trasmessi tra il client e il server, utilizzando certificati SSL/TLS aggiornati	No	No

Tabella 1.7: Requisiti di compatibilità.

1.5 Casi d'uso

Di seguito vengono descritti dettagliatamente tutti i casi d'uso relativi al gestionale oggetto di studio.

UC-1: Gestione autenticazione

UC-1.1: Login web

Attore primario: Utente non autenticato.

Descrizione: L'utente inserisce le proprie credenziali per autenticarsi e accedere al sistema.

Precondizioni: L'utente è registrato e ha credenziali valide.

Postcondizioni: L'utente è autenticato e può accedere alla *dashboard* di appartenenza.

Scenario principale:

1. L'utente avvia l'applicazione web.
2. L'utente inserisce le credenziali e conferma (UC-1.2 Inserimento credenziali).
3. Il sistema verifica le credenziali.
4. Se le credenziali sono corrette, l'utente accede alla *dashboard*.

Estensioni:

UC-1.2: Inserimento credenziali L'utente inserisce *username* e *password* nel *form* di login.

Scenari alternativi:

UC-1.3: Credenziali non valide Il sistema verifica le credenziali e rileva che sono errate.

UC-1.4: Logout

Attore primario: Utente autenticato.

Descrizione: L'utente si disconnette dal sistema.

Precondizioni: L'utente è autenticato.

Postcondizioni: L'utente viene disconnesso e ritorna alla pagina di *login*.

Scenario principale:

1. L'utente seleziona l'opzione *"Logout"* dal menu.
2. Il sistema chiude la sessione e mostra la schermata di *login*.

UC-2: Reset password

Attore primario: Utente non autenticato.

Descrizione: L'utente richiede il reset della *password* tramite email.

Precondizioni: L'utente ha un indirizzo email registrato nel sistema.

Postcondizioni: Il sistema invia un'email con un *link* per reimpostare la *password*.

Scenario principale:

1. L'utente seleziona "Password dimenticata" nella pagina di *login*.
2. Il sistema chiede l'inserimento dell'indirizzo email dell'utente.
3. L'utente inserisce l'email e il sistema invia un *link* per il *reset*.
4. L'utente riceve l'email e reimposta la password tramite il link.

Scenari alternativi: Email non trovata: Se l'email non è registrata nel sistema, l'utente riceve un messaggio di errore e può reinserire l'email corretta.

UC-3: Visualizzazione dashboard

Attore primario: Utente autenticato.

Descrizione: L'utente accede alla *dashboard*, che mostra il menu di navigazione e le informazioni principali in primo piano.

Precondizioni: L'utente è autenticato.

Postcondizioni: La *dashboard* viene visualizzata correttamente.

Scenario principale:

1. L'utente completa il *login* con successo.
2. Il sistema carica la *dashboard* come pagina principale, che contiene il menu e le informazioni principali. La *dashboard* caricata sarà differenziata a seconda della tipologia di utente che ha effettuato il *login*.

UC-4: Inserimento nuovo cliente

Attore primario: Sistemista.

Descrizione: Il sistemista inserisce un nuovo cliente all'interno del gestionale. I dati inseriti devono essere validati e sincronizzati con Clockify.

Precondizioni: L'utente è profilato come sistemista.

Postcondizioni: Un nuovo cliente viene creato nel database interno e su Clockify tramite API con i dati correttamente inseriti e validati.

Scenario principale:

1. Il sistemista accede alla sezione "Nuovo Cliente" dalla *dashboard*.
2. Il sistema visualizza il form per l'inserimento dei dati del cliente (UC-4.1).
3. Il sistemista inserisce i dati richiesti per il nuovo cliente (UC-4.1.1).
4. Il sistema valida i dati inseriti (UC-4.2).
5. Se la validazione ha successo, il cliente viene creato con successo e il sistema conferma l'operazione.

UC-4.1: Inserimento dati nuovo cliente

Attore primario: Sistemista.

Descrizione: Il sistemista inserisce i dati di un nuovo cliente nel gestionale.

Precondizioni: L'utente deve autenticato e profilato come sistemista.

Postcondizioni: I dati del cliente vengono inseriti e passano alla fase di validazione.

Scenario principale:

1. Il sistemista accede alla pagina "Nuovo cliente" dalla *dashboard*.

2. Il sistema presenta un form con campi vuoti per l'inserimento dei dati del cliente (UC-4.1.1).

3. Il sistemista inserisce i dati richiesti e conferma.

UC-4.1.1: Inserimento dei campi di un nuovo cliente

Attore primario: Sistemista.

Descrizione: Il sistemista inserisce i vari campi richiesti per la creazione del nuovo cliente.

Precondizioni: Il sistemista è nella schermata di inserimento di un nuovo cliente.

Postcondizioni: Il form viene compilato con i dati forniti dal sistemista.

Scenario principale:

1. Il sistemista vede un form con i seguenti campi:

   - ragione sociale del cliente (obbligatorio);
   - partita IVA / codice fiscale (obbligatorio);
   - indirizzo email dell'amministrazione (obbligatorio);
   - indirizzo email per invio rapportini *ticket* (obbligatorio);
   - contatto telefonico (obbligatorio);
   - via (obbligatorio);
   - civico (obbligatorio);
   - città (obbligatorio);
   - CAP (obbligatorio);
   - provincia (obbligatorio).

2. Il sistemista inserisce i valori nei campi.

3. Il sistemista conferma l'inserimento dei dati.

Estensioni:

UC-4.1.1.1: Validazione dei dati cliente

- Il sistema verifica che tutti i campi siano inseriti.
- Se uno di questi dati non è valido, il sistema ne richiede la correzione.

UC-4.2: Conferma creazione nuovo cliente

Attore primario: Sistemista.

Descrizione: Il sistema valida i dati e conferma la creazione del nuovo cliente.

Precondizioni: I dati del cliente sono stati inseriti correttamente nel form.

Postcondizioni: Il cliente viene creato nel database locale e viene creato un nuovo cliente su Clockify.

Scenario principale:

1. Il sistema verifica che tutti i dati obbligatori siano presenti e corretti (UC-4.1.1.1).
2. Se la validazione ha successo, il sistema crea il nuovo cliente.
3. Il sistema crea il nuovo cliente sul database locale e su Clockify e conferma l'avvenuta creazione.

UC-5: Inserimento nuovo ticket

UC-5.1: Scelta del cliente

Attore primario: Sistemista.

Descrizione: Il sistemista sceglie il cliente a cui associare il nuovo *ticket*.

Precondizioni: Il sistemista deve essere autenticato e il cliente deve essere registrato.

Postcondizioni: Il cliente è associato al *ticket*.

Scenario principale:

1. Il sistemista accede alla sezione "Nuovo Ticket".
2. Il sistema presenta un elenco dei clienti.
3. Il sistemista sceglie il cliente a cui associare il nuovo *ticket*.

UC-5.2: Inserimento nome ticket

Attore primario: Sistemista.

Descrizione: Il sistemista inserisce il nome del *ticket* in fase di creazione.

Precondizioni: Il sistemista ha scelto un cliente.

Postcondizioni: Il *ticket* viene creato su Clockify e associato al cliente.

Scenario principale:

1. Il sistemista inserisce il nome del *ticket*.
2. Il sistema crea il *ticket* su Clockify associato al cliente.

UC-6: Elenco dei tickets aperti e sospesi - sistemista

Attore primario: Sistemista.

Descrizione: L'utente visualizza l'elenco dei *ticket* aperti o sospesi associati per ogni cliente.

Precondizioni: L'utente deve essere autenticato e profilato come sistemista.

Postcondizioni: L'elenco dei *ticket* viene visualizzato correttamente.

Scenario principale:

1. Il sistemista effettua il login.
2. Il sistema visualizza nella pagina principale tutti i *tickets* aperti e sospesi suddivisi per cliente, con la possibilità di vederne i dettagli.

UC-7: Elenco dei tickets - cliente

Attore primario: Cliente.

Descrizione: L'utente visualizza l'elenco dei *tickets* ad esso associati.

Precondizioni: L'utente deve essere autenticato e profilato come cliente.

Postcondizioni: L'elenco dei *tickets* viene visualizzato correttamente.

Scenario principale:

1. Il cliente effettua il login.
2. Il sistema visualizza nella pagina principale tutti i *tickets* ad esso associati, con la possibilità di vederne i dettagli.

UC-8: Visualizza dettagli ticket

UC-8.1: Chiudi ticket

Attore primario: Sistemista.

Descrizione: Il sistemista chiude un *ticket*, con l'opzione di inviare una email automatica al cliente.

Precondizioni: Il sistemista deve essere autenticato e il *ticket* deve essere aperto o sospeso.

Postcondizioni: Il *ticket* viene chiuso e, se selezionato, viene inviato automaticamente via email al cliente.

Scenario principale:

1. Il sistemista visualizza i dettagli del *ticket*.
2. Il sistemista seleziona "Chiudi ticket".
3. Il sistema chiede se inviare una email al cliente.
    - **UC-8.1.1: Invio email sì**: Se il sistemista seleziona "Sì", viene inviata una email al cliente.
    - **UC-8.1.2: Invio email no**: Se il sistemista seleziona "No", il *ticket* viene chiuso senza invio di email.

UC-8.2: Sospendi ticket

Attore primario: Sistemista.

Descrizione: Il sistemista sospende un *ticket* in attesa di ulteriori informazioni.

Precondizioni: Il sistemista deve essere autenticato.

Postcondizioni: Il *ticket* viene sospeso.

UC-8.3: Riapri ticket

Attore primario: Sistemista.

Descrizione: Il sistemista riapre un *ticket* precedentemente chiuso o sospeso.

Precondizioni: Il *ticket* deve essere chiuso o sospeso.

Postcondizioni: Il *ticket* torna allo stato "aperto".

UC-8.4: Creazione nuova attività

Attore primario: Sistemista.

Descrizione: Il sistemista inserisce una nuova attività. I dati inseriti devono essere validati e sincronizzati con Clockify.

Precondizioni: L'utente è profilato come sistemista e sta visualizzando la pagina dei dettagli di un *ticket*.

Postcondizioni: Una nuova attività viene creata e associata al *ticket* di riferimento su Clockify tramite API con i dati correttamente inseriti e validati.

Scenario principale:

1. Il sistemista accede ai dettagli di un *ticket* dalla *dashboard*.
2. Il sistemista accede al form per l'inserimento dei dati dell'attività cliccando su "Aggiungi nuova attività".
3. Il sistema visualizza il form per l'inserimento dei dati della nuova attività (UC-8.4.1).
4. Il sistemista inserisce i dati richiesti per la nuova attività (UC-8.4.1.1).
5. Il sistema valida i dati inseriti (UC-8.4.2).
6. Se la validazione ha successo, l'attività viene creata con successo e il sistema conferma l'operazione.

UC-8.4.1: Inserimento dati dell'attività

Attore primario: Sistemista.

Descrizione: Il sistemista inserisce i dati della nuova attività legata ad un *ticket* nel gestionale.

Precondizioni:

- L'utente deve essere autenticato e profilato come sistemista.
- L'utente deve selezionare un *ticket* dalla *dashboard* e visualizzarne i dettagli.

Postcondizioni: I dati della nuova attività vengono inseriti e passano alla fase di validazione.

Scenario principale:

1. Il sistemista accede ai dettagli di un *ticket* dalla *dashboard*.
2. Il sistemista seleziona il pulsante "Aggiungi nuova attività" dalla pagina dei dettagli di un *ticket*.
3. Il sistema presenta un form con campi vuoti per l'inserimento dei dati della nuova attività (UC-8.4.1.1).
4. Il sistemista inserisce i dati richiesti e conferma.

UC-8.4.1.1: Inserimento dei campi di una nuova attività

Attore primario: Sistemista.

Descrizione: Il sistemista inserisce i campi richiesti per la creazione della nuova attività.

Precondizioni: Il sistemista è nella schermata di inserimento di una nuova attività.

Postcondizioni: Il form viene compilato con i dati forniti.

Scenario principale:

1. Il sistemista vede un form con i seguenti campi:
   - data e ora di inizio attività (obbligatorio);
   - data e ora di fine attività (obbligatorio);
   - descrizione dell'attività (obbligatorio).
2. Il sistemista inserisce i valori nei campi.
3. Il sistemista conferma l'inserimento dei dati.

Estensioni: **UC-8.4.1.1.1: Validazione dei dati cliente**

- Il sistema verifica che tutti i campi siano inseriti.
- Se uno di questi dati non è valido, il sistema richiede la correzione dei campi errati.

UC-8.4.2: Conferma creazione nuova attività

Attore primario: Sistemista.

Descrizione: Il sistema valida i dati e conferma la creazione della nuova attività.

Precondizioni: I dati della nuova attività sono stati inseriti correttamente nel form.

Postcondizioni: L'attività viene creata e associata al *ticket* su Clockify tramite API.

Scenario principale:

- Il sistema verifica che tutti i dati obbligatori siano presenti e corretti (UC-8.4.1.1.1).
- Se la validazione ha successo, il sistema crea la nuova attività.
- Il sistema crea la nuova attività Clockify e conferma l'avvenuta creazione.

Scenari alternativi:

Dati non validi: Se uno o più campi non superano la validazione, il sistema avvisa l'utente e richiede la correzione.

UC-8.5: Modifica attività

Attore primario: Sistemista.

Descrizione: Il sistemista modifica i dettagli di un'attività associata ad un *ticket*.

Precondizioni:

- Il sitemista è autenticato nel sistema.
- L'attività è già esistente e associata ad un *ticket*.

Postcondizioni: I dettagli dell'attività vengono aggiornati e sincronizzati con Clockify.

Scenario principale:

1. Il sistemista accede alla sezione dei dettagli del *ticket* dalla *dashboard* e seleziona l'attività che vuole modificare.
2. Il sistema mostra i dettagli attuali dell'attività.
3. Il sistemista modifica uno o più dei seguenti campi:
   - data e ora di inizio attività;
   - data e ora di fine attività;
   - descrizione dell'attività.
4. Il sistemista conferma la modifica.
5. Il sistema aggiorna l'attività su Clockify.

Estensioni: **UC-8.5.1: Validazione dei dati modificati**
Dopo aver confermato le modifiche, il sistema verifica che:

- la descrizione sia presente;
- le date e la durata siano valide e non in conflitto tra di loro;
- se ci sono problemi con la validità dei dati (es. data di fine precedente alla data di inizio), il sistema mostra un messaggio di errore e richiede la correzione.

Scenari alternativi: **UC-8.5.2: Modifica non salvata**
Se il sistemista annulla l'operazione o esce dalla schermata senza confermare, il sistema non salva le modifiche e i dati rimangono invariati.

UC-8.6: Elimina attività

Attore primario: Sistemista.

Descrizione: Il sistemista elimina un'attività associata ad un *ticket*.

Precondizioni:

- Il sistemista è autenticato nel sistema.
- L'attività è esistente e associata ad un *ticket*.

Postcondizioni: L'attività viene rimossa dal *ticket* e da Clockify.

Scenario principale:

1. Il sistemista accede alla sezione dei dettagli del *ticket* dalla *dashboard* e seleziona l'attività che vuole eliminare.
2. Il sistema chiede una conferma prima di procedere con l'eliminazione.
3. Il sistemista conferma l'eliminazione.
4. Il sistema rimuove l'attività da Clockify.
5. Il sistema aggiorna la vista del *ticket*, rimuovendo l'attività eliminata.

Estensioni: **UC-8.6.1: Conferma di eliminazione**

Il sistema richiede al sistemista di confermare l'operazione di eliminazione per prevenire cancellazioni accidentali. Se la conferma non viene data, l'attività non viene eliminata.

UC-9: Modifica ticket

Attore primario: Sistemista.

Descrizione: Il sistemista modifica il nome di un ticket esistente.

Precondizioni:

- Il sistemista è autenticato nel sistema.
- Il ticket deve esistere ed essere associato ad un cliente.

Postcondizioni: Il nome del ticket viene aggiornato nel sistema e sincronizzato con Clockify.

Scenario principale:

1. Il sistemista seleziona il *ticket* dalla *dashboard* e visualizza le informazioni, inclusi il cliente associato e il nome attuale.
2. Il sistemista modifica il nome del *ticket*.
3. Il sistemista conferma la modifica.
4. Il sistema valida il nuovo nome del *ticket* (UC-9.1).
5. Il sistema aggiorna il nome del *ticket* su Clockify (UC-9.2).
6. Il sistema riporta l'utente alla schermata di dettagli del *ticket*.

UC-9.1: Validazione del nome del ticket

Attore primario: Sistemista.

Descrizione: Il sistema verifica la correttezza del nome del *ticket*.

Precondizioni: Il sistemista deve essere nella schermata di modifica del *ticket*.

Postcondizioni: Il sistema procede alla modifica del nome del *ticket* su Clockify.

Scenario principale:

Quando il sistemista conferma l'inserimento e il sistema verifica che il nuovo nome del *ticket*:

- Non sia vuoto.

- Non superi il limite di caratteri consentiti (255 caratteri).

Se la validazione fallisce, il sistema mostra un messaggio di errore specifico e richiede una correzione.

UC-9.2: Conferma modifica nome ticket

Attore primario: Sistemista.

Descrizione: Il sistema conferma la modifica del nome del *ticket*.

Precondizioni: Il sistema valida il nome modificato del *ticket*.

Postcondizioni: Il sistema aggiorna correttamente il nome del *ticket*.

Scenario principale: Il sistema aggiorna correttamente il nome del *ticket* su Clockify e rimanda l'utente nella schermata con i dettagli del *ticket*.

Estensioni: UC-9.2.1: Modifica non confermata

Se il sistemista annulla l'operazione, il sistema non salva le modifiche e ritorna alla schermata di modifica.

UC-10: Modifica anagrafica cliente

Attore primario: Sistemista.

Descrizione: Il sistemista modifica i dettagli anagrafici di un cliente esistente nel gestionale.

Precondizioni:

- Il sistemista è autenticato.
- Il cliente deve essere già presente nel sistema.

Postcondizioni: I dettagli anagrafici del cliente vengono aggiornati nel database locale e sincronizzati con Clockify.

Scenario principale:

1. Il sistemista accede alla lista di tutti i clienti inseriti nel database.

2. Il sistemista seleziona il cliente da modificare.

3. Il sistema visualizza i dettagli attuali del cliente.

4. Il sistemista modifica uno o più campi anagrafici.

5. Il sistemista conferma le modifiche.

6. Il sistema valida i dati inseriti (UC-10.1: Validazione dei dati modificati anagrafica cliente).

7. Se i dati sono validi, il sistema aggiorna i dettagli nel database e li sincronizza con Clockify e riporta l'utente alla *dashboard* (UC-10.2: Conferma modifica dati anagrafica cliente).

UC-10.1: Validazione dei dati modificati anagrafica cliente

Attore primario: Sistemista.

Descrizione: Il sistema valida i dati modificati del cliente.

Precondizioni:

- Il sistemista è autenticato.
- Il cliente deve essere presente nel sistema.

Postcondizioni: I dettagli anagrafici del cliente vengono aggiornati nel database locale e sincronizzati con Clockify.

Scenario principale: Prima di salvare le modifiche effettuate, il sistema verifica che:

- Tutti i campi obbligatori siano stati compilati.
- I dati rispettino il formato corretto.
- Non esistano duplicati del campo "Partita IVA / Codice Fiscale", univoco nel sistema.

Se la validazione fallisce, il sistema mostra un messaggio di errore specifico e richiede la correzione dei dati.

Scenari alternativi: UC-10.1.1: Annullamento delle modifiche

Se il sistemista decide di annullare l'operazione prima di confermare, il sistema non salva le modifiche e ritorna alla schermata precedente, mantenendo i dati originali del cliente.

UC-10.2: Conferma modifica dati anagrafica cliente

Attore primario: Sistemista.

Descrizione: Il sistemista conferma la modifica dei dettagli anagrafici del cliente.

Precondizioni: Il sistema valida i dati anagrafici modificati del cliente.

Postcondizioni: Il sistema aggiorna correttamente i dati anagrafici del cliente.

Scenario principale: Il sistema aggiorna correttamente i dati anagrafici del cliente nel database locale e li sincronizza con Clockify. Al termine, rimanda l'utente alla *dashboard*.

UC-11: Creazione ricevuta

Attore primario: Sistemista.

Descrizione: Il sistemista crea una ricevuta di prestazione occasionale selezionando uno o più *tickets* chiusi di un cliente.

Precondizioni:

- Il sistemista deve essere autenticato.
- Il cliente selezionato deve avere almeno un *ticket* chiuso.

Postcondizioni:

- La ricevuta viene creata e, a scelta del sistemista, inviata via email al cliente.
- Il sistema notifica al sistemista che la ricevuta è stata creata correttamente.

Scenario principale:

1. Il sistemista accede all'area dedicata alla creazione delle ricevute di prestazione occasionale.
2. Il sistema mostra un elenco di clienti.
3. Il sistemista seleziona un cliente al quale associare la ricevuta.
4. Il sistema visualizza la lista dei *tickets* chiusi associati al cliente.
5. Il sistemista seleziona i *ticket* da includere nella ricevuta.
6. Il sistemista richiede l'anteprima della ricevuta.
7. Il sistema genera e mostra un'anteprima della ricevuta.
8. Il sistemista conferma la creazione della ricevuta.
9. Il sistema chiede all'utente se vuole inviare la ricevuta via email al cliente.
10. Il sistemista sceglie se inviare o meno la ricevuta via email.
11. Il sistema notifica al sistemista che la ricevuta è stata creata correttamente.

UC-11.1: Selezione cliente

Attore primario: Sistemista.

Descrizione: Il sistemista seleziona il cliente al quale associare la ricevuta di prestazione occasionale.

Precondizioni:

- Il sistemista deve essere autenticato.
- Deve esserci almeno un cliente presente nel sistema.

Postcondizioni: Il cliente viene selezionato e il sistema procede a mostrare i *ticket* chiusi associati a quel cliente.

Scenario principale:

1. Il sistemista accede all'area del gestionale dedicata alla creazione delle ricevute.
2. Il sistema mostra una lista di clienti.
3. Il sistemista seleziona il cliente desiderato.

UC-11.2: Selezione tickets

Attore primario: Sistemista.

Descrizione: Il sistemista seleziona uno o più *tickets* chiusi da includere nella ricevuta.

Precondizioni: Il cliente selezionato deve avere almeno un *ticket* chiuso associato.

Postcondizioni: I *tickets* selezionati vengono associati alla ricevuta in fase di creazione.

Scenario principale:

1. Il sistemista visualizza la lista dei *tickets* chiusi associati al cliente selezionato.
2. Il sistemista seleziona uno o più *tickets* da includere nella ricevuta.

UC-11.3: Anteprima ricevuta

Attore primario: Sistemista.

Descrizione: Il sistemista richiede un'anteprima della ricevuta prima di confermare la sua creazione.

Precondizioni: Il sistemista deve aver selezionato uno o più *tickets* chiusi da includere nella ricevuta.

Postcondizioni: Il sistema genera e mostra un'anteprima della ricevuta senza salvare i dati nel database.

Scenario principale:

1. Il sistemista seleziona l'opzione per richiedere l'anteprima della ricevuta.

2. Il sistema genera e mostra l'anteprima della ricevuta senza eseguire alcun salvataggio.

UC-11.4: Conferma creazione ricevuta

Attore primario: Sistemista.

Descrizione: Il sistemista conferma la creazione della ricevuta e decide se inviarla via email al cliente.

Precondizioni: Il sistemista deve aver visionato l'anteprima della ricevuta.

Postcondizioni: La ricevuta viene creata e, a seconda della scelta del sistemista, può essere inviata via email al cliente.

Scenario principale:

1. Il sistemista conferma la creazione della ricevuta.

2. Il sistema chiede all'utente se desidera inviare la ricevuta via email al cliente.

3. Il sistemista sceglie se inviare la ricevuta al cliente:

   - **UC-11.4.1: Invio email sì**: Se il sistemista seleziona "Sì", viene inviata una email al cliente con allegata la ricevuta.
   - **UC-11.4.2: Invio email no**: Se il sistemista seleziona "No", la ricevuta viene creata senza invio di email.

4. Il sistema notifica al sistemista che la ricevuta è stata creata correttamente.

UC-11.4.1: Invia ricevuta via email

Attore primario: Sistemista.

Descrizione: Il sistemista sceglie di inviare una copia della ricevuta via email al cliente.

Precondizioni:

- La ricevuta deve essere stata creata correttamente.
- L'indirizzo email del cliente deve essere presente nel sistema.

Postcondizioni: La ricevuta viene inviata via email al cliente.

Scenario principale:

1. Il sistemista sceglie di inviare la ricevuta via email.
2. Il sistema invia la ricevuta all'indirizzo email del cliente.
3. Il sistema conferma al sistemista che l'email è stata inviata con successo.

UC-11.4.2: Non inviare ricevuta via Email

Attore primario: Sistemista.

Descrizione: Il sistemista sceglie di non inviare la ricevuta via email al cliente.

Precondizioni: La ricevuta deve essere stata creata correttamente.

Postcondizioni: La ricevuta viene creata ma non inviata via email.

Scenario principale:

1. Il sistemista sceglie di non inviare la ricevuta via email.
2. Il sistema notifica al sistemista che la ricevuta è stata creata correttamente.

Capitolo 2

Presentazione delle tecnologie utilizzate

In questo capitolo vengono descritte le principali caratteristiche delle tecnologie e degli strumenti utilizzati per la costruzione del gestionale EdGiGest, assieme alle motivazioni che ne hanno guidato l'adozione.

2.1 Laravel

È stato ritenuto fondamentale costruire il gestionale attraverso un ambiente di sviluppo performante, completo ed intuitivo: per questo motivo è stato scelto Laravel[4].

Laravel[5], nato nel 2011 e attualmente nella sua versione 11, è un framework PHP[6] estremamente flessibile che semplifica lo sviluppo di applicazioni web grazie alla sua architettura, alla disponibilità di strumenti integrati e a una vasta *community* ben documentata.

Laravel segue l'architettura *Model-View-Controller* (MVC), che aiuta a mantenere il codice organizzato e separa chiaramente la logica dei dati e le interazioni con il *database* (*Model*), la presentazione dei dati all'utente (*View*), e il controllo ed elaborazione delle richieste in input (*Controller*) [3].

[4] `https://blog.dreamztech.com/why-laravel-is-the-most-popular-php-framework-in-2023/`

[5] `https://laravel.com/`

[6] `https://www.php.net/`

Il flusso di lavoro del modello MVC, illustrato nella figura 2.1, è il seguente:

- l'utente interagisce con l'interfaccia grafica (*View*).
- il *controller* gestisce l'*input* e ha il compito di aggiornare il *model*.
- il *model* modifica o recupera i dati dal *database* e li notifica al *controller*.
- il *controller* si occupa di aggiornare la *view* con i dati del *model*.

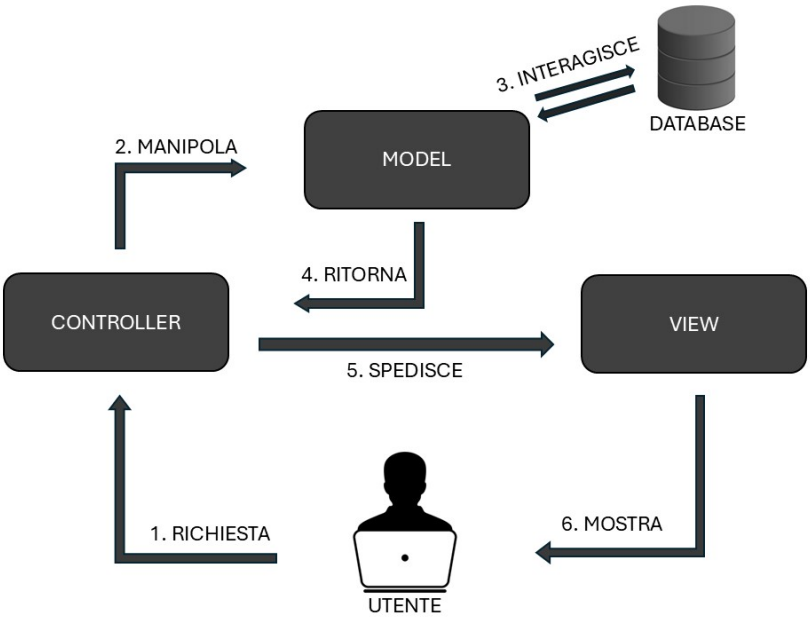


Figura 2.1: Flusso di lavoro del modello MVC.

Inoltre, integrato nel *framework* Laravel vi è Eloquent[7], il sistema di Object-Relational Mapping (ORM), progettato per semplificare l'interazione con i *database*. Eloquent fornisce un'interfaccia semplice e intuitiva per gestire i dati, consentendo di lavorare con delle *query* semplificate grazie a una sintassi intuitiva e fluida.

Altra funzionalità degna di nota in Laravel è Blade[8], il motore di *template* progettato per semplificare la creazione di interfacce utente di-

[7] `https://laravel.com/docs/11.x/eloquent/`
[8] `https://laravel.com/docs/11.x/blade/`

namiche, consentendo di scrivere codice PHP in modo pulito all'interno delle *view* e integrando facilmente i dati provenienti dai *controller*.

Infine si ritiene di valore sottolineare che Laravel integra le API REST in modo semplice ed efficiente, fornendo strumenti per gestire richieste HTTP e strutturare risposte JSON. Queste tecnologie web verranno descritte nel paragrafo successivo.

2.2 API REST

Le API (*Application Programming Interfaces*) sono strumenti fondamentali nello sviluppo *software* e permettono a diversi programmi di comunicare tra loro. Le API stabiliscono regole e protocolli che consentono a *software* differenti di interagire e condividere dati senza dover conoscere a vicenda i loro dettagli interni. In pratica, funzionano come un ponte tra sistemi, rendendo possibili interazioni efficienti tra applicazioni, sistemi operativi, *database*, servizi web e molto altro.

Un tipo particolarmente diffuso di API sono le API REST (*Representational State Transfer*). Queste ultime seguono uno stile architetturale che sfrutta i protocolli HTTP[9] per permettere alle applicazioni di inviare richieste e ricevere risposte tramite metodi standard come `GET, POST, PUT e DELETE`. Le API REST sono progettate per essere leggere, scalabili e facilmente utilizzabili su internet [4].

Ciò che distingue le API REST è legato all'utilizzo delle risorse, rappresentate da URI[10] (*Uniform Resource Identifier*) univoci, e sull'indipendenza tra client e server.

Il client può interagire con le risorse del server inviando richieste HTTP e la connessione avviene in questo modo:

- Quando viene fatta una richiesta API, questa viene elaborata attraverso l'URI includendo un verbo, delle intestazioni e, talvolta, un corpo della richiesta.

[9] `https://httpwg.org/specs/`

[10] `https://datatracker.ietf.org/doc/html/rfc3986/`

- Dopo aver ricevuto una richiesta valida, l'API chiama il programma esterno o server web e invia la richiesta.

- Il server remoto elabora la richiesta e invia una risposta all'applicazione richiedente con le informazioni richieste. Un tipico formato di risposta potrebbe essere JSON[11] (*JavaScript Object Notation*), che utilizza coppie chiave-valore e può rappresentare oggetti complessi, array e altri tipi di dati, facilitando l'integrazione e l'interpretazione.

Le figure 2.2 e 2.3 illustrano quanto appena descritto, mostrando un esempio di chiamata API a Clockify per ottenere la lista completa dei clienti presenti nel sistema. La prima figura mostra lo schema del dialogo tra client e server remoto attraverso l'API, mentre la seconda offre un esempio pratico della chiamata e della relativa risposta, visualizzati tramite il *software* Postman[12].

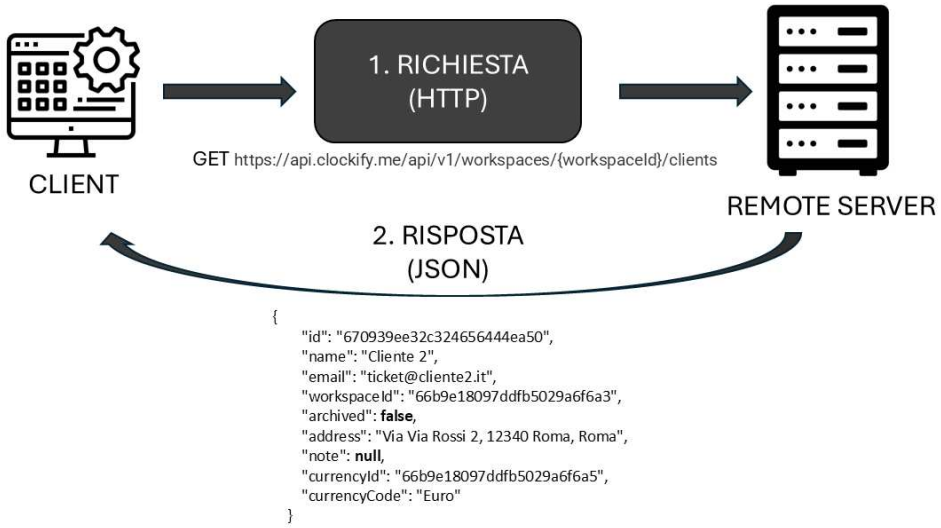


Figura 2.2: Schema di funzionamento di una API REST.

[11] https://datatracker.ietf.org/doc/html/rfc8259/

[12] https://www.postman.com/

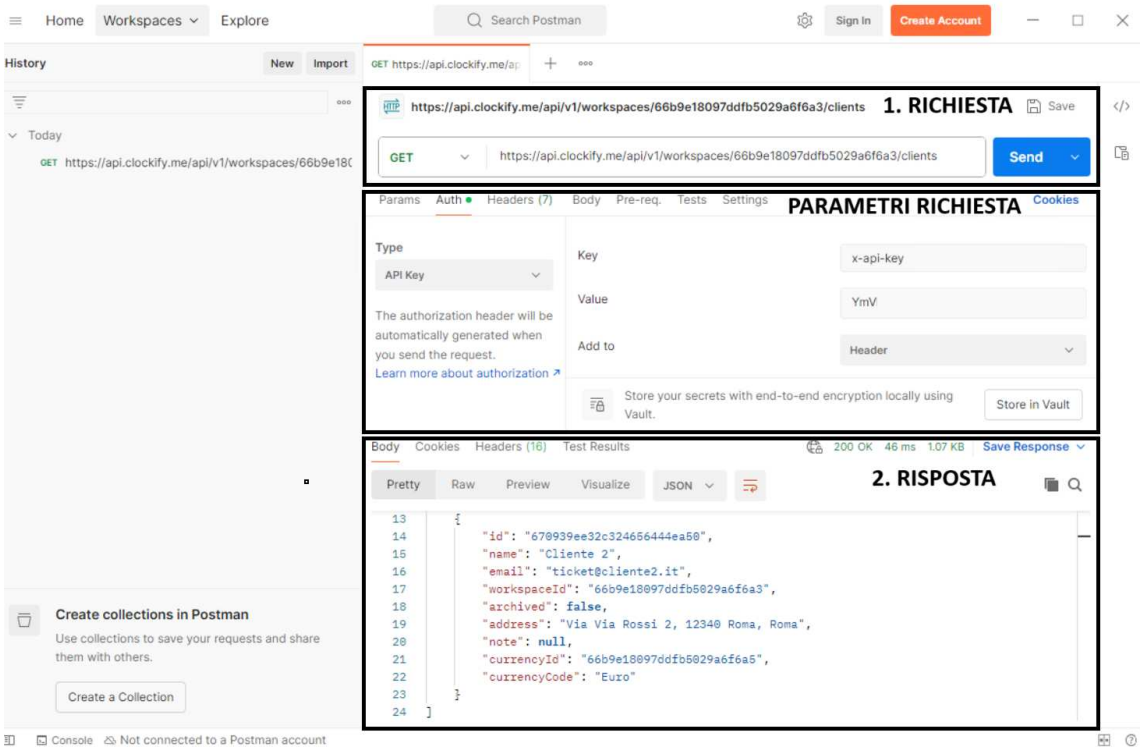


Figura 2.3: Dialogo REST nell'applicazione Postman.

2.3 MySql

MySql è un sistema *open source* per la gestione di *database* relazionali che utilizza il linguaggio SQL (*Structured Query Language*) per gestire ed interrogare i dati. Nato nel 1995 e attualmente mantenuto da Oracle Corporation, MySql è ampiamente utilizzato da sviluppatori e aziende di ogni dimensione[13].

È stato deciso di utilizzare MySql come RDBMS (*Relational Database Management System*) [2] in quanto noto per la sua alta velocità, affidabilità e facilità di utilizzo. Inoltre, da uno studio in cui sono state confrontate le prestazioni in un framework Laravel dei RDBMS SQL Server, MySql e PostgreSQL [7], MySql ha dimostrato buone prestazioni quando utilizzato con un numero ridotto di record e con hardware bassa/media qualità, confermando la sua capacità di essere adatto nelle piccole applicazioni.

[13] `https://www.mysql.com/why-mysql/`

2.4 Clockify

Clockify[14] è un *software* di monitoraggio del tempo progettato per tracciare le ore di lavoro su progetti e attività. Il programma permette agli utilizzatori di poter inserire i propri tempi di lavoro dividendoli per ogni cliente e associandoli a diversi progetti - nel presente elaborato equiparati ai *ticket*.

Nella figura 2.4 è riportata una schermata di Clockify con la calendarizzazione dei tempi di lavoro per una settimana lavorativa e il menu principale sul lato sinistro, dal quale è possibile accedere a tutte le sezioni dell'applicazione.

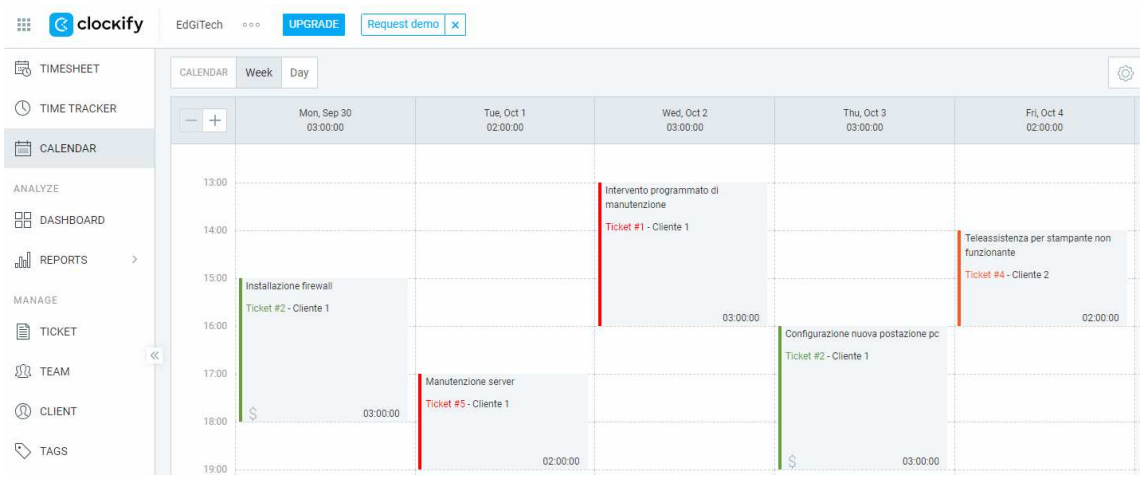


Figura 2.4: Funzione calendario di Clockify per la gestione dei tempi di lavoro.

Il piano gratuito di Clockify, al momento della redazione di questo elaborato, è particolarmente interessante, in quanto offre funzionalità essenziali senza limiti di utenti o progetti. Un altro vantaggio fondamentale del piano gratuito è l'accesso alle API[15], che permettono di automatizzare flussi di lavoro e integrare Clockify con altri sistemi, rendendolo utile per team di sviluppo e aziende che necessitano di personalizzazioni tecniche. Questa applicazione è stata selezionata in quanto rispecchia la struttura *ticket*-attività che i committenti richiedono, permettendo di

[14] `https://clockify.me/`
[15] `https://docs.clockify.me/`

poter sfruttare quindi un software già collaudato e di ottimizzare la base di dati interna e la quantità di codice. Inoltre, gli utilizzatori del gestionale potranno inserire i propri tempi di lavoro anche dall'applicazione *mobile* "Clockify Time Tracker" e visualizzare i dati in tempo reale su EdGiGest.

2.5 Implementazione del progetto

In questo paragrafo vengono indicate le tecnologie utilizzate e le scelte adottate per rendere l'ambiente di sviluppo del gestionale efficiente e sicuro.

Il progetto del gestionale EdGiGest è stato creato e sviluppato su macchina virtuale con sistema operativo Windows Server 2019. Questa macchina virtuale è stata configurata su un *server* utilizzando VMware ESXi[16], un *software* di virtualizzazione che ha permesso di creare un ambiente protetto e isolato per l'*hosting* del progetto.

Per lo sviluppo del gestionale è stato utilizzato l'IDE Microsoft Visual Studio Code[17], che offre un set di strumenti completo e potente per lo sviluppo in PHP. Il vantaggio di lavorare in Visual Studio è stato dato dalla possibilità di utilizzare tutte le funzionalità avanzate come il *debug* e l'auto-completamento, rendendo lo sviluppo più fluido e produttivo.

Il progetto è stato caricato sin dall'inizio su GitHub[18], permettendo una gestione efficace del codice attraverso il controllo di versione. Grazie all'integrazione con Visual Studio, ogni modifica al codice sorgente del gestionale è stata sincronizzata con la *repository* su GitHub, mantenendo un registro chiaro delle versioni e degli aggiornamenti.

In figura 2.5 è possibile vedere la schermata del progetto del gestionale EdGiGest caricata sulla piattaforma di sviluppo collaborativo GitHub.

[16] `https://www.vmware.com/products/cloud-infrastructure/esxi-and-esx/`
[17] `https://code.visualstudio.com/`
[18] `https://github.com/`

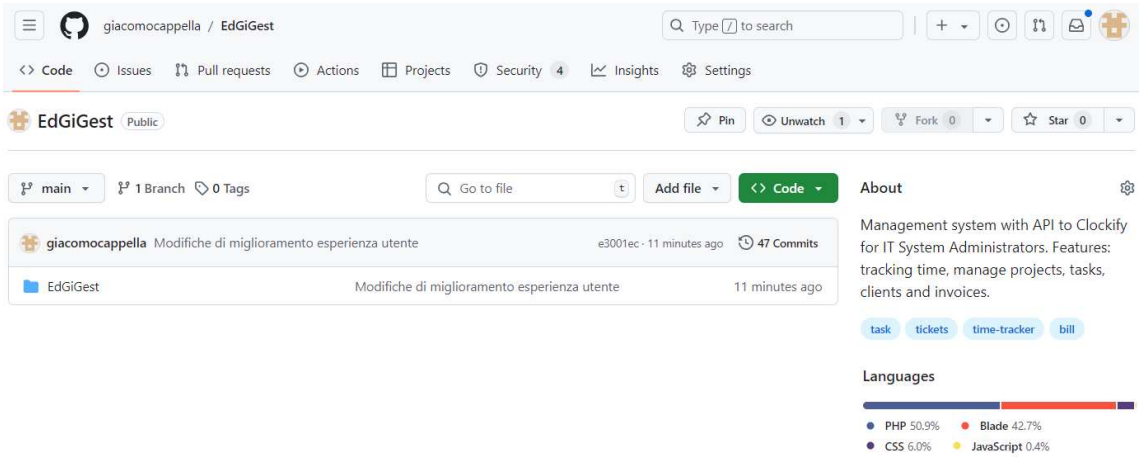


Figura 2.5: Progetto EdGiGest su GitHub.

Attualmente, il gestionale è accessibile mediante una VPN (*Virtual Private Network*), che consente ai soli utenti autorizzati di accedervi in modo sicuro, anche da postazioni remote. Questo garantisce un livello di sicurezza elevato per la protezione dei dati in esso contenuti e delle comunicazioni, poiché solo chi ha accesso alla VPN può interagire con il gestionale.

Capitolo 3

Progettazione e implementazione della base di dati

Nel mondo della programmazione *software*, la base di dati rappresenta il cuore dell'infrastruttura informativa, consentendo la memorizzazione e l'elaborazione delle informazioni necessarie al funzionamento del programma. Nel contesto dello sviluppo del gestionale EdGiGest, la progettazione accurata del *database* è fondamentale per garantire la consistenza, l'affidabilità e la facilità di accesso ai dati.

Questo capitolo si concentra sulla progettazione e implementazione della base di dati necessaria per il corretto funzionamento del *software*, a partire dalla modellazione concettuale fino ad arrivare alla rappresentazione logica. L'obiettivo è quello di fornire una struttura in grado di rispondere ai requisiti funzionali emersi durante l'analisi dei requisiti descritta nel Capitolo 1. In particolare, viene utilizzato il modello Entity-Relationship (ER)[2] per rappresentare la struttura concettuale del *database* e successivamente si procede con la trasformazione di tale modello in schema relazionale.

Nella parte finale del capitolo invece vengono descritti i passaggi seguiti per la creazione del *database* e delle tabelle attraverso il *framework* Laravel.

3.1 Diagramma ER

Il diagramma Entity-Relationship (ER) è uno strumento molto spesso utilizzato nella fase di progettazione concettuale delle basi di dati relazionali e consente di descrivere in modo grafico la realtà di interesse per fare in modo di verificare se sono stati considerati tutti gli elementi presenti nell'analisi dei requisiti.

Tale schema è formato da tre costrutti base:

- entità: rappresentano elementi concreti o astratti oggetto nella realtà di interesse sui quali il sistema deve raccogliere informazioni;
- relazioni: rappresentano un legame logico tra le entità, ovvero descrive la logica di collegamento tra i dati;
- attributi: rappresentano proprietà comuni di entità. Anche le relazioni possono possedere degli attributi.

Di seguito vengono descritte le entità presenti nel diagramma ER che è stato utilizzato per la progettazione del *database* del gestionale, il quale viene presentato in figura 3.1.

- Cliente: entità che comprende i clienti che hanno avuto o hanno in corso un rapporto lavorativo con i sistemisti di EdGiTech.
- Utenza cliente: entità che include le credenziali di accesso al sistema riservate ai clienti, i quali possono visualizzare i *tickets* a loro associati.
- Sistemista: entità che comprende i lavoratori sistemisti di EdGiTech che effettuano le lavorazioni e creano le ricevute di prestazione occasionale.
- Ricevuta: entità che include le ricevute create dai sistemisti ai fini del pagamento delle prestazioni svolte.

- *Ticket*: entità che contiene i *tickets* di lavorazione associati ad ogni cliente a cui lavorano i sistemisti.
- Attività: entità che include le attività legate ai *tickets* di lavorazione svolti per ogni cliente a cui lavorano i sistemisti.

È importante sottolineare che la parte dello schema relativa a *ticket*, attività e alle loro relazioni con le altre entità appena descritte non sono stati implementati nella base di dati del gestionale in quanto interamente gestita da Clockify attraverso il collegamento tramite API REST. Si è deciso comunque di mostrare nel diagramma ER le entità e le relative relazioni che Clockify gestisce con gli attributi ritenuti significativi per dare un'idea generale della gestione e memorizzazione delle informazioni.

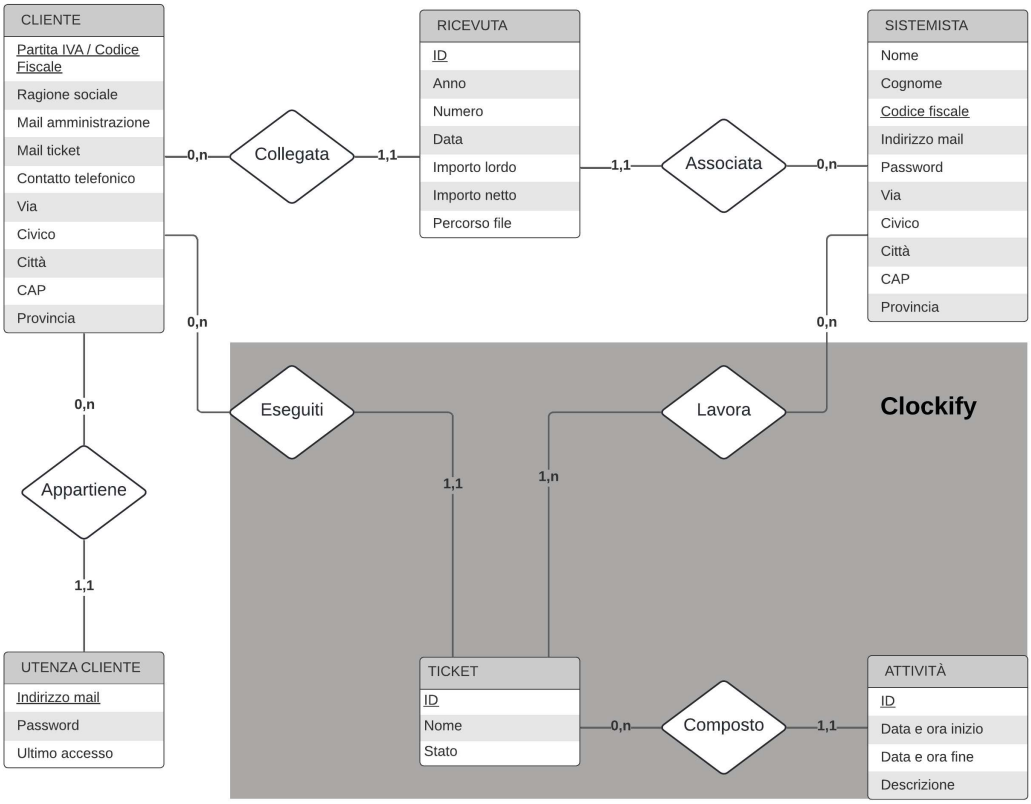


Figura 3.1: Diagramma ER del gestionale EdGiGest.

3.2 Schema relazionale

Il modello logico è un modello di dati che utilizza le strutture specifiche dell'implementazione di un *database*, mantenendosi però ad un livello più astratto e indipendente dai dettagli concreti dei dati.

Il modello logico relazionale, che si basa appunto sul concetto di relazione [2], definisce come i dati sono organizzati, le relazioni tra di essi e le regole di integrità che sono presenti tra queste.

Uno schema relazionale è composto da tabelle, ognuna delle quali rappresenta un'entità all'interno del *database*. Le tabelle sono formate da righe e colonne, dove ogni riga corrisponde a una singola istanza dell'entità ed ogni colonna rappresenta un attributo di quella entità. Inoltre, per ogni tabella, è presente la chiave primaria che identifica univocamente ogni record e l'eventuale chiave esterna che stabilisce la relazione con le altre tabelle del *database*.

Nella figura 3.2 è rappresentato lo schema relazionale del gestionale di cui successivamente vengono illustrati tutti i dettagli.

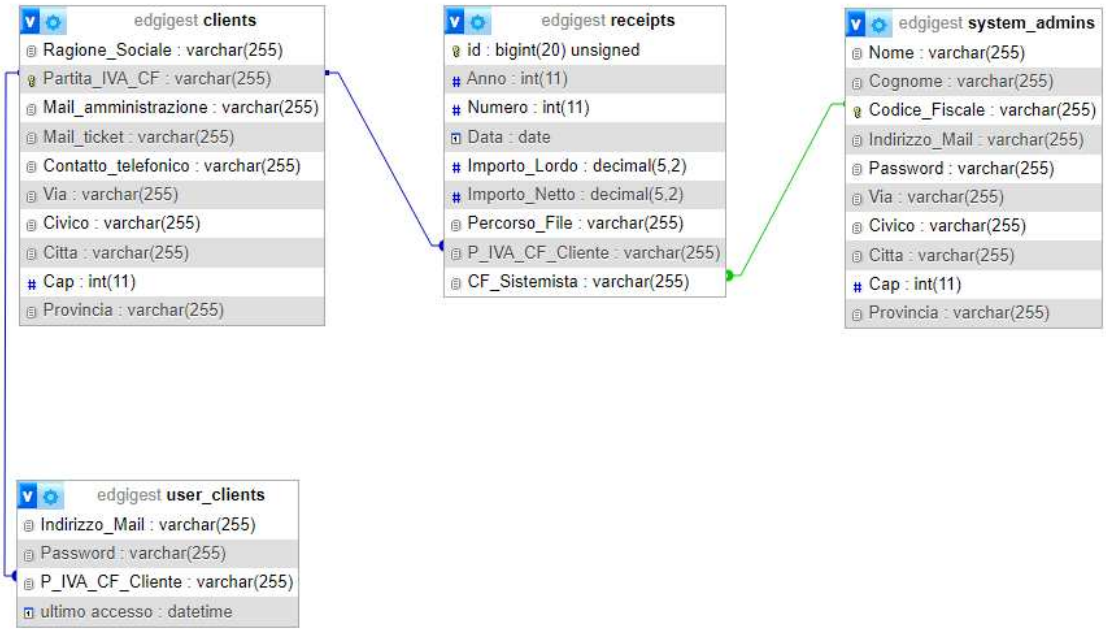


Figura 3.2: Schema logico del gestionale EdGiGest.

Come già anticipato, e come mostrato nella figura 3.2, la parte del *database* relativa ai *tickets* e alle attività non è stata implementata, poiché attualmente è completamente gestita dal *software* Clockify. Tuttavia, l'attuale struttura della base di dati è stata progettata per poter includere in futuro anche questi elementi, qualora il *software* di monitoraggio del tempo di lavoro non fosse più disponibile, oltre ad eventuali nuovi moduli del gestionale che potrebbero essere sviluppati.

Vengono ora descritte le tabelle presenti nello schema logico assieme alle relazioni che legano le stesse.

- `clients`: tabella delle anagrafiche clienti, contenente la ragione sociale, la partita IVA o il codice fiscale (chiave primaria), un indirizzo email per le comunicazioni con l'amministrazione, un indirizzo email per l'invio dei rapportini di lavorazione e una serie di campi che rappresentano l'indirizzo della sede legale, necessari per la creazione della ricevuta della prestazione.

- `user_clients`: tabella che contiene lo *username* (chiave primaria) e la *password* dei clienti per permettere l'accesso all'area riservata del gestionale contenente la visualizzazione dei *tickets* di lavorazione a loro assegnati. Inoltre è presente la chiave esterna che identifica il cliente a cui sono associate le credenziali e un campo che memorizza l'ultimo accesso al gestionale. Un cliente può possedere più credenziali di accesso al gestionale. Tutte le password memorizzate in questa tabella sono criptate.

- `system_admins`: tabella che contiene le anagrafiche dei sistemisti che lavorano per EdGiTech, necessarie per la creazione delle ricevute. In questo caso la chiave primaria è il codice fiscale del lavoratore. In questa tabella sono inoltre inseriti l'indirizzo email e la *password* necessarie per l'accesso al gestionale, criptata anche in questo caso.

- `receipts`: tabella che si riferisce alle ricevute della prestazione lavorativa, contenente un codice identificativo (chiave primaria), l'anno

di riferimento, il numero della ricevuta e la data di emissione della stessa, l'importo lordo calcolato in base alle ore di prestazione svolta e il relativo importo netto. È anche presente un campo che permette il salvataggio del percorso in cui verrà archiviata la ricevuta all'interno del gestionale. Infine sono presenti due chiavi esterne, rispettivamente la partita IVA o il codice fiscale del cliente e il codice fiscale del sistemista. Per ogni cliente e per ogni sistemista possono relativamente essere associate più ricevute di prestazione occasionale.

3.3 Implementazione della base di dati

Successivamente alla fase di progettazione si è passati all'implementazione della base di dati per il gestionale EdGiGest.

Per la creazione del *database*, è stato scelto phpMyAdmin[19], *software* gratuito che MySql mette a disposizione per la gestione delle basi di dati. In figura 3.3 viene presentato phpMyAdmin nella fase di creazione del *database* denominato "edgigest".

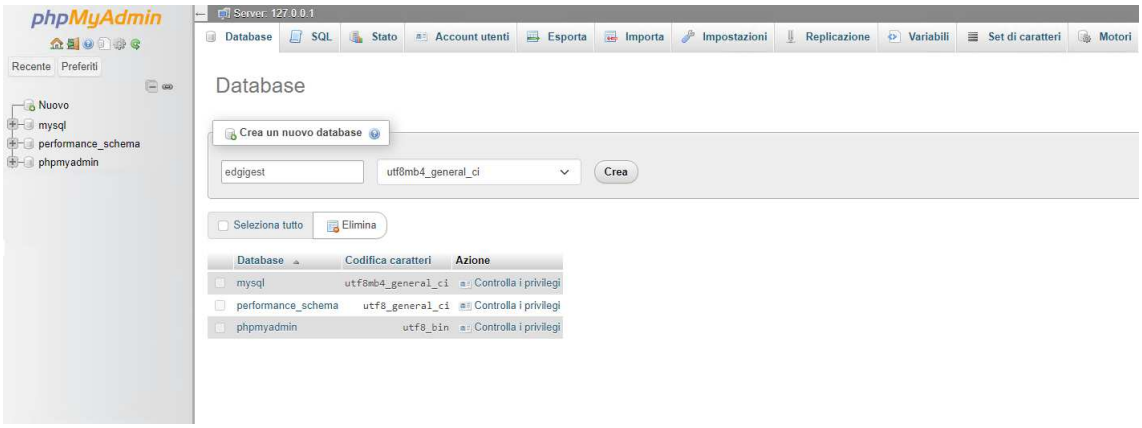


Figura 3.3: Creazione del *database* del gestionale EdGiGest.

Dopo la creazione della base di dati, è stato necessario informare l'ambiente Laravel indicando il tipo di motore SQL, il nome del *database* e le credenziali da utilizzare per il collegamento ai dati. Questo viene

[19] `https://phpmyadmin.net/`

fatto nel file `.env`, impiegato per gestire le variabili d'ambiente che contengono le informazioni di configurazione di Laravel, la cui sezione è riportata di seguito.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=edgigest
DB_USERNAME=root
DB_PASSWORD=****
```

Quando il collegamento con il *database* è attivo, è possibile procedere con la creazione delle tabelle e delle loro relazioni. In Laravel, le modalità di creazione sono molteplici, di seguito sono indicate le più rilevanti:

- Migrazioni di Laravel: consentono di creare, modificare e cancellare tabelle tramite comandi PHP.
- *Query* SQL da Laravel: si possono eseguire *query* SQL all'interno di un *controller* nel codice Laravel.
- Strumento phpMyAdmin: è possibile aggiungere tabelle direttamente dall'applicazione *web-based* di gestione *database*.
- *Command Line Interface* (CLI) del *database*: si può utilizzare l'interfaccia a riga di comando del *database* per aggiungere manualmente una tabella utilizzando codice SQL.

Si è deciso di procedere con la creazione delle tabelle servendosi delle migrazioni di Laravel, in quanto si tratta di strumento integrato che permette di utilizzare un codice facilitato.

A titolo esemplificativo, vengono ora riportati i passaggi salienti per la creazione della tabella `receipts` attraverso le migrazioni di Laravel.

Innanzitutto, è stato necessario creare un file `.php` per la migrazione. Questo può essere fatto tramite lo strumento Artisan[20], l'interfaccia a

[20] `https://laravel.com/docs/11.x/artisan/`

riga di comando integrata in Laravel che automatizza la creazione dei file, includendo le intestazioni e posizionandoli automaticamente nel percorso corretto. Alternativamente, il file può essere creato manualmente e posizionato nel percorso `app/database/migrations` del progetto.

Il comando per la creazione della migrazione dalla linea di comando di Laravel tramite Artisan è il seguente:

```
php artisan make:migration create_table_receipts
```

Il comando produce il risultato riportato in figura 3.4, che conferma l'avvenuta creazione del file di migrazione all'interno della rispettiva cartella.

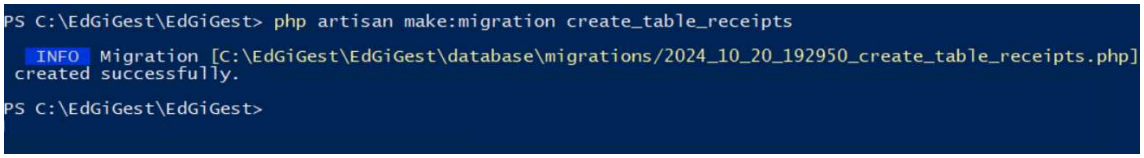


Figura 3.4: Creazione di una migrazione con Artisan.

Il file di migrazione appena creato ha l'aspetto illustrato in figura 3.5. La struttura del file generata da Artisan contiene due funzioni principali:

- `up()`: definisce le operazioni che devono essere eseguite quando la migrazione viene applicata. Di norma, include istruzioni per creare tabelle, aggiungere colonne, effettuare delle aggiunte o modifiche ai campi delle tabelle già in essere.

- `down()`: rappresenta la funzione opposta a `up()` e viene eseguita quando si vuole effettuare il *rollback* della migrazione. La funzione `down()` tipicamente contiene il codice per eliminare le tabelle o le modifiche introdotte nella funzione `up()`.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('receipts', function (Blueprint $table) {
            $table->id();
            $table->timestamps();

        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('receipts');
    }
};
```

Figura 3.5: Struttura del file di migrazione.

A seguire, è stato necessario modificare la funzione `up()` immettendo i campi da inserire all'interno della tabella. Nel caso della tabella `receipts`, la funzione contiene il seguente codice.

```
Schema::create('receipts', function (Blueprint $table) {
        $table->id();
        $table->integer('Anno');
        $table->integer('Numero');
        $table->date('Data');
        $table->decimal('Importo_Lordo',5,2);
        $table->decimal('Importo_Netto',5,2);
        $table->string('Percorso_File')->nullable();
        $table->string('P_IVA_CF_Cliente');
        $table->string('CF_Sistemista');

        $table->foreign('P_IVA_CF_Cliente')
              ->references('Partita_IVA_CF')
              ->on('clients')
              ->onDelete('cascade')
              ->onUpdate('cascade');

        $table->foreign('CF_Sistemista')
```

```
            ->references('Codice_Fiscale')
            ->on('system_admins')
            ->onDelete('cascade')
            ->onUpdate('cascade');
});
```

Attraverso il codice sopra riportato, viene impostata la creazione dei campi descritti nello schema relazionale esposto nel precedente paragrafo. Inoltre, vengono definiti i vincoli d'integrità referenziale per il collegamento con le tabelle `clients` e `system_admins`, includendo le operazioni di cancellazione e aggiornamento in cascata.

La funzione `down()`, invece, contiene solamente la sola istruzione che cancella la tabella `receipts`, nel caso di annullamento della migrazione.

Per rendere la migrazione effettiva, e quindi creare la tabella `receipts` sul *database*, è necessario eseguire il seguente comando Artisan sulla CLI di Laravel:

```
php artisan migrate
```

Una volta inviato il comando, si riceve la risposta positiva di creazione della tabella nella base di dati. In figura 3.6 viene mostrata la tabella `receipts` creata, visibile su phpMyAdmin.

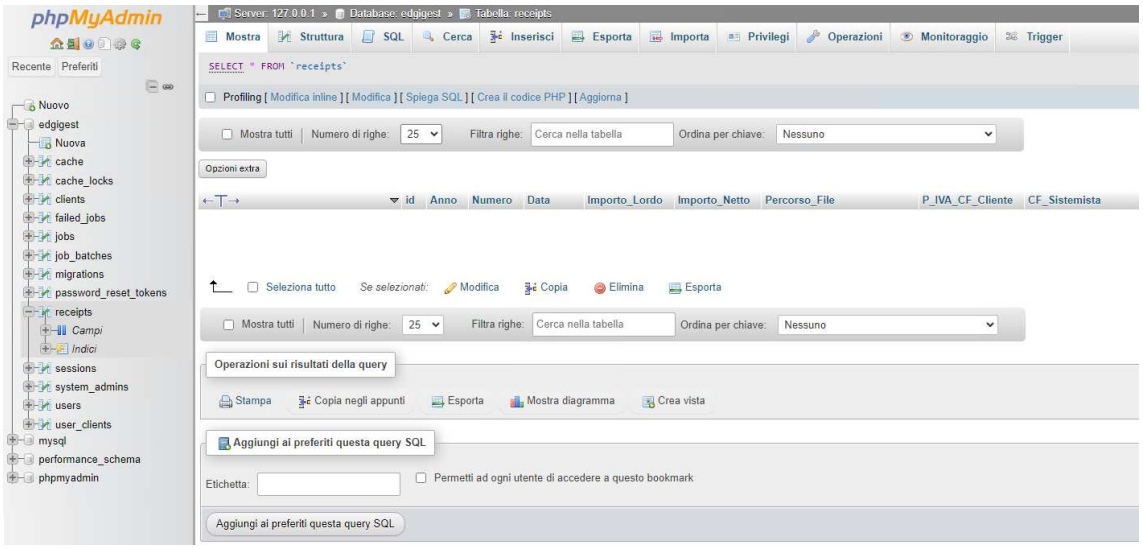


Figura 3.6: Tabella receipts su phpMyAdmin

Per eseguire le operazioni CRUD sul *database* con Laravel, è necessario un passaggio aggiuntivo: la creazione di un *model* associato alla

tabella. Questo rappresenta una classe che mappa i campi della tabella e gestisce le operazioni sul *database*, consentendo la creazione di istanze per l'inserimento, la modifica e la gestione dei *record*.

La creazione del *model* per la tabella `receipts` avviene anch'essa tramite un comando Artisan:

```
php artisan make:model Receipt
```

Questo comando crea il file `Receipt.php` all'interno della cartella `app/Models`. In questo caso il nome del *model* è impostato al singolare, in quanto rappresenta un'istanza dell'entità `receipts`, ovvero una singola ricevuta.

Di seguito, in figura 3.7, viene riportato il file `Receipt.php`.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Receipt extends Model
{
    use HasFactory;

    protected $fillable = [
        'Anno',
        'Numero',
        'Data',
        'Importo_Lordo',
        'Importo_Netto',
        'Percorso_File',
        'P_IVA_CF_Cliente',
        'CF_Sistemista',
    ];

    public function client()
    {
        return $this->belongsTo(Client::class);
    }

    public function system_admin()
    {
        return $this->belongsTo(System_admin::class);
    }
}
```

Figura 3.7: *Model* Receipt.php

È possibile notare che nel *model* in questione viene dichiarata la variabile `$fillable`, la quale definisce quali campi della tabella possono essere riempiti. Questo avviene in quanto Laravel consente di proteggere i *model* dall'assegnazione non desiderata di campi, quindi è necessario specificare esplicitamente i campi che possono essere modificati in modo sicuro.

Infine, sono presenti due funzioni che definiscono le relazioni della tabella `receipts` con `clients` e `system_admins`. La funzione `client()` definisce una relazione uno-a-molti, il che significa che una ricevuta può essere associata ad un solo cliente, ma che per un cliente possono esistere più ricevute. Analogamente, anche la funzione `system_admin()` definisce che per ogni ricevuta è associato un solo sistemista, e per ogni sistemista possono essere emesse più ricevute di prestazione occasionale.

Capitolo 4

Presentazione del gestionale

In questo capitolo vengono illustrate le funzionalità del gestionale che sono state implementate. Attraverso una serie di immagini e descrizioni vengono presentate le varie sezioni dell'applicazione, spiegando il funzionamento delle pagine e delle operazioni in esse contenute. In particolare, verrà descritta più nel dettaglio una sezione rappresentativa di EdGiGest che racchiude in sé tutte le tecniche implementative utilizzate, fungendo da modello per le altre pagine dell'applicativo. Questa sezione fornisce una visione completa della maggior parte delle operazioni che vengono eseguite all'interno del gestionale per comprenderne il funzionamento complessivo.

4.1 Layout

La scelta del tema grafico e del *layout* del gestionale è stata dettata dalla semplicità. Pertanto è stato utilizzato un *layout* che garantisca un'interfaccia pulita e intuitiva, in cui tutte le informazioni sono facilmente accessibili e a portata di mano, permettendo così al sistemista che lo utilizza di concentrarsi esclusivamente sulle sue attività.

Lo stile dell'intera interfaccia è strettamente collegato alla *dashboard* adottata. Si è pensato di utilizzare come pagina iniziale una *dashboard*

fornita dal *template* amministrativo Hope UI[21], una libreria di componenti per interfacce grafiche *open source* basata su Bootstrap[22]. La *dashboard* scelta, mostrata nel successivo paragrafo, è stata ampiamente personalizzata e semplificata per adattarsi alle esigenze aziendali.

4.2 Panoramica del gestionale

4.2.1 Dashboard

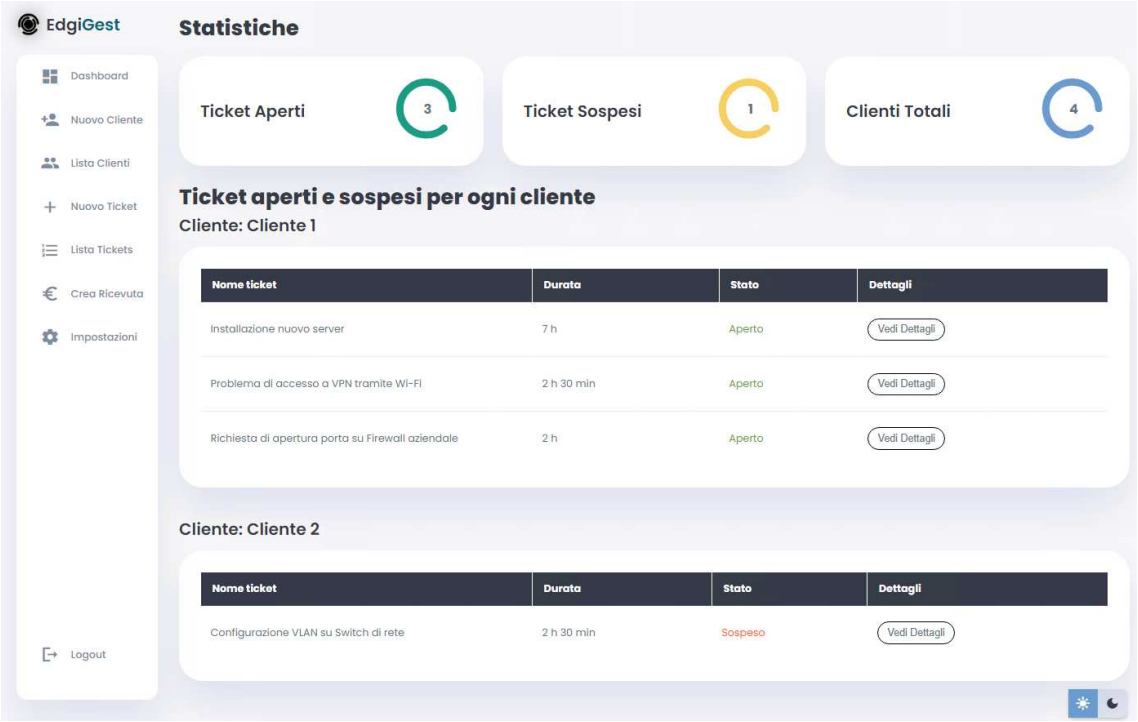


Figura 4.1: *Dashboard* del gestionale EdGiGest.

La *dashboard*, rappresentata nella figura 4.1, è la schermata principale del gestionale, che deve dare al sistemista che lo utilizza tutte le informazioni primarie ben organizzate. Innanzitutto, nella parte sinistra della schermata, è presente il menu principale che dà accesso a tutte le funzionalità implementate:

- Nuovo Cliente: funzione dedicata alla creazione dei clienti.

[21] `https://hopeui.iqonic.design/`
[22] `https://getbootstrap.com/`

- Lista Clienti: funzione che permette la visualizzazione di tutti i clienti con la possibilità di selezionarne uno e di poterne visualizzare o modificare i dettagli inseriti in precedenza.

- Nuovo Ticket: funzione per la creazione dei *tickets* di lavorazione e delle relative attività collegate.

- Lista Tickets: funzione dedicata alla visualizzazione di tutti i *tickets* presenti nel gestionale, indipendentemente dal loro stato. Vi è la possibilità di selezionare un *ticket* e vederne i dettagli e le attività collegate.

- Crea Ricevuta: sezione del gestionale che permette la creazione delle ricevute di prestazione occasionale e l'invio delle stesse al cliente.

- Impostazioni: funzione dedicata alle impostazioni del gestionale, come la gestione degli utenti e dei parametri per l'invio automatico delle email.

Nella parte alta della *dashboard* sono stati inseriti dei contatori che riportano, nell'ordine, il numero di *s* aperti, sospesi e il numero totale di clienti inseriti nel gestionale.

Infine, nella parte centrale della pagina è presente un elenco di *tickets* aperti e sospesi organizzati per ogni cliente. Questa parte è di fondamentale importanza per fare in modo che il sistemista, una volta effettuato il login al gestionale, abbia un accesso diretto a tutti i *tickets* per poterne gestire velocemente le attività collegate. Per ogni riga è presente il pulsante "Vedi dettagli", il quale rimanda alla funzione di gestione della singola lavorazione, che verrà descritta nel paragrafo 4.2.7.

4.2.2 Funzione Nuovo Cliente

Attraverso questa funzionalità è possibile inserire una nuova anagrafica cliente. Come si può notare dalla figura 4.2, viene richiesto al sistemista di inserire i campi indicati nell'analisi dei requisiti.

Una volta confermato l'inserimento, il sistema verifica la presenza e la correttezza dei dati e procede con l'inserimento degli stessi

- sul database locale, creando un nuovo record sulla tabella `clients`;
- su Clockify, in cui verranno sincronizzati tramite API POST la ragione sociale e l'indirizzo completo.

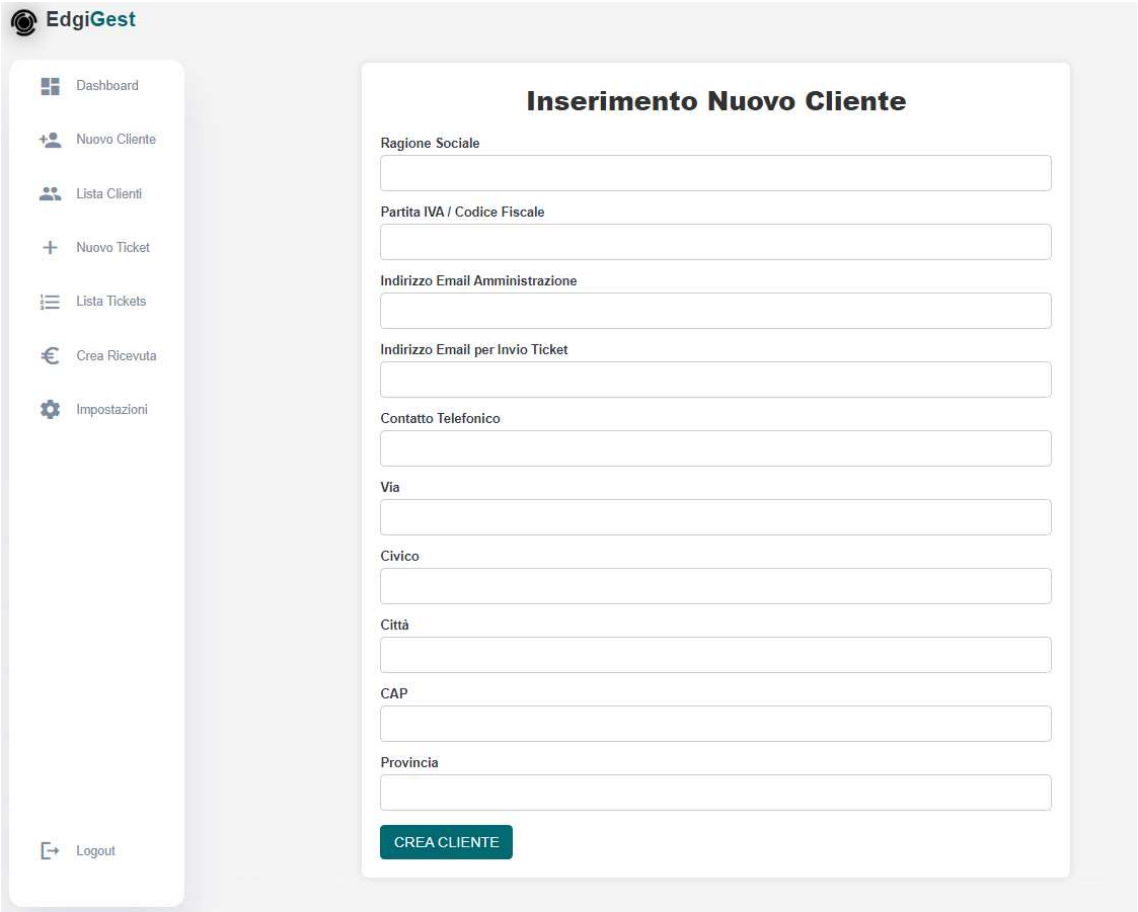


Figura 4.2: Funzionalità per l'inserimento di nuovo cliente.

4.2.3 Funzione Lista Clienti

La seguente funzionalità è stata progettata e implementata per poter visualizzare e gestire le anagrafiche cliente già presenti a sistema.

Dalla *dashboard*, cliccando sul pulsante "Lista Clienti" si verrà indirizzati verso la funzionalità visibile in figura 4.3, in cui è presente un elenco di tutti i clienti inseriti nel gestionale. In questa è possibile selezionare un cliente per poterne visualizzare i dettagli cliccando sul pulsante "Modifica" ed eventualmente effettuare delle rettifiche. I campi aggiornati, analogamente alla creazione del cliente, verranno aggiornati sul database locale e su Clockify.

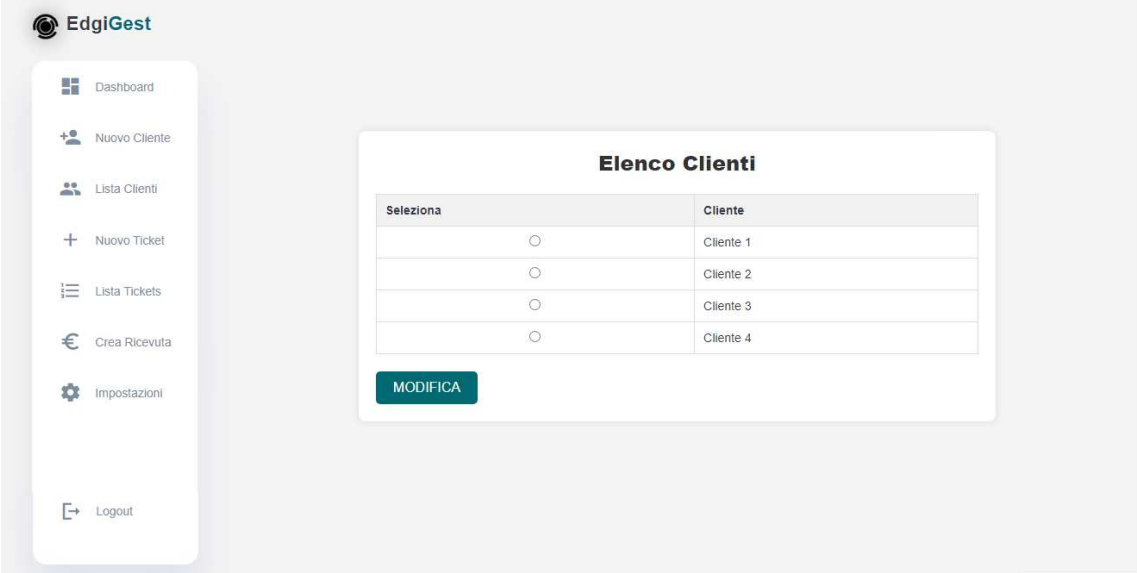


Figura 4.3: Funzionalità Elenco Clienti.

4.2.4 Funzione Nuovo Ticket

Dal menu principale presente nella *dashboard* è possibile accedere alla funzionalità di creazione di un nuovo *ticket* di lavorazione, mostrata nella figura 4.4.

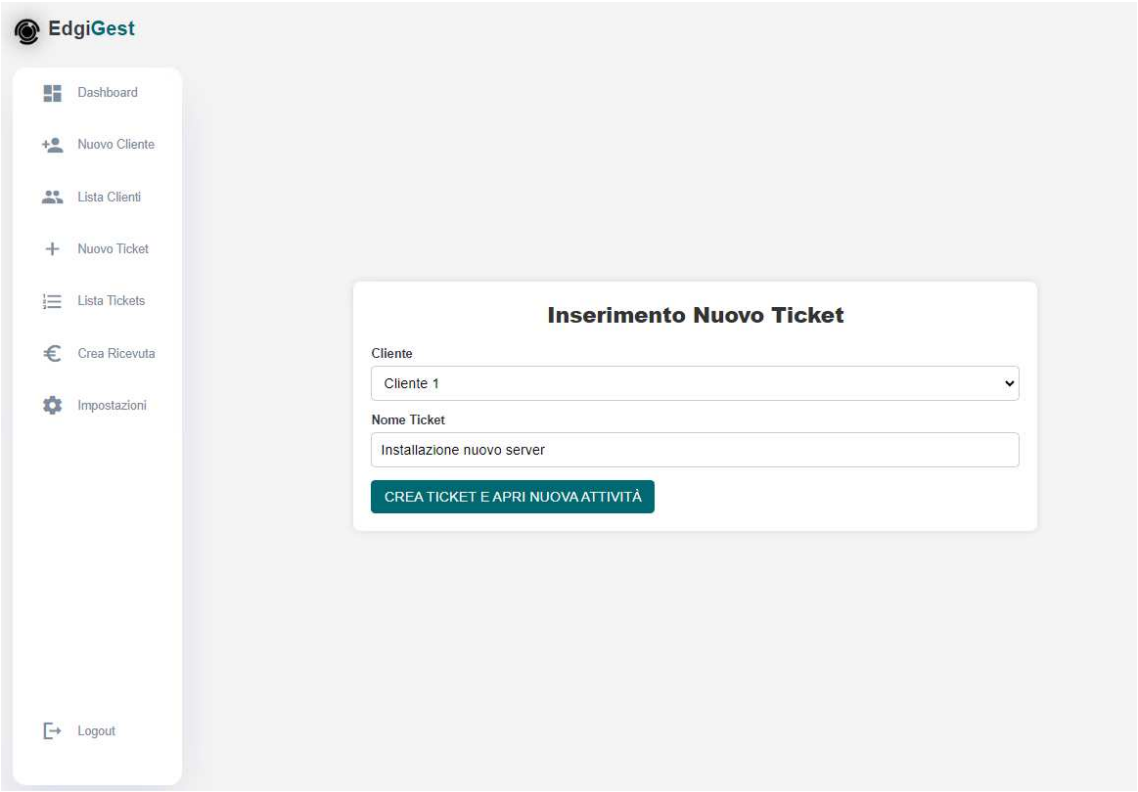


Figura 4.4: Funzionalità per la creazione di un nuovo *ticket*.

In prima istanza, per poter creare un *ticket* è necessaria la scelta del cliente a cui associare tale lavorazione. Successivamente, è obbligatoria l'indicazione di un nome che identifichi in generale il lavoro da svolgere.

Una volta confermata la creazione, il sistema crea il *ticket* su Clockify tramite API POST e rimanda l'utente alla funzionalità di creazione di una nuova attività, presentata nel successivo sottoparagrafo. Questo è stato pensato in quanto, di norma, il sistemista accede al gestionale e procede alla creazione del *ticket* di lavorazione quando si è già effettuata, o pianificata, un attività in esso contenuta.

4.2.5 Funzione Nuova Attività

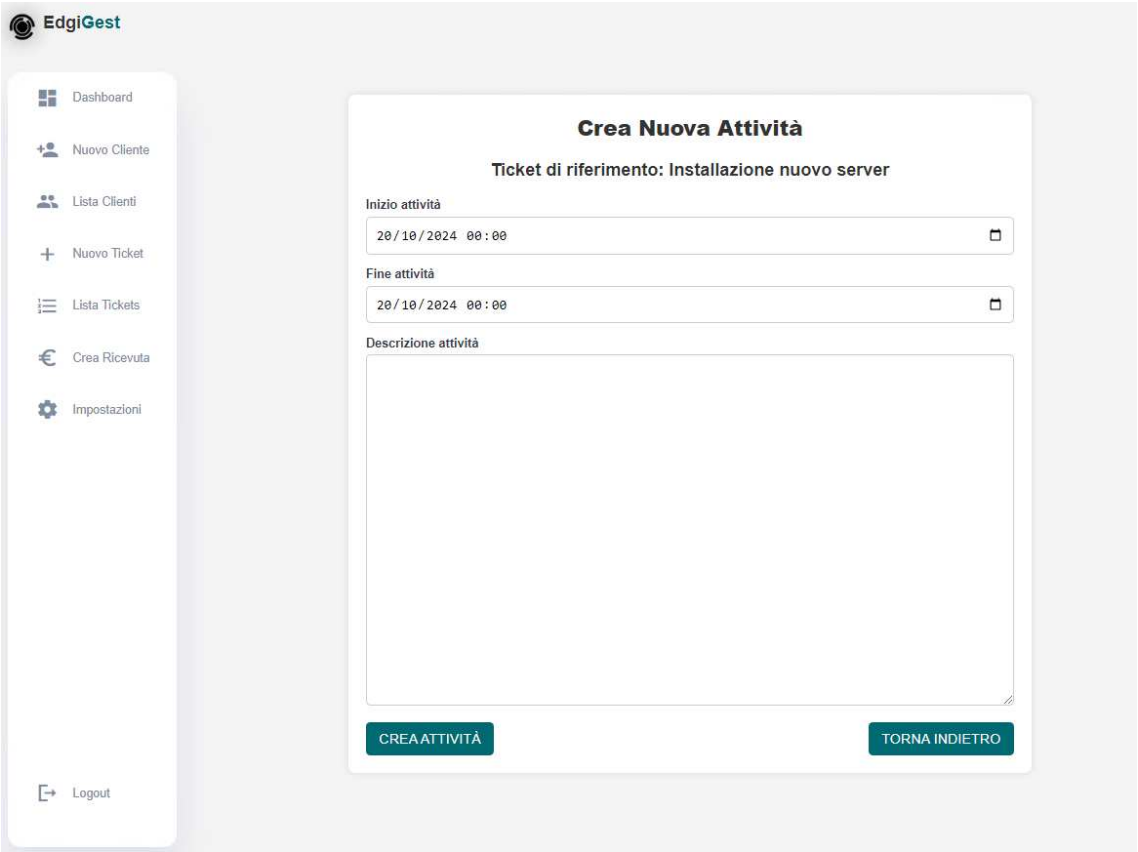


Figura 4.5: Funzionalità per la creazione di una nuova attività.

Nella figura 4.5 è possibile vedere la funzione relativa all'aggiunta di una nuova attività.

Si ricorda che questa è saldamente legata ad un *ticket*, pertanto è possibile aggiungerne una successivamente alla creazione di un *ticket* oppure dalla pagina dei dettagli dello stesso, se già esistente.

Per ogni attività deve essere indicata data e ora di inizio e di termine, assieme ad una descrizione dettagliata della lavorazione svolta per il cliente. Per agevolare il sistemista, il gestionale imposta data iniziale e finale con la data odierna, ma rimane un campo liberamente modificabile.

Al termine dell'inserimento dati, quando l'utente conferma la creazione, il sistema controlla che le date non entrino in conflitto, che la de-

scrizione sia presente e procede con la creazione dell'attività su Clockify tramite API POST.

4.2.6 Funzione Lista Tickets

Cliccando sul pulsante del menu "Lista Tickets" si viene indirizzati verso una pagina riepilogativa, rappresentata nella figura 4.6, in cui viene mostrata una lista con tutti i *tickets*, ordinati per cliente, assieme al loro stato attuale.

Questa funzionalità è stata pensata per dare al sistemista la possibilità di vedere uno storico di tutti i *tickets*, comprensivi anche di quelli già chiusi, allo scopo di consultazione o per la ricerca di risoluzioni di problemi già affrontati in passato.

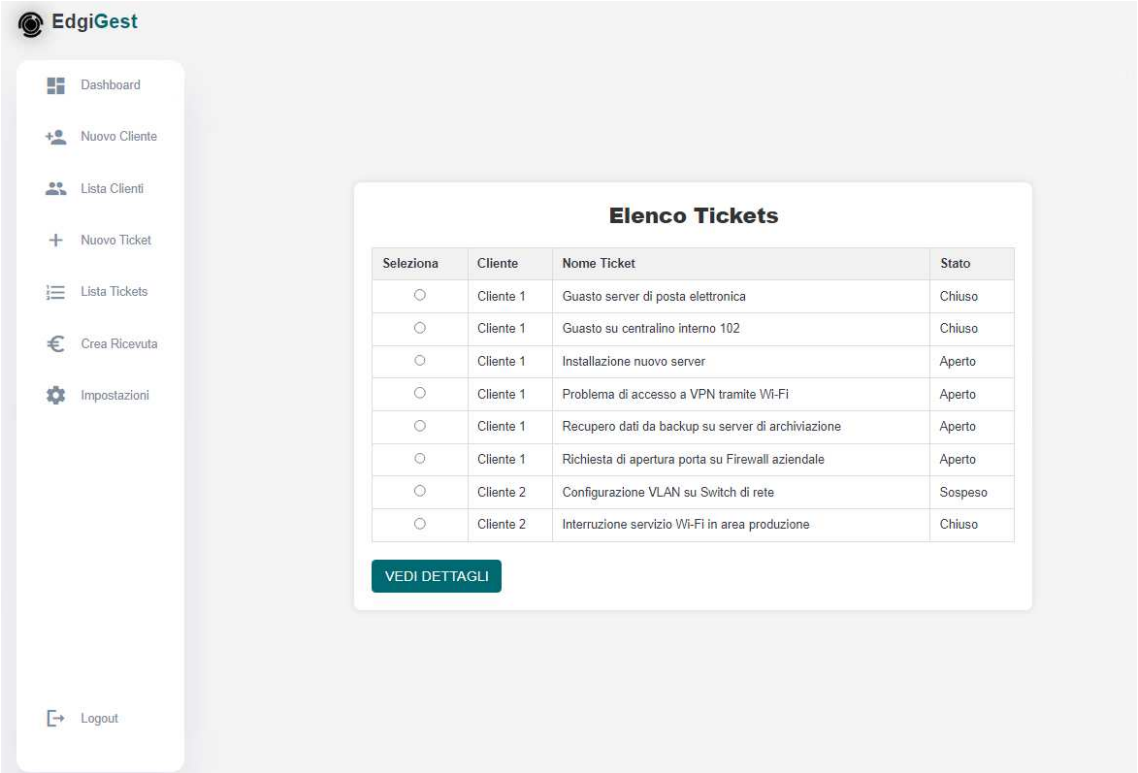


Figura 4.6: Funzionalità Lista Tickets.

Da questa schermata è possibile selezionare un *ticket* e, cliccando sul pulsante "Vedi dettagli" si verrà trasportati alla funzione dedicata, illustrata nella successivo sottoparagrafo.

4.2.7 Funzione Dettagli Ticket

La funzionalità per la gestione di un *ticket* è di rilevante importanza, in quanto si entra nel vivo della gestione delle lavorazioni. Questa è illustrata nella figura 4.7.

Figura 4.7: Funzionalità Dettagli Ticket.

In primo luogo è presente il nome del *ticket* con i dettagli più importanti, quali il nome del cliente a cui è associato, l'id su Clockify, lo stato attuale e l'ammontare orario delle lavorazioni in esso contenute. Successivamente vi è una descrizione delle componenti indicate nella figura 4.7:

1. Lista Attività: tabella in cui sono riportate, in ordine di data, le attività create e associate al *ticket* di lavorazione. Nello specifico, per ogni attività, sono presenti la data e l'ora di inizio e fine, la durata in ore e la descrizione tecnica della lavorazione svolta.

2. Pulsanti gestione attività: nella parte bassa della pagina sono presenti i pulsanti per la gestione delle attività. Attraverso il pulsante

"Aggiungi attività" sarà possibile creare una nuova attività associata al *ticket* (paragrafo 4.2.5). Il pulsante "Modifica attività" permette, dopo la selezione dell'attività da modificare, di cambiare i dettagli di questa. Analogamente, il pulsante "Elimina attività" permette la cancellazione di un'attività previa conferma del sistemista, mostrata nella figura 4.8.

Figura 4.8: Interfaccia per la conferma di eliminazione di un'attività.

3. Modifica Ticket: pulsante che permette la modifica del nome del *ticket*. Una volta confermato il nuovo nome, il sistema, attraverso una chiamata API PUT a Clockify, lo sincronizzerà con lo stesso.

4. Sospendi Ticket: pulsante che permette di cambiare lo stato del *ticket* in "Sospeso", previa conferma del sistemista.

5. Chiudi Ticket: funzione che permette la chiusura del *ticket* una volta terminate le lavorazioni. Una volta premuto questo pulsante, il sistema chiede se inviare una email al cliente con il *report* della lavorazione svolta, contenente il dettaglio di tutte le attività ad essa associate. L'interfaccia che richiede all'utente di confermare la chiusura di un *ticket* è riportata nella figura 4.9, mentre un esempio di email con il *report* di un *ticket* chiuso viene illustrato nella figura 4.10.

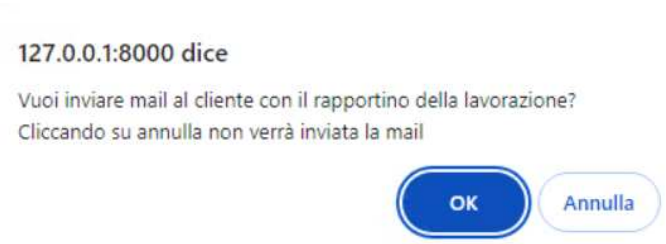


Figura 4.9: Interfaccia per la conferma di chiusura di un *ticket*.

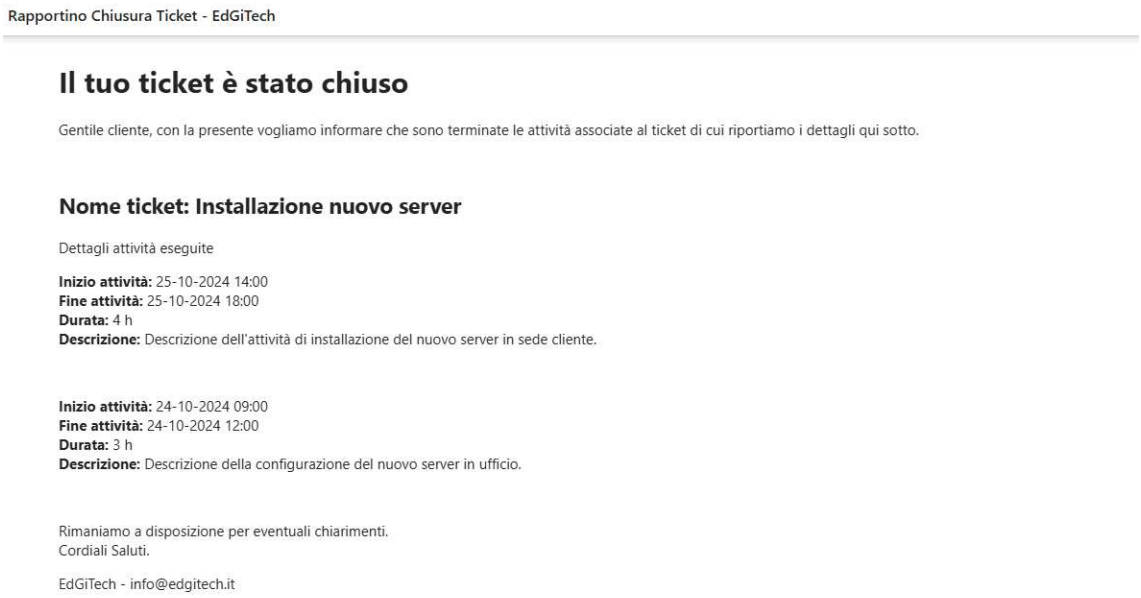


Figura 4.10: Email con il *report* di un *ticket*.

4.2.8 Funzione Crea Ricevuta

In quest'ultima sezione viene presa in rassegna la parte finale dell'iter di una o più lavorazioni effettuate per un cliente: la creazione della ricevuta di prestazione occasionale, necessaria ai fini del pagamento delle competenze del sistemista.

Partendo dalla *dashboard* e cliccando sul pulsante "Crea Ricevuta" si accede alla funzionalità dedicata.

In primo luogo, il sistemista deve poter scegliere un cliente al quale associare la ricevuta. Per implementare quanto richiesto, si è deciso di ottenere la lista dei clienti attraverso una API GET a Clockify.

La chiamata API avviene in questo modo:

1. Viene recuperata dall'ambiente di Laravel la chiave di sicurezza per poter effettuare la chiamata API;
2. Viene definito l'URI della chiamata;
3. Viene effettuata la chiamata API a Clockify con la chiave di sicurezza e l'URI appena definiti;
4. Viene verificato se la chiamata ha avuto successo, in caso affermativo viene interpretata la risposta, che contiene in questo caso la lista dei clienti, e incapsulata all'interno di una variabile;
5. Nel caso in cui la chiamata API non abbia avuto successo, il sistema restituisce un errore in cui viene indicato che non è stato possibile recuperare i dati.

I passaggi appena descritti per recuperare la lista dei clienti da Clockify tramite API vengono implementati attraverso il seguente codice all'interno del *controller* designato.

```
//1. Recupero dall'ambiente la chiave di sicurezza
$apiKey = env('API_KEY');

//2. Definisco l'URI per la chiamata API
$urlclient='https://api.clockify.me/api/v1/workspaces/
66b9e18097ddfb5029a6f6a3/clients';

//3. Chiamata API a Clockify con chiave e URI appena definiti
$responseclient = Http::withHeaders([
'x-api-key' => $apiKey,
])->withoutVerifying()->get($urlclient);


//4. Verifico se la chiamata ha avuto successo
if ($responseclient->successful()) {
    //Interpreto il risultato e lo inserisco su $dataclient
    $dataclient = $responseclient->json();

    // Passo i clienti recuperati alla view
    return view('NewReceipt', ['client'=>$dataclient]);
```

```
//5. Altrimenti gestisco l'errore
} else {
      return response()->json(['error' =>
      'Unable to fetch data'], 500);
}
```

Se la chiamata ha successo, i dati recuperati vengono inviati alla *view*, mostrata nella figura 4.11, che li posiziona in una lista ordinata per nome.

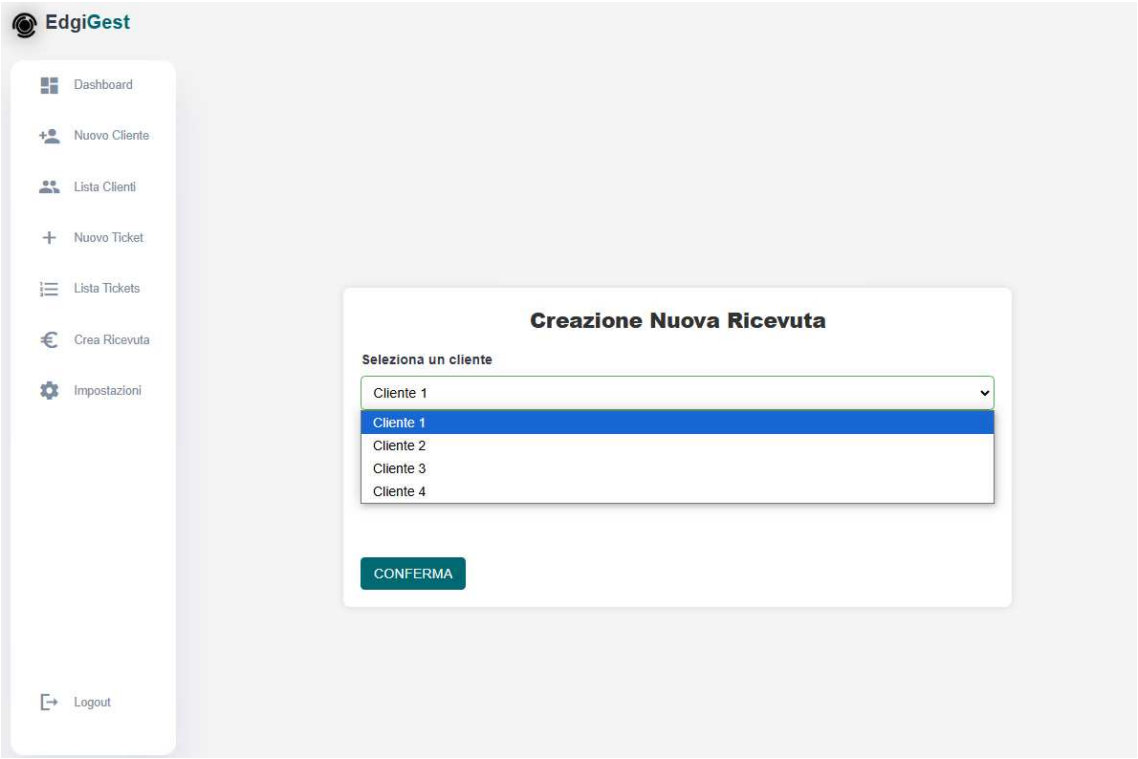


Figura 4.11: Scelta del cliente per la creazione di una ricevuta.

Una volta selezionato un cliente e cliccato il pulsante "conferma", il sistema invia la selezione effettuata al *controller*, il quale si occuperà di recuperare tutti i *tickets* chiusi del cliente in questione. Analogamente ai clienti, per il recupero dei *tickets* verrà effettuata una chiamata API GET simile a quella descritta precedentemente.

La figura 4.12 mostra in che modo vengono visualizzati i *tickets* recuperati.

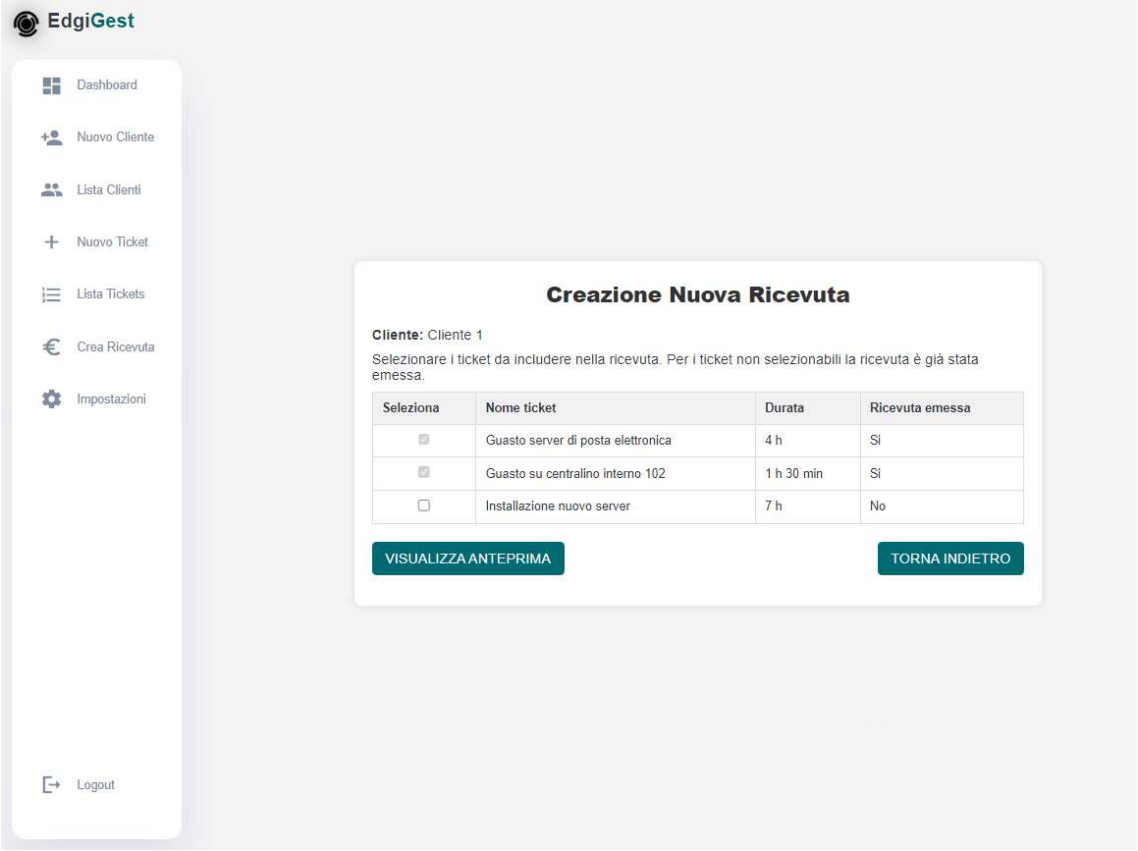


Figura 4.12: Elenco dei *tickets* da includere nella ricevuta.

Come si può notare, vengono mostrate tutte le lavorazioni concluse per il cliente e vengono resi selezionabili i soli *tickets* per i quali la ricevuta non è ancora stata emessa.

A questo punto il sistemista seleziona le lavorazioni da includere nella ricevuta e clicca su "Visualizza anteprima".

Anche in questo caso la *view* invia i *tickets* selezionati al *controller*, il quale genera un'anteprima della ricevuta.

Si è deciso di far visualizzare un'anteprima della ricevuta senza effettuare salvataggi, per dare la possibilità al sistemista di poter eseguire delle simulazioni e comunicare preventivamente al cliente l'ammontare totale, qualora fosse necessaria una sua conferma prima dell'emissione della stessa.

La schermata con l'anteprima della ricevuta è visibile nella figura 4.13.

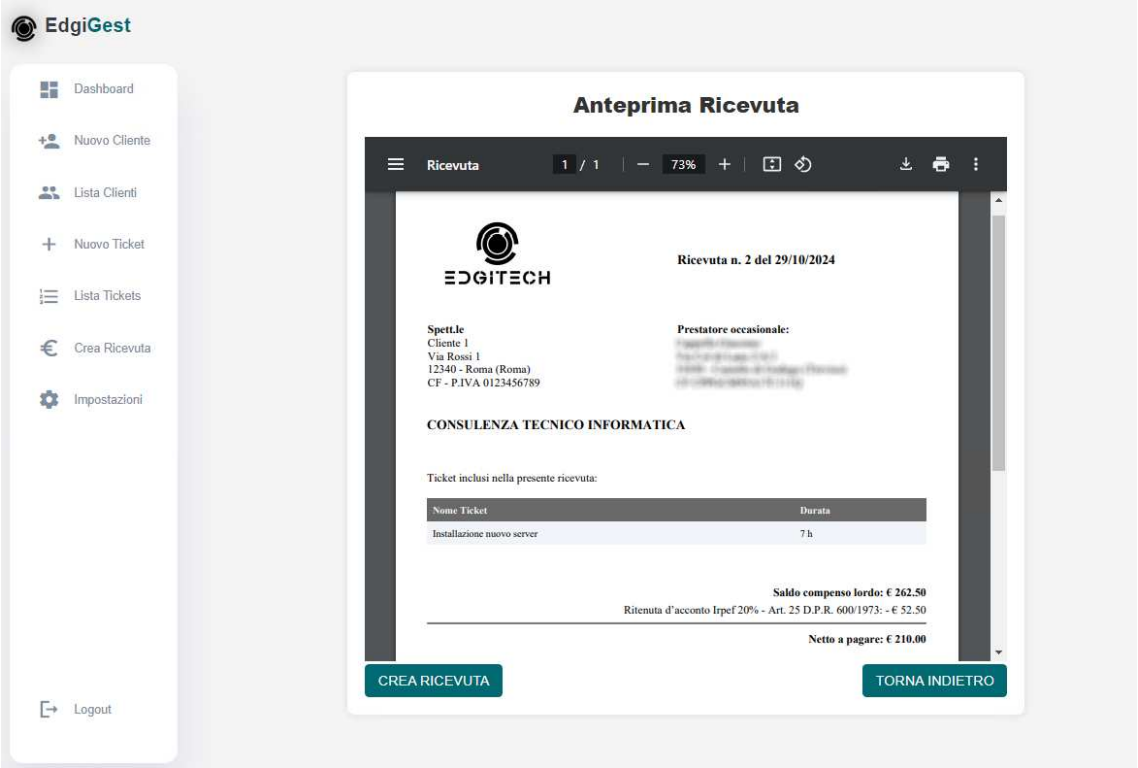


Figura 4.13: Funzionalità Anteprima Ricevuta

Da questa schermata il sistemista può visionare l'anteprima e attraverso il pulsante "Crea Ricevuta" procedere con l'emissione. Prima dell'effettiva creazione il sistema chiede all'utente se inviare copia della stessa al cliente all'indirizzo email di amministrazione. La richiesta di invio della ricevuta via email è mostrato nella figura 4.14.

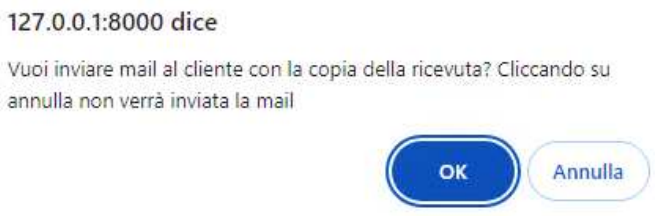


Figura 4.14: Richiesta di invio copia ricevuta emessa al cliente

Indipendentemente dalla selezione effettuata, il *controller* designato per la creazione della ricevuta effettua queste operazioni:

1. recupera i dati dei *tickets* dalla *view*, necessari per aggiornare il campo "Ricevuta emessa";

2. aggiorna, mediante una API PUT a Clockify, il campo "Ricevuta emessa" impostandola su "Si".

3. crea una nuova riga sulla tabella receipts con tutti i dettagli richiesti della ricevuta creata, come il numero, l'anno di riferimento, il cliente e il sistemista associati;

4. nel caso in cui il sistemista abbia scelto di inviare copia della ricevuta via email al cliente, invia il messaggio di posta elettronica all'indirizzo inserito in anagrafica, allegando la ricevuta appena creata;

5. rimanda l'utente alla *view* nella figura 4.15 in cui viene confermata la creazione ed, eventualmente, l'invio della ricevuta.

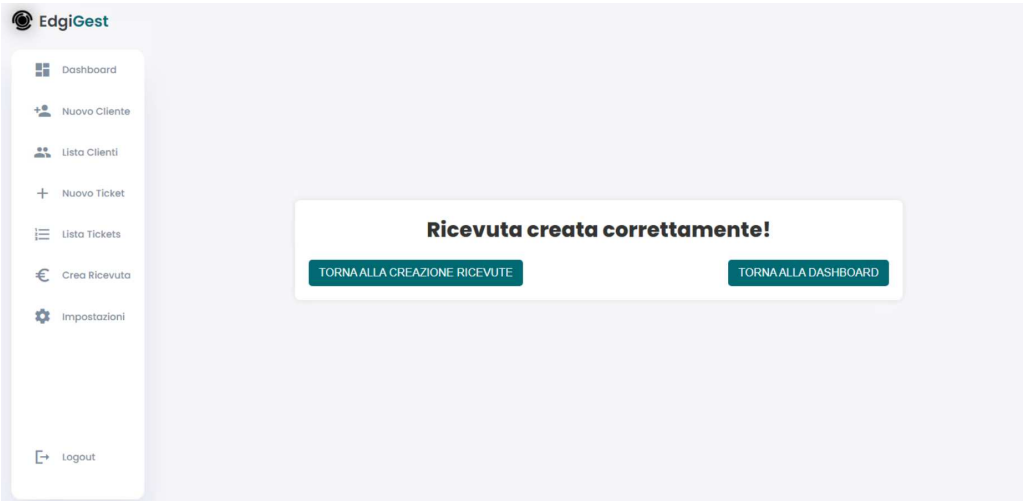
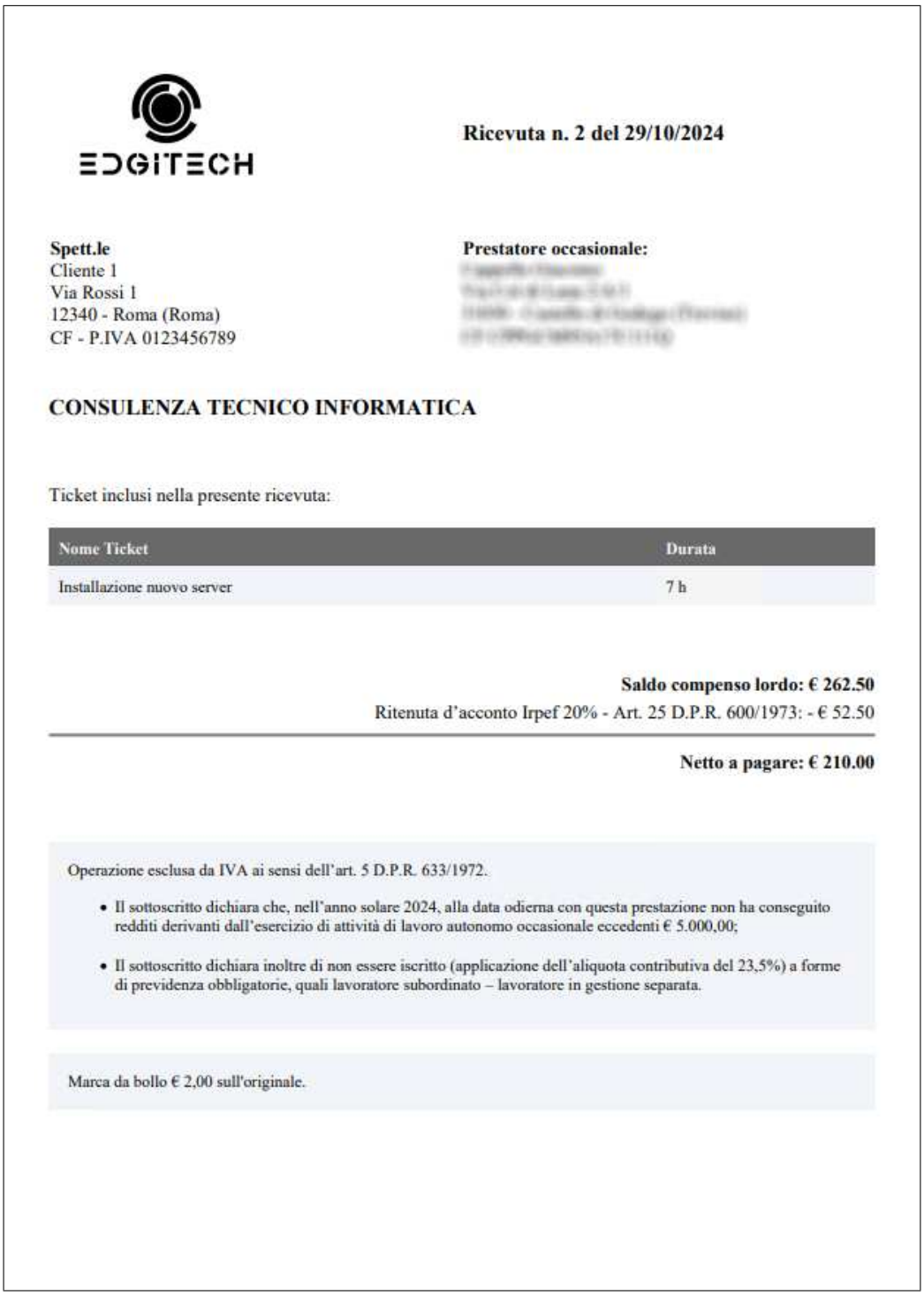


Figura 4.15: Conferma di creazione e invio ricevuta.

Di seguito, nella figura 4.16, viene riportato per intero un esempio di ricevuta di prestazione occasionale.



EDGITECH

Ricevuta n. 2 del 29/10/2024

Spett.le
Cliente 1
Via Rossi 1
12340 - Roma (Roma)
CF - P.IVA 0123456789

Prestatore occasionale:
[illegible]

CONSULENZA TECNICO INFORMATICA

Ticket inclusi nella presente ricevuta:

Nome Ticket	Durata
Installazione nuovo server	7 h

Saldo compenso lordo: € 262.50
Ritenuta d'acconto Irpef 20% - Art. 25 D.P.R. 600/1973: - € 52.50

Netto a pagare: € 210.00

Operazione esclusa da IVA ai sensi dell'art. 5 D.P.R. 633/1972.

- Il sottoscritto dichiara che, nell'anno solare 2024, alla data odierna con questa prestazione non ha conseguito redditi derivanti dall'esercizio di attività di lavoro autonomo occasionale eccedenti € 5.000,00;
- Il sottoscritto dichiara inoltre di non essere iscritto (applicazione dell'aliquota contributiva del 23,5%) a forme di previdenza obbligatorie, quali lavoratore subordinato – lavoratore in gestione separata.

Marca da bollo € 2,00 sull'originale.

Figura 4.16: Ricevuta di prestazione occasionale.

Conclusione

Il progetto, ampiamente descritto in questo elaborato, è nato con una finalità principalmente didattica, offrendo un'opportunità per consolidare ed ampliare le competenze nell'ambito della gestione di basi di dati e dello sviluppo *software*. Tuttavia, il lavoro non si esaurirà con la fine dell'elaborato: lo sviluppo del gestionale sarà portato avanti e ulteriormente implementato anche in futuro.

Sono state sviluppate tutte le funzionalità indicate come obbligatorie nell'analisi dei requisiti, tuttavia, il progetto non è ancora completo: alcune funzioni cruciali per la sicurezza, come la doppia autenticazione e l'implementazione del certificato HTTPS, sono ancora da realizzare e rappresentano degli sviluppi fondamentali per la futura messa in produzione del gestionale.

Lavorare a questo progetto ha permesso di esplorare le dinamiche del *framework* Laravel, studiandone in dettaglio la sua struttura interna e le modalità di comunicazione tra il *framework* stesso, il *database* e le API a Clockify.

Inoltre, il gestionale presenta interessanti possibilità di ampliamento, specialmente nel caso in cui EdGiTech, attualmente un'idea imprenditoriale in fase embrionale ma con solide basi di partenza, dovesse diventare una vera e propria azienda. In tale scenario, sarebbe necessario aggiornare alcune funzionalità, come la gestione delle ricevute di prestazione occasionale, trasformandola in una gestione completa delle fatture.

Nonostante i successi raggiunti, vi sono alcuni aspetti critici da considerare. L'utilizzo delle API di Clockify potrebbe, infatti, presentare dei rischi: in futuro, queste potrebbero non essere più disponibili nel

piano gratuito, subire modifiche strutturali o sperimentare malfunzionamenti. Questi fattori potrebbero rendere il gestionale non operativo o addirittura inutilizzabile, richiedendo un intervento non indifferente per renderlo indipendente da Clockify. Sarebbe, in tal caso, necessario riadattare il codice e adeguare il *database* locale, integrando le tabelle mancanti per gestire tutte le funzionalità *offline*.

In conclusione, questo progetto ha rappresentato un percorso di crescita formativa, offrendo la possibilità di acquisire competenze rilevanti e lasciando aperti ampi margini di miglioramento per il futuro, sia sul piano delle funzionalità sia in vista di possibili sviluppi aziendali.

Bibliografia

[1] Xianjun Chen, Zhoupeng Ji, Yu Fan, and Yongsong Zhan. Restful API Architecture Based on Laravel Framework. *Journal of Physics: Conference Series*, 910, 2017.

[2] Di Buccio E. Di Nunzio G. M. *Basi di dati. Progettazione concettuale, logica e SQL*, pages 1–65. Società Editrice Esculapio, 2019.

[3] Sinha Sanjib. *Beginning Laravel: Build Websites with Laravel 5.8*, pages 1–9. Apress Berkeley, CA, 2 edition, 2019.

[4] Fielding Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures - Doctoral dissertation.* PhD thesis, University of California, Irvine, 2000.

[5] Hironori Washizaki. *Guide to the Software Engineering Body of Knowledge (SWEBOK Guide)*, volume 4, pages 1.1–1.15. IEEE Computer Society, 2024.

[6] Hironori Washizaki. *Guide to the Software Engineering Body of Knowledge (SWEBOK Guide)*, volume 4, pages 7.1–7.5. IEEE Computer Society, 2024.

[7] Rafał Wodyk and Maria Skublewska-Paszkowska. Performance comparison of relational databases SQL Server, MySQL and PostgreSQL using a web application and the Laravel framework. *Journal of Computer Sciences Institute*, pages 358–364, 2020.

Sitografia

[1] Amazon Web Services, Inc. (2024) *Cos'è un framework? - Spiegazione del framework di programmazione e ingegneria - AWS*.
Disponibile a: `https://aws.amazon.com/it/what-is/framework/` (Visitato il 16.09.24).

[2] Treccani (2012) *Api - Enciclopedia*.
Disponibile a: `https://www.treccani.it/enciclopedia/api_(Lessico-del-XXI-Secolo)/` (Visionato il 16.09.24).

[3] Ministero del Lavoro e delle Politiche Sociali (2024) *Prestazioni occasionali*.
Disponibile a: `https://www.lavoro.gov.it/temi-e-priorita/rapporti-di-lavoro-e-relazioni-industriali/focus-on/disciplina-rapporto-lavoro/pagine/prestazioni-occasionali/` (Visitato il 16.10.24).

[4] DreamzTech (2023) *Why Laravel is the Most Popular PHP Framework in 2023?*.
Disponibile a: `https://blog.dreamztech.com/why-laravel-is-the-most-popular-php-framework-in-2023/` (Visionato il 30.09.24).

[5] Laravel (2024) *Laravel - The PHP Framework For Web Artisans*.
Disponibile a: `https://laravel.com/` (Visionato il 20.09.24).

[6] The PHP Group (2024) *PHP: Hypertext Preprocessor*.
Disponibile a: `https://www.php.net/` (Visionato il 16.10.24).

[7] Laravel (2024) *Eloquent*.
Disponibile a: `https://laravel.com/docs/11.x/eloquent/` (Visionato il 30.09.24).

[8] Laravel (2024) *Blade Templates*.
Disponibile a: `https://laravel.com/docs/11.x/blade/` (Visionato il 30.09.24).

[9] HTTP Working Group (2024) *HTTP Documentation*.

Disponibile a: `https://httpwg.org/specs/` (Visionato il 30.09.24).

[10] Internet Engineering Task Force (2005) *Uniform Resource Identifier (URI): Generic Syntax*.

Disponibile a: `https://datatracker.ietf.org/doc/html/rfc3986/` (Visionato il 30.09.24).

[11] Internet Engineering Task Force (2017) *The JavaScript Object Notation (JSON) Data Interchange Format* .

Disponibile a: `https://datatracker.ietf.org/doc/html/rfc8259/` (Visionato il 30.09.24).

[12] Postman (2024) *Postman: The World's Leading API Platform*.

Disponibile a: `https://www.postman.com/` (Visionato il 21.09.24).

[13] Oracle (2024) *Why MySQL?*.

Disponibile a: `https://www.mysql.com/why-mysql/` (Visionato il 21.09.24).

[14] Clockify (2024) *Clockify - Free Time Tracking Software*.

Disponibile a: `https://clockify.me/` (Visionato il 30.09.24).

[15] Clockify (2024) *Clockify API (v1)*.

Disponibile a: `https://docs.clockify.me/` (Visionato il 30.09.24).

[16] Broadcom (2024) *What is ESXI*.

Disponibile a: `https://www.vmware.com/products/cloud-infrastructure/esxi-and-esx/` (Visionato il 30.09.24).

[17] Microsoft (2024) *Visual Studio Code - Code Editing*.

Disponibile a: `https://code.visualstudio.com/` (Visionato il 30.09.24).

[18] GitHub (2024) *GitHub - Let's build from here*.

Disponibile a: `https://github.com/` (Visionato il 30.09.24).

[19] phpMyAdmin Contributors (2024) *phpMyAdmin*.

Disponibile a: `https://www.phpmyadmin.net/` (Visionato il 19.10.24).

[20] Laravel (2024) *Artisan Console*.

Disponibile a: `https://laravel.com/docs/11.x/artisan/` (Visionato il 19.10.24).

[21] Hope UI (2024) *Production-ready Open Source Bootstrap 5 Admin Dashboard.* Disponibile a: `https://hopeui.iqonic.design/` (Visionato il 10.10.24).

[22] Bootstrap (2024) *Bootstrap - The most polular HTML, CSS, and JS library in the world.* Disponibile a: `https://getbootstrap.com/` (Visionato il 10.10.24).

Elenco delle figure

Elenco delle tabelle

Ringraziamenti

Al termine di questo elaborato, desidero esprimere il mio sincero ringraziamento a tutti coloro che mi hanno supportato durante il mio percorso di studi e, in diversi modi, hanno contribuito alla realizzazione di questa tesi.

Vorrei dedicare un pensiero affettuoso alla mia fidanzata, Martina, che per me è indispensabile. Il suo amore colora tutte le mie giornate.

Ringrazio di cuore il mio migliore amico e collega Edoardo, nella speranza che questa tesi possa rappresentare un trampolino di lancio per la nostra futura attività.

Un grande ringraziamento va a mia sorella Alice, per tutto l'aiuto e la condivisione avuti in questi anni universitari vissuti assieme.

Sono molto grato a tutta la mia famiglia, che mi ha sempre spronato a perseguire i miei obiettivi, sostenendomi con costanza e fiducia. Un pensiero particolare va a mio nonno Romano, che è sempre stato per me un punto di riferimento e, da lassù, sono certo di vederlo felice per il traguardo che sto raggiungendo.

Desidero infine esprimere la mia riconoscenza alla mia seconda famiglia, quella dell'Istituto Agrario "Domenico Sartor" di Castelfranco Veneto, con la quale dal 2015 sono cresciuto sia dal punto di vista personale che professionale, per la fiducia da sempre accordatami e per il costante sostegno ricevuto.