



**Università  
degli Studi  
di Ferrara**

## **Corso di Laurea in Ingegneria Elettronica e Informatica**

Interfaccia web per la gestione delle tasse  
scolastiche in un ambiente E-learning per scuole  
superiori

Relatrice:

**Prof. Elena Bellodi**

Laureando:

**Federico Righi**

Secondo Relatore:

**Dr.Ing. Arnaud Nguembang Fadja**

Anno Accademico 2022/2023



# **Indice**

<b>Introduzione</b>	<b>5</b>
<b>1 Scopo del progetto</b>	<b>7</b>
<b>2 Tecnologie utilizzate</b>	<b>10</b>
<b>3 Realizzazione del modulo di pagamento</b>	<b>15</b>
3.1 Installazione del progetto . . . . .	<b>15</b>
3.2 Implementazione del form di creazione e aggiornamento di una rata . . . . .	<b>18</b>
3.3 Implementazione del form di pagamento di una rata speci- fica . . . . .	<b>32</b>
<b>Conclusione</b>	<b>46</b>
<b>Sitografia</b>	<b>48</b>
<b>Ringraziamenti</b>	<b>50</b>



## Introduzione

Il lavoro che verrà delineato di seguito fa riferimento a una realtà emersa recentemente nel campo dell'informatica e, più precisamente, dei sistemi web: System Afrik Information Technology (SYSAIT). L'azienda in questione è specializzata nello sviluppo di prodotti digitali mirati alla creazione di piattaforme innovative nel campo dell'intelligenza artificiale, con un costante impegno nella ricerca di soluzioni all'avanguardia.

Il tema della tesi rappresenta uno degli aspetti cruciali della piattaforma, concentrandosi sulla gestione dei pagamenti delle tasse scolastiche per gli studenti iscritti a una specifica scuola secondaria. La piattaforma in questione è denominata E-learning, un progetto intelligente e rivoluzionario che costituisce un punto di convergenza tra tecnologia e necessità primarie. In particolare, si propone di offrire un'esperienza scolastica all'avanguardia per gli studenti africani che, a causa di varie problematiche, trovano difficoltà nello sfruttare appieno le opportunità offerte.

Il lavoro svolto rappresenta uno degli elementi chiave della piattaforma, anche se non è stato il primo modulo implementato. Infatti, SYSAIT ha inizialmente sviluppato diversi moduli fondamentali prima di affrontare l'attuale attività di tesi, per cui le porzioni di programma esistenti sono state contestualizzate alle esigenze progettuali e poi concretizzate. Nello svolgimento di questo servizio sono state impiegate nozioni e competenze relative allo sviluppo di sistemi web tramite l'utilizzo di tecnologie web di base quali HTML, CSS e Javascript con il supporto di opportuni frameworks quali VueJs e Quasar.

La tesi si compone di 3 capitoli oltre la descrizione introduttiva: il primo capitolo si occupa di raccontare gli obiettivi progettuali, il secondo della spiegazione delle tecnologie utilizzate mentre il terzo dell'implementazione del modulo di pagamento spezzato in ulteriori tre sottocapitoli riguardanti circa l'installazione del progetto in locale, la progettazione dei form dedicati alla creazione e aggiornamento di una rata e del form adibito al pagamento di una rata specifica.

Grazie a questo lavoro è stato possibile evidenziare l'importanza del modulo di pagamento in tale progetto, aspetto che verrà descritto più dettagliatamente nella parte conclusiva della tesi.



# 1 Scopo del progetto

Come introdotto in precedenza, SYSAIT si occupa di fornire prodotti digitali all'avanguardia che possano rivestire un ruolo impattante sull'economia mondiale e in particolare in quella africana. Analizzando più dettagliatamente le sue origini, questa azienda è nata con lo scopo di rilanciare il successo del continente africano arginando le problematiche di cui è afflitta e dando una spinta a diversi settori quali amministrativo, economico, sociale ed educativo; per queste ragioni è nato E-learning, la piattaforma attualmente in corso di progettazione da parte di SYSAIT. Essa incarna il senso di speranza orientato alla costruzione di un futuro migliore professando l'obiettivo dello sviluppo di un'Africa digitale. Tale piattaforma è multilingua, multistruttura e multiservizio pensata per la gestione di oltre 10000 studenti e 5000 operatori, rendendola un punto di riferimento funzionale, tecnologico e qualitativo per il settore educativo; queste peculiarità hanno permesso a SYSAIT di acquisire oltre 100 clienti, anche di grandi dimensioni, con servizi differenti tra loro quali gestione completa dell'anagrafica, delle lezioni, delle iscrizioni e dei pagamenti.

Essa si compone di 3 componenti fondamentali:

- *E-learning*, dedicata alla gestione della scuola;
- *E-school*, esposizione della scuola tramite un sito web;
- *E-learning-api*, che costituisce l'API back-end dei moduli precedenti.

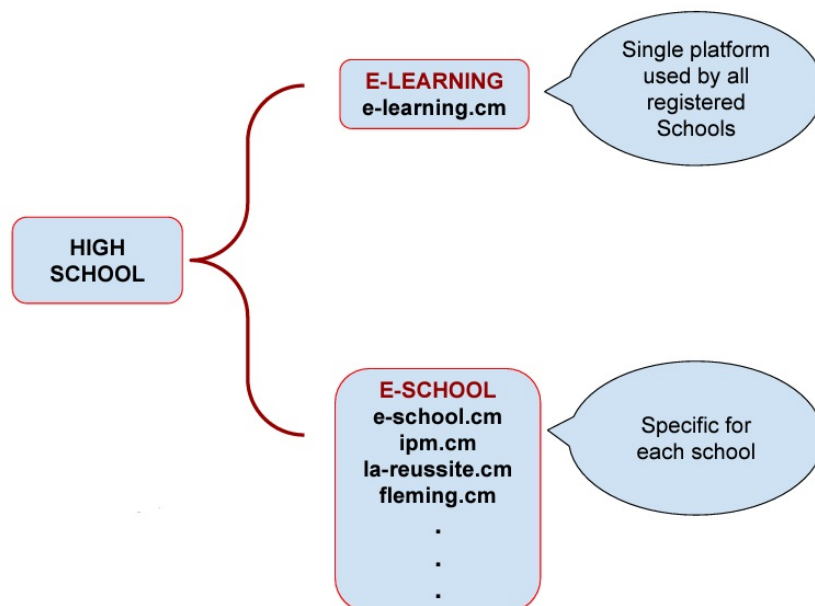


Figura 1.1: Schema SYSAIT

Il modulo E-learning si compone di diverse sezioni:

- Un'area dedicata alla gestione dell'autenticazione per diversi ruoli (amministratori, docenti, studenti, genitori, ecc.)
- Un modulo per la gestione del personale scolastico
- Un modulo per la gestione degli studenti
- Un modulo per la gestione delle classi
- Un modulo per la gestione degli esami, sia quelli sequenziali che quelli ufficiali
- Un modulo per la gestione delle pagelle e dei risultati degli studenti
- Un modulo per la gestione dei pagamenti delle tasse scolastiche
- Un modulo per la gestione del curriculum scolastico
- Un modulo per la gestione delle presenze degli studenti alle lezioni
- Un modulo per la gestione dei compiti assegnati agli studenti
- Un modulo per la gestione delle competizioni

In relazione alla sezione dei pagamenti, gli studenti sono tenuti a pagare le seguenti tipologie di tasse:

1. **Tassa di iscrizione:** richiesta al momento dell'iscrizione ai corsi di studi secondari e valida per l'intero ciclo scolastico. Lo studente può scegliere la modalità di pagamento, che può avvenire presso la segreteria della scuola, tramite borsa di studio, credito telefonico, banca designata o online. Questa tassa è unica e non può essere pagata a rate, il suo importo è stabilito dalla scuola. Nel caso di ritiro dalla scuola, lo studente deve comunque pagare l'intera tassa di iscrizione.
2. **Tassa di frequenza:** dovuta annualmente ed è suddivisibile in rate, con il pagamento della prima all'inizio dell'anno scolastico e le successive secondo quanto stabilito dalla scuola. Anche per questa tassa, l'importo è determinato dalla scuola e lo studente può scegliere la modalità di pagamento. In caso di trasferimento, lo studente dovrà pagare le rate secondo quanto stabilito dalla nuova scuola.
3. **Tassa d'esame:** riservata esclusivamente al momento della presentazione della domanda per gli esami ufficiali. Il suo importo è stabilito dallo Stato e la modalità di pagamento è a discrezione dello studente. A differenza delle altre tasse, non è suddivisibile in rate e nel caso in cui lo studente non riuscisse a sostenere l'esame per motivi imprevisti, è tenuto a pagare l'intera tassa.

Per le tasse qui sopra elencate, ogni studente potrebbe essere soggetto a un contributo aggiuntivo se non rispetta la scadenza di pagamento.



Diversamente, oltre a fungere anche come da CMS (*Content Management System*), il modulo E-school è progettato per offrire una visione completa dell'organizzazione scolastica, comprese le sue strutture, le attività extracurricolari, i club disponibili, i contatti di riferimento, la sede, le ultime novità pertinenti, la storia della scuola, gli esami ufficiali, le statistiche e altre informazioni rilevanti.

Il fine della tesi è incentrato sulla realizzazione lato front-end del modulo di pagamento delle rate tramite le tecnologie opportune e consentire agli utenti di accedere facilmente ai servizi della piattaforma; tutto questo deve produrre un design reattivo, gestire la funzionalità multilingua per una base di utenti globale, rendere visibili i dati delle fatture relativi ad ogni rata e abilitare la funzionalità di pagamento per gli utenti.

Le funzionalità specifiche da apportare al progetto lato front-end sono le seguenti:

- Nella sezione “Pagamenti” di uno studente, implementare la tabella delle tasse dello studente che comprende nelle colonne i dati principali di ogni rata (nome, categoria, data di scadenza, codice della fattura, ammontare da pagare e lo stato del pagamento);
- Opzione di modifica e cancellazione per ogni rata della tabella;
- Aggiunta di un bottone per il pagamento di una rata solo per quelle in attesa di pagamento e per quelle scadute (dato verificabile tramite lo stato del pagamento di una rata);
- Creazione di un popup lanciato alla pressione del bottone di modifica che consente di aggiornare i dati di una specifica rata;
- Creazione di un popup per aggiungere una nuova rata da pagare per lo studente;
- Creazione di un popup contenente i dati di pagamento alla pressione del pulsante di pagamento, il quale deve mostrare i dati principali della fattura, un input per allegare files salvati su drive e una select per il metodo di pagamento;
- Adattamento e aggiornamento dei file di traduzione per rendere il modulo di pagamento perfettamente comprensibile nelle diverse lingue riconosciute dall'applicativo.

## 2 Tecnologie utilizzate

Il progetto si avvale di un unico sistema di gestione di database relazionali (RDBMS), PostgreSQL<sup>1</sup>, il quale rappresenta una congrua scelta di utilizzo per la sua capacità di estensibilità e il suo essere un sistema open-source; esso possiede diverse caratteristiche chiave che gli hanno permesso di diventare uno dei database più utilizzati:

- **Affidabilità e Conformità agli Standard**

- PostgreSQL è una solida soluzione per la gestione dei dati, offrendo una semantica ACID per le transazioni che garantisce affidabilità e coerenza dei dati.
- Supporta una vasta gamma di tipi di dati, dai comuni INTEGER e VARCHAR ai TIMESTAMP e BOOLEAN, consentendo agli sviluppatori di rappresentare i dati in modo accurato e efficiente.
- Possiede la capacità di memorizzare oggetti binari di grandi dimensioni, come immagini, video o suoni, il che lo rende adatto anche per applicazioni che richiedono la gestione di dati multimediali.
- La sua affidabilità è ulteriormente rafforzata da funzionalità come il *logging write-ahead*, che fornisce una robusta registrazione delle operazioni e garantisce la tolleranza agli errori.

- **Estensioni**

- PostgreSQL offre una serie di potenti caratteristiche: il recupero *point-in-time*, il *Multi-Version Concurrency Control* (MVCC), i *tablespace*, i controlli di accesso granulari e altri ancora.
- Il controllo di concorrenza a multi-versione (MVCC) consente letture e scritture concorrenti delle tabelle, bloccando solo gli aggiornamenti concorrenti sulla stessa riga. Questo approccio riduce al minimo i conflitti e migliora le prestazioni del sistema.

- **Portabilità**

- PostgreSQL si distingue per il suo solido supporto a Unicode e a una vasta gamma di set di caratteri internazionali e codifiche di caratteri multi-byte. È in grado di gestire efficacemente l'ordinamento, la sensibilità alle maiuscole e alle minuscole, e la formattazione in diverse lingue e contesti culturali.
- E' multiplatforma, il che significa che può essere eseguito su diversi sistemi operativi, inclusi Linux, Microsoft Windows, macOS, FreeBSD e Solaris.

- **Caricamento Dinamico**

- PostgreSQL offre la possibilità di caricare dinamicamente il codice personalizzato degli utenti, consentendo loro di estendere le funzionalità del database attraverso l'aggiunta di nuove funzioni o tipi di dati. Questa caratteristica unica consente di modificare il funzionamento del database "al volo", consentendo una rapida implementazione di nuove strutture di archiviazione e applicazioni senza dover interrompere il server.

---

<sup>1</sup>PostgreSQL

Tutte queste peculiarità e altre ancora permettono di poter gestire al meglio i dati di database di grandi dimensioni e, in questo caso, i dati del progetto promosso da SYSAIT.

Per semplificare l'amministrazione dei database viene utilizzato pgAdmin 4<sup>2</sup>, un'interfaccia di amministrazione web open source progettata per gestire e interagire con database PostgreSQL; offre un'ampia gamma di funzionalità, inclusi strumenti per la creazione e la gestione di database, la scrittura di query SQL, la gestione degli utenti e la supervisione delle prestazioni. L'applicazione è indirizzata sia agli amministratori del database, sia agli utenti e gestisce i permessi prelevandoli dal database PostgreSQL. pgAdmin 4 permette di creare un database da zero, creare ed ottimizzare le tabelle, integrando un sistema di feedback sulla creazione di queste ultime per evitare errori. L'amministratore dispone di un'interfaccia grafica multi-piattaforma sviluppata in Python; questa interfaccia consente all'amministratore di eseguire diverse operazioni, tra cui l'inserimento di nuovi utenti, la modifica delle loro password e la gestione dei permessi associati agli utenti nel database utilizzando il linguaggio standard SQL. L'interfaccia è stata progettata per essere accessibile in una vasta gamma di contesti linguistici, con supporto per una dozzina di lingue diverse.

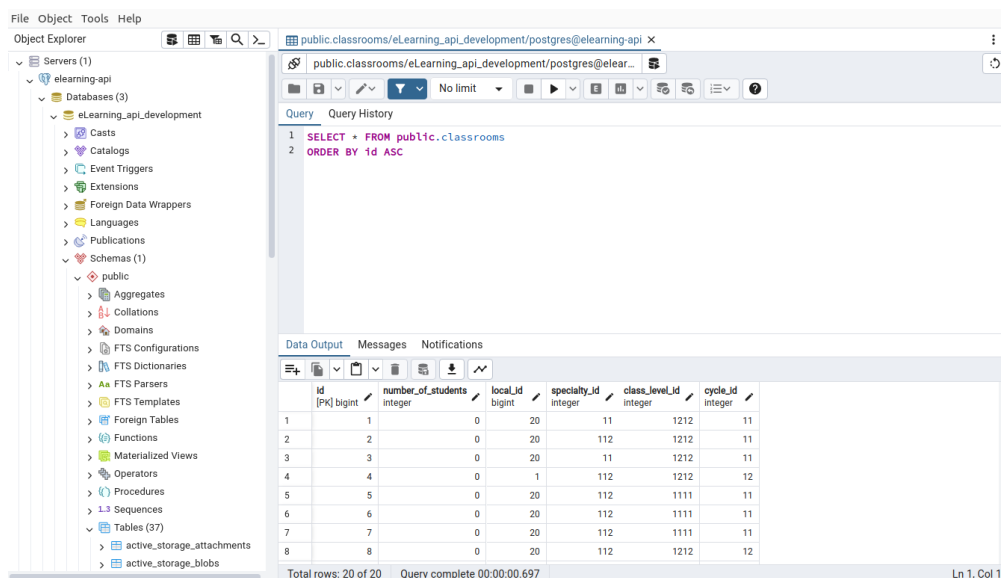


Figura 2.1: interfaccia pgAdmin 4

Generalmente il sistema è regolato da un unico server back-end, un'unica piattaforma gestionale e l'impiego di prodotti quali AWS<sup>3</sup> (*Amazon Web Services*) ed Heroku<sup>4</sup>, rispettivamente per la produzione e l'impiego di ambienti di test per la rappresentazione delle diverse scuole presenti nell'applicativo.

<sup>2</sup>pgAdmin 4

<sup>3</sup>AWS

<sup>4</sup>Heroku

Completando il discorso lato back-end, esso viene governato dal framework Rails<sup>5</sup> che sviluppa l'applicazione web tramite Ruby, un linguaggio di programmazione object-based famoso per la sua versatilità e semplicità, il quale trova impiego in una vasta gamma di applicazioni tra cui, come in questo caso, nello sviluppo di piattaforme digitali.

D'altro canto, la produzione lato front-end si arma dell'utilizzo di HTML, CSS e Javascript nonché di opportuni frameworks per la comunicazione tramite API con il back-end, ovvero VueJs<sup>6</sup> e Quasar<sup>7</sup>. VueJs è un framework Javascript progressivo e reattivo utilizzato per la creazione di interfacce utente avanzate; basato su un'architettura di componenti, permette la definizione di componenti riutilizzabili con il proprio stato e logica. La peculiarità di tale prodotto è che offre un efficace sistema di binding bidirezionale che sincronizza automaticamente il modello della vista con lo stato dell'applicazione. Quindi VueJs è dotato di due funzionalità fondamentali:

- **Rendering Dichiarativo:** Vue amplia l'HTML standard introducendo una sintassi di template che consente di definire in modo esplicito l'output HTML basato sullo stato JavaScript, semplificando la descrizione della struttura dell'interfaccia utente.
- **Reattività:** Vue monitora automaticamente le modifiche nello stato JavaScript e gestisce in modo efficiente l'aggiornamento del DOM quando si verificano cambiamenti.

Quasar è un framework *open-source VueJs-based*, ovvero basato sul framework VueJs descritto in precedenza, che permette agli sviluppatori di creare siti web responsive e, grazie alla sua CLI (*Command Line Interface*), applicazioni *cross-platform* (web, desktop, mobile e browser extension) da un'unica codebase.

Per il corretto svolgimento del lavoro, è stato necessario il supporto di una macchina virtuale (Oracle VM VirtualBox<sup>8</sup>) che permettesse di lavorare in un ambiente *Linux-based*, più nello specifico Ubuntu, opportunamente configurato e adattato alle specifiche esigenze. Successivamente si è proseguito all'installazione dell'IDE Visual Studio Code<sup>9</sup>, strumento essenziale per l'attività di tesi e su cui sono state installate specifiche estensioni. Sempre all'interno dell'ambiente Ubuntu sono stati installati in locale i progetti di front-end e back-end, in modo tale da potervi lavorare con la versione più aggiornata comprendendo tutte le porzioni di codice e files relativi ai diversi moduli e opzioni già implementate (l'installazione del progetto in locale è avvenuta tramite terminale).

I passaggi per l'apprendimento dei frameworks, la spiegazione degli obiettivi di progetto e i passi per la corretta installazione dei software e variabili d'ambiente sono presenti all'interno della piattaforma Confluence<sup>10</sup>, che mette a disposizione degli sviluppatori e del team di lavoro la suddivisione dei moduli dell'applicativo in cartelle, mostrate nella parte sinistra dell'interfaccia, le quali indirizzano ad altre cartelle contenenti elementi correlati.

A supporto dell'attività, ha funto da mezzo integrativo la creazione di diagrammi tramite draw.io<sup>11</sup>.

---

<sup>5</sup>Rails

<sup>6</sup>VueJs

<sup>7</sup>Quasar

<sup>8</sup>VirtualBox

<sup>9</sup>Visual Studio Code

<sup>10</sup>Confluence

<sup>11</sup>draw.io

draw.io è un software che permette di creare diagrammi direttamente dal browser web, senza ricorrere ad applicazioni. Tale servizio è multipiattaforma, *opensource* e funziona anche con strumenti quali Google Drive e DropBox, sfruttandoli come spazio su cui salvare i propri progetti. E' comunque disponibile l'app desktop per i sistemi operativi macOS, Linux e Windows. draw.io è dotato di un'interfaccia intuitiva che facilita la creazione di diagrammi attraverso un semplice sistema di trascinamento e rilascio (*drag & drop*). Gli utenti possono beneficiare di una vasta gamma di modelli già pronti da personalizzare, semplificando il processo di creazione dei diagrammi. Oltre ai diagrammi di flusso, draw.io supporta la realizzazione di organigrammi, diagrammi di rete e molte altre tipologie di diagrammi. La sua versatilità lo rende una scelta ideale per chiunque abbia bisogno di visualizzare concetti complessi in modo chiaro e intuitivo.

Infine, durante l'attività di tesi, per rendere visibile gli aggiornamenti e/o modifiche al progetto in locale front-end ai collaboratori e componenti del team si è ricorsi alla piattaforma BitBucket<sup>12</sup>, un software di gestione delle versioni (VCS – *Version Control System*) basato sul web e di proprietà di Atlassian<sup>13</sup> (proprietaria anche di Confluence); consente agli sviluppatori di creare repository sia pubblici che privati, dove possono collaborare gestendo il codice in modo condiviso. Le principali attività effettuate sul codice includono:

- *pull*: acquisire una copia del codice presente nel repository remoto e salvarla localmente sul proprio ambiente di sviluppo.
- *push*: caricare nel repository remoto le modifiche apportate al codice in locale, rendendo così disponibili le ultime modifiche agli altri membri del team.
- *merge*: combinare le modifiche effettuate da più sviluppatori in modo da integrare le diverse versioni del codice.
- *commit*: conferma delle modifiche apportate che vengono registrate nel sistema di controllo di versione.

Per sfruttare queste funzionalità, è stato necessario creare un nuovo branch (feature/Popups) su Bitbucket dedicato al progetto svolto in cui effettuare i salvataggi dei progressi del lavoro. Successivamente, da terminale sono stati digitati i seguenti comandi:

- *git fetch*: aggiornare il git locale riconoscendo i nuovi branch creati su bitbucket
- *git checkout feature/Popups*: spostamento sul branch chiamato "feature/Popups"
- *git branch*: visualizzare il branch corrente

Sono stati digitati i seguenti comandi per comunicare l'identità dell'utente che lavorerà in questo branch (email e nome di Bitbucket):

- *git config --global user.email "tu@esempio.com"*
- *git config --global user.name "Il tuo nome"*

---

<sup>12</sup>BitBucket

<sup>13</sup>Atlassian

Per fare in modo di creare un salvataggio e caricarlo su questo branch in Bitbucket, è necessario trascrivere i comandi seguenti nell'interfaccia del terminale:

1. *git add .* (aggiunta delle modifiche relative alla directory corrente e relative sotto-directory nella staging area, in cui vengono preparati i file per il commit)
2. *git commit -m "messaggio di commit"* (creare un'istanza di salvataggio in locale assegnandole un nome)
3. *git push* (caricamento del commit precedente sul branch selezionato di Bitbucket)

Ogni modifica al codice viene registrata e salvata come una versione separata, consentendo agli sviluppatori di tracciare l'evoluzione del progetto nel tempo. Questo permette di recuperare versioni precedenti del codice in caso di necessità o di risolvere eventuali problemi che possano emergere durante lo sviluppo collaborativo.

## 3 Realizzazione del modulo di pagamento

### 3.1 Installazione del progetto

Lo svolgimento del lavoro è susseguito dalla configurazione del progetto in locale sulla virtual machine Ubuntu utilizzabile tramite l'applicazione di Oracle "VirtualBox"; dopo l'installazione di quest'ultima, il processo di set-up è stato organizzato in due principali punti:

1. Installazione del progetto Back-end in locale;
2. Installazione del progetto Front-end in locale;

#### 1) Installazione progetto Back-end in locale

L'installazione del progetto Back-end è articolato nei seguenti passi:

1. Creazione account personale su Bitbucket, piattaforma che offre servizi di hosting di repository per il controllo di versione
2. Configurazione chiave SSH sull'ambiente Bitbucket, per instaurare una connessione sicura tra due macchine remote in una rete aperta non protetta
3. Installazione Ruby (versione 3.0.2)
4. Installazione Rails (versione 6.1.7.6)
5. Installazione PostgreSQL (versione 14.10)
  - Installazione pgAdmin 4 (`sudo apt install pgadmin4` da linea di comando)
6. Clonazione progetto locale sul computer
  - `git clone git@bitbucket.org:SYS_AIT/elearning-api.git`
7. Creazione, nella root directory del progetto, di un file denominato ".env", il quale va configurato tramite le opportune variabili d'ambiente
8. Installazione delle dipendenze
  - `sudo apt-get install libpq-dev`
  - `bundle install o make dependencies`
9. Creazione e inizializzazione del database locale
  - `rails db:create`
10. Completamento inizializzazione database e creazione tabelle
  - `rails db:migrate`
  - `rails db:migrate RAILS_ENV=test`
  - `make migrations`

## 11. Lancio del progetto localmente

- `rails server -p 3002` o `make dev`

Dopo aver completato i precedenti passaggi per una corretta configurazione e installazione del progetto, sono state create delle risorse aggiuntive tramite comandi da terminale; i risultati di questi ultimi sono stati opportunamente verificati in modo tale che ognuno di essi abbia comportato gli effetti voluti al database. Questa procedura si può compiere tramite la piattaforma pgAdmin 4.

- Creazione di un utente amministratore, il quale è responsabile della creazione delle risorse seguenti:
  - Aprire il file `elearning-api/lib/tasks/create_user.rake`
  - modifica il contenuto di tale file con i propri dati personali
  - esegui il comando `rake create_user`
- Creazione di un tema, quindi una combinazione di colori
  - Apri il file `elearning-api/lib/tasks/create_theme.rake` e possibilmente modifica la denominazione e i colori di tale tema
  - imposta la variabile `ADMIN_USER` (es. `ADMIN_USER = 1`) nel file `".env"` con l'identificativo dell'utente amministratore per creare un tema
  - esegui il comando `rake create_theme`
- Creazione di una scuola radice (scuola avente come numero identificativo "00000000000")
  - Imposta la variabile `ADMIN_USER` (es. `ADMIN_USER = 1`) nel file `".env"` con l'identificativo dell'utente amministratore per creare la scuola radice
  - esegui il comando `rake create_root_school`
- Creazione di una scuola
  - Imposta la variabile `ADMIN_USER` (es. `ADMIN_USER = 1`) nel file `".env"` con l'identificativo dell'utente che popolerà la scuola
  - imposta la variabile `NUMBER_OF_RECORDS` per definire il numero di scuole che vuoi creare (`NUMBER_OF_RECORDS = 1`)
  - esegui il comando `rake create_theme`
  - apri il file `elearning-api/lib/tasks/rake/populate_schools.rake` e possibilmente modificalo con le nuove informazioni della scuola
  - esegui il comando `rake populate_schools`
  - Notare che `root_id` e `parent_id` per ogni scuola creata corrisponde all'identificativo della scuola radice SYSAIT (creato precedentemente)
  - Notare inoltre che `theme_id` è l'identificativo del primo tema nel database (creato precedentemente)



- Creazione di un periodo scolastico
  - Apri il file `elearning-api/lib/tasks/rake/populate_scholastic_periods.rake` e possibilmente modificalo con le informazioni del nuovo periodo scolastico
  - imposta nel file `".env"` le variabili d'ambiente `SCHOOL_ID` (es. `SCHOOL_ID = 2`), `ADMIN_USER` (es. `ADMIN_USER = 1`) e `NUMBER_OF_RECORDS` (es. `NUMBER_OF_RECORDS = 1` per creare un periodo scolastico)
  - esegui il comando `rake populate_scholastic_periods`
- Creazione di alcuni locali per la scuola
  - imposta nel file `".env"` le variabili d'ambiente `SCHOOL_ID` (es. `SCHOOL_ID = 2`), `ADMIN_USER` (es. `ADMIN_USER = 1`) e `NUMBER_OF_RECORDS` (es. `NUMBER_OF_RECORDS = 20` per creare 20 locali)
  - esegui il comando `rake populate_locals`
- Creazione di alcuni classi per la scuola
  - Imposta nel file `".env"` le variabili d'ambiente `SCHOOL_ID` (es. `SCHOOL_ID = 2`), `ADMIN_USER` (es. `ADMIN_USER = 1`) e `NUMBER_OF_RECORDS` (es. `NUMBER_OF_RECORDS = 20` per creare 20 classi)
  - esegui il comando `rake populate_classrooms`
- Creazione di uno studente ( esso crea `user`, `attend` e `attend_scholastic_period`)
  - Apri il file `elearning-api/lib/tasks/populate_students.rake` e possibilmente modificalo con le nuove informazioni dello studente. Ricordare di compilare `sign_up_params` e `student_params` con i valori corretti
  - imposta nel file `".env"` le variabili d'ambiente `SCHOOL_ID` (es. `SCHOOL_ID = 2`), `SCHOLASTIC_PERIOD_ID` (`SCHOLASTIC_PERIOD_ID = 1`) `ADMIN_USER` (es. `ADMIN_USER = 1`), e `NUMBER_OF_RECORDS` (es. `NUMBER_OF_RECORDS = 1`), ovvero il numero di studenti da creare
  - esegui il comando `rake populate_students`

## 2) Installazione progetto Front-end in locale

Dopo aver configurato il progetto back-end, i passi da seguire per implementare il progetto lato front-end sono semplici:

1. clonare il progetto con il comando `git clone git@bitbucket.org:SYS.AIT/elearning-payment.git`
2. creare come per il back-end, il file `".env"`, copiare e incollare dentro il contenuto di `".example.env"`
3. lanciare il comando `yarn install`
4. avviare il server con il comando `quasar dev -p 8899` (se `quasar` non è riconosciuto, lanciare `yarn quasar dev -p 8899`)

Dopo aver lanciato il server, si apre automaticamente il sito web del progetto nel browser predefinito.

## 3.2 Implementazione del form di creazione e aggiornamento di una rata

Aperto per la prima volta il progetto su Visual Studio Code, esso era composto da numerose cartelle contenenti vari files precedentemente implementati che raffiguravano le sezioni del sito funzionanti, quindi per lo scopo del progetto è stato fondamentale modificare i files presenti e crearne alcuni per rendere le parti esistenti compatibili con gli aggiornamenti impiegati e anche con i nuovi files creati.

Prima di procedere con la spiegazione del lavoro, è necessario introdurre la struttura del progetto front-end e come sono suddivise le cartelle che contengono i files utili allo scopo di quest'ultimo; l'ambiente è composto per la maggior parte da documenti aventi estensione `.vue`, quindi scritti secondo il costrutto del framework Vuejs, in cui ognuno di essi viene suddiviso in tre parti:

- *template*: definisce la struttura HTML e la presentazione visuale della pagina, dove vengono definite le interfacce utente, gli elementi HTML e gli eventi associati. In sintesi, definisce cosa viene visualizzato nella pagina e come viene organizzato.
- *script*: parte della pagina che contiene la logica Javascript e gestisce il comportamento della pagina. Il codice all'interno viene eseguito dal motore Vue e interagisce con il template e gli altri dati della pagina. All'interno vi vengono definiti i dati, metodi, lifecycle hooks e ulteriori funzionalità.
- *css*: definisce lo stile visivo della pagina, compreso il layout, i colori, i font e altri aspetti dell'aspetto visivo; viene applicato direttamente agli elementi HTML nel template per definire l'aspetto finale della pagina.

Questi documenti assumono un ruolo preciso e sono suddivisi in varie directory all'interno del progetto; all'interno della cartella "components" sono presenti:

- *forms*: cartella in cui sono alloggiati files `.vue` aventi come scopo quello di rappresentare del contenuto all'interno delle finestre di popup lanciate nelle diverse pagine dell'applicativo. La denominazione di questi ultimi è contraddistinta dalla parola chiave "Form" seguita dal nome dell'argomento, una variabile di tipo stringa che ha un contenuto diverso da pagina in pagina.
- *utils*, cartella contenente:
  - *tables*: directory che contiene i files `.vue` relativi alla composizione di tabelle all'interno delle pagine del sito; questi documenti vengono importati dalle diverse pagine dell'applicativo per creare tabelle e modificarne le sottoparti di cui sono composte (es. intestazione della tabella, corpo della tabella, righe, bottoni, celle della tabella, ...)
  - Ulteriori files `.vue` che vengono importati e utilizzati dagli altri files dell'applicativo, ovvero documenti che sono stati progettati allo scopo di essere adattabili alle diverse esigenze del software (tabelle e sottoelementi, bottoni, file picker, componenti per l'input, ...)
- diverse sottodirectory non di interesse per il lavoro corrente

In seguito altre importanti directory che compongono il progetto:

- *css*: contiene file di stile utilizzati in alcune sezioni del sito;
- *i18n*: directory che ingloba tre sottocartelle (en, it, fr), le quali racchiudono i vari files javascript adibiti alla definizione delle traduzioni delle parole principali impiegate nel sito nelle tre lingue principali, ovvero inglese, italiano e francese. Per motivi di ordine, in ogni cartella di una lingua vengono creati tanti files javascript quante sono le diverse sezioni del sito web in cui impiegare determinate parole.
- *mixins*: strumenti che consentono di condividere e riutilizzare la logica comune tra più componenti Vue. In sostanza, un mixin rappresenta un set di opzioni che possono essere “mescolate” all’interno di un componente Vue in modo tale da implementare funzionalità aggiuntive (metodi, funzioni, *lifecycle hooks*,...). La loro utilità risiede nell’evitare duplicazione di codice nei diversi componenti, migliorandone chiarezza e comprensione, rendendolo inoltre meno soggetto ad errori. Nel nostro caso, la cartella dei mixins è suddivisa per moduli che rappresentano diverse aree dell’applicativo e contiene inoltre importanti documenti con estensione .js che si occupano di definire le differenti logiche che verranno meglio documentate in seguito.
- *pages*: una delle directory principali, comprendente tante cartelle quante sono le sezioni distinte dell’applicativo e che definiscono i documenti .vue relativi ad ogni pagina del sito, incorporando componenti, tabelle e files precedentemente descritti.
- *router*: modulo che definisce una serie di “rotte” all’interno dell’applicazione, dove ogni rotta corrisponde a una determinata URL e viene associata a un componente Vue specifico. Essa offre differenti funzionalità, come la possibilità di gestire rotte nidificate, passare parametri attraverso le rotte, gestire rotte protette che richiedono l’autenticazione dell’utente e altro ancora.

All'apertura del browser successiva al lancio del server di front-end ci si imbatte nella home page del portale web, tramite il quale è necessario autenticarsi effettuando il login con i dati dell'utente amministratore configurati nella fase di creazione di un utente successiva all'installazione del progetto back-end in locale. La pagina home si presenta come riportato dalla figura 3.2.1:

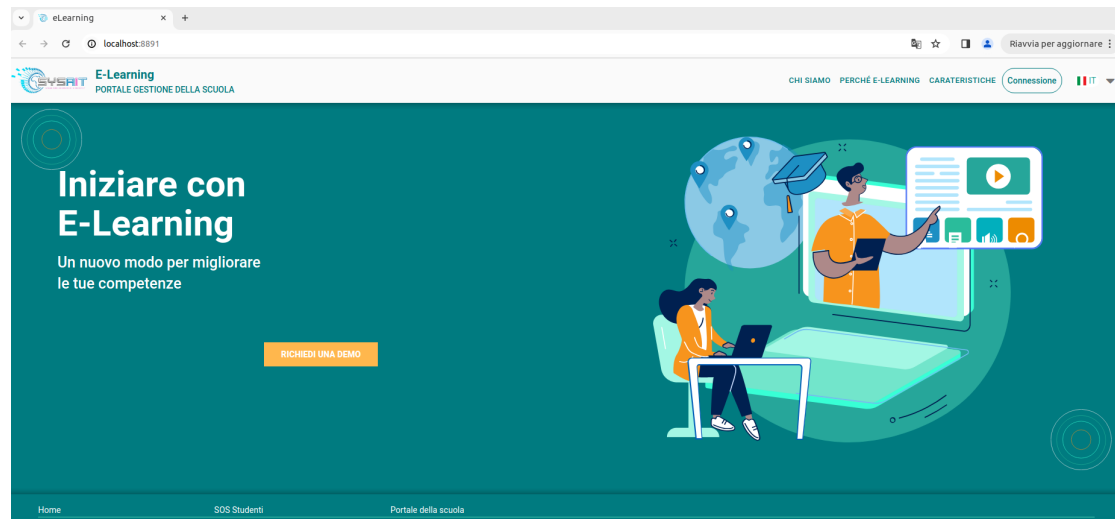


Figura 3.2.1: Home page del sito web E-Learning

Fatto questo, bisogna spostarsi nella pagina di uno studente, e per fare questo è necessario innanzitutto selezionare il periodo scolastico di interesse, cliccare sul pulsante dedicato agli studenti e sceglierne uno tra i disponibili nell'applicativo.

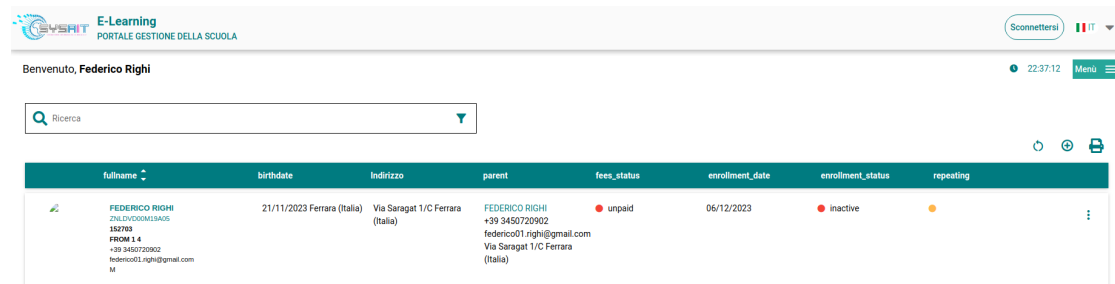
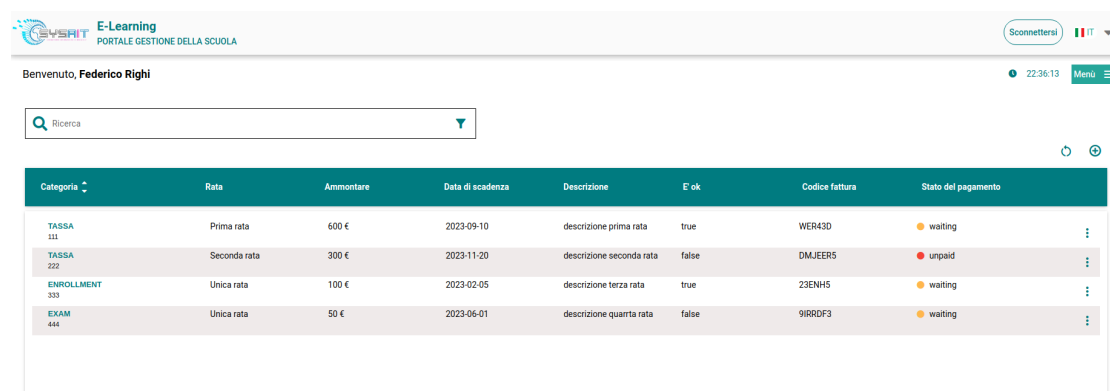


Figura 3.2.2: Pagina di uno studente e relative funzionalità

L'implementazione parte da un file esistente chiamato "Payments.vue" il quale è responsabile della creazione del contenuto della pagina lanciata alla pressione del pulsante "Pagamenti" di uno studente (come indicato dalla Figura 3.2.2). Questo file sarà particolarmente utile anche per la creazione del popup dedicato al pagamento di una specifica rata, per cui il suo contenuto verrà adeguatamente approfondito nella sezione successiva.

In sostanza, il template di questo documento utilizza un componente chiamato "HsTable", pagina preesistente nell'applicativo al momento dell'installazione del progetto front-end locale che contiene tutte le configurazioni necessarie per rendere visibile sullo schermo una tabella. Andando ad analizzare più approfonditamente questo file, esso è composto da ulteriori componenti che, adoperati insieme, forniscono un'idea precisa di tabella utilizzata in tutte le parti della piattaforma; questa mossa rappresenta indubbiamente una notevole strategia per risparmiare codice e di limitare gli errori, mantenendo la limitazione ed eliminazione della ridondanza come uno dei capisaldi della programmazione del progetto.

Quindi, cliccando sul pulsante "Pagamenti", si viene indirizzati nella pagina dedicata alle diverse rate che devono essere pagate dallo studente preso in considerazione; questi dati, naturalmente, vengono definiti nel back-end e vengono resi disponibili alla parte di front-end e quindi renderizzati attraverso opportune chiamate API. Una "chiamata API" (*Application Programming Interface*) è una richiesta inoltrata da un'applicazione o programma verso un endpoint, ovvero un altro programma o servizio web che rappresenta la sorgente delle risorse desiderate; Il servizio o l'applicazione riceve la richiesta, elabora le informazioni fornite e restituisce una risposta al chiamante. I dati sono posizionati in una tabella (grazie all'utilizzo del componente HsTable nella pagina "Payments.vue") suddivisi per attributi specifici in diverse colonne.



The screenshot shows the E-Learning portal interface. At the top, there's a header with the logo, "E-Learning PORTALE GESTIONE DELLA SCUOLA", a login button "Sconnettersi", and a language selector "IT". Below the header, a greeting "Benvenuto, Federico Righi" is displayed. A search bar with the placeholder "Ricerca" is present. The main content area features a table with the following data:

Categoria	Rata	Ammontare	Data di scadenza	Descrizione	E' ok	Codice fattura	Stato del pagamento
TASSA 111	Prima rata	600 €	2023-09-10	descrizione prima rata	true	WER43D	waiting
TASSA 222	Seconda rata	300 €	2023-11-20	descrizione seconda rata	false	DMJEER5	unpaid
ENROLLMENT 333	Unica rata	100 €	2023-02-05	descrizione terza rata	true	23ENH5	waiting
EXAM 444	Unica rata	50 €	2023-06-01	descrizione quarta rata	false	9IRRDF3	waiting

Figura 3.2.3: tabella delle rate da pagare per uno specifico studente

Gli attributi di ogni rata sono quelli riportati nella figura precedente; in particolare, 'categoria', 'rata', 'descrizione', 'codice fattura' e 'stato del pagamento' sono di tipo stringa, 'data di scadenza' di tipo date, 'ammontare' di tipo intero ed 'è ok' è una variabile booleana.

In fondo ad ogni riga della tabella, sono presenti tre puntini di selezione che, se cliccati, fanno comparire delle opzioni: stampa, modifica e cancellazione. Il bottone "cancellazione" è abilitato in tutte le parti dell'applicativo e già funzionante, la stampa è stato creato ma non abilitato e quindi non funzionante, mentre la modifica viene utilizzato in altre sezioni della piattaforma ed è uno degli obiettivi quello di implementare correttamente la sua applicazione per le singole rate da pagare.

La creazione di questi pulsanti avviene nel file javascript "context.js" che definisce una serie di funzioni che creano contesti specifici (context) basati su determinati parametri. Ogni contesto fornisce un insieme di "elementi" che possono essere utilizzati per generare interfacce utente dinamiche, come appunto pulsanti o informazioni. Nel nostro caso, i pulsanti citati vengono definiti nella seguente funzione:

```
const globalElements = (data) => {
  return [
    { event: 'print', icon: 'fas fa-print', label: i18n.t('print'), data },
    { event: 'update', icon: 'fas fa-pen', label: i18n.t('update'), data },
    { event: 'delete', icon: 'fas fa-trash', label: i18n.t('delete'), iconColor: 'red', data },
  ]
}
```

Figura 3.2.4: funzione per la definizione dei pulsanti globali

In sostanza, è stata definita una funzione "globalElements" per generare elementi comuni a più contesti, quindi per rendere questi pulsanti disponibili ovunque. E' importante sottolineare che ogni bottone dell'applicativo lancia uno specifico evento, da come si può evincere dal codice espresso dalla Figura 3.2.4; oltre all'evento, per ognuno di essi viene specificata una determinata icona (la figura affianco al nome del pulsante), la relativa traduzione della descrizione dei pulsanti utilizzando la sezione dedicata ai file di traduzione (importata da "context.js") e un oggetto contenente informazioni aggiuntive che vengono utilizzate per definire le proprietà degli elementi dell'interfaccia utente (*data*).

Il primo passo concreto per la realizzazione del popup per l'aggiornamento di una rata è stato quello di creare un file apposito che mostrasse il contenuto desiderato, denominato "Form-Fees.vue". Il nome non è stato assegnato casualmente.

In precedenza è stato nominato "HsTable" come componente atto alla creazione di una tabella. All'interno della sua sezione template, viene definito un altro componente fondamentale, ovvero "FormPopup", che ha la funzione specifica di definire un componente Vue che rappresenta un popup modale per la gestione dei record all'interno dell'applicazione. Il suo template utilizza ulteriori componenti:

- *HsModal*: per visualizzare il popup;
- *ManageRecord*: gestisce le azioni relative al salvataggio e all'annullamento del form;
- *component*: utilizzato per rendere dinamico il tipo di componente da renderizzare all'interno del componente FormPopup;
- *AdminMonitoring*: gestisce la visualizzazione di un modal (componente dell'interfaccia utente che si sovrappone al contenuto principale della pagina web) per l'aggiornamento dei dati di monitoraggio;

Esso viene lanciato tramite "HsTable.vue" quando viene premuto un determinato pulsante, il quale scatena un preciso evento che viene analizzato attraverso le funzioni personalizzate create nella sezione 'methods' del tag script proprio all'interno di questo file. Ognuna di esse viene realizzata e dedicata per specifico evento (es. delete, update). Quando una di queste funzioni realizza che l'evento lanciato è quello per cui sono dedicate, viene chiamata la funzione 'TBL\_opendForm', responsabile dell'apertura del form quando viene chiamata.

```
TBL_opendForm(event, data = {}){
  this.formProps['formParams'] = this.bodyProps['formParams'] || {};
  this.formProps = {
    ...this.formProps,
    ...(data || {}),
    event: event,
  }
  this.formProps.openForm = true;
},
```

Figura 3.2.5: funzione TBL\_opendForm definita in HsTable.vue

Essa svolge le seguenti azioni:

- Inizializza il *formProps* dell'istanza componente con un oggetto vuoto se non è già definito
  - *formProps* è un'astrazione utilizzata per incapsulare tutte le informazioni necessarie per gestire lo stato e il comportamento del form all'interno del componente "HsTable".  
Definisce:
    - \* *openForm*: Un flag booleano che indica se il form deve essere aperto o chiuso;
    - \* *title*: Il titolo del form da mostrare nell'intestazione del modulo;
    - \* *event*: Il tipo di evento che ha innescato l'apertura del form
    - \* *formParams*: Un oggetto contenente i parametri del form, come i dati del modulo stesso o altre configurazioni specifiche del form
- Combina i dati passati alla funzione con l'oggetto *formProps*
- Imposta la chiave event di *formProps* con il valore fornito come argomento della funzione, che rappresenta il tipo di evento che ha innescato l'apertura del form
- Imposta la proprietà openForm di *formProps* su 'true', indicando che il form deve essere aperto.

Dopodichè viene lanciato "FormPopup" soddisfacendo la condizione espressa nella direttiva 'v-if', come mostrato in figura:

```
<FormPopup
  v-if="formProps.openForm"
  :key="`FormPopup${formProps.openForm}`"
  v-bind="formProps"
  @modalClosed="formClosed"
/>
```

Figura 3.2.6: componente FormPopup nel template di HsTable.vue

“FormPopup” riesce ad innescare uno specifico form grazie alle istruzioni del metodo del ciclo di vita ‘mounted’: esso infatti importa il componente del form corrispondente utilizzando l’argomento passato al componente come parte del nome del componente, imposta il componente corrente (*currentForm*) con il componente del form importato, esegue una funzione di fetch (*fetch\_fn*) per ottenere i dati relativi al form e assegna i dati ottenuti dal fetch alla variabile *formProp* per essere utilizzati nel form. Inoltre, nella sezione ‘computed’ viene definito una funzione dedicata alla renderizzazione del titolo del form. Nel template, per lanciare dinamicamente un form, esso viene specificato tramite il tag “component” per indicare un generico form che verrà lanciato.

```
async mounted(){
  const { default: defaultData } = await
  import('src/components/forms/Form${this.argument.charAt(0).toUpperCase() +
  this.argument.slice(1)}.vue');
  this.currentForm = defaultData;
  console.log('Current Form:', this.currentForm);
  this.fetch_fn();
},

popupFormTitle() {
  return this.formParams && this.formParams.attribute ? '' : !this.isNotForm &&
  this.$18n.t(`${this.argument}FormTitle`).concat(' ${this.id || ''}');
},
```

Per questo motivo il nome del nuovo form, contrassegnato dalla parola ‘Fees’.

Alla base del lavoro svolto è fondamentale rispettare queste regole:

- La modalità di scrittura del codice e le relative scelte implementative devono produrre delle pagine con assenza di ridondanza di codice;
- Garantire compattezza, comprensibilità e pulizia nel codice scritto, ad esempio:
  - indentazione corretta dei tag;
  - istruzioni per personalizzazione dello stile degli elementi del template solo all’interno del tag style dedicato al css;
- Il contenuto dei form deve essere *responsive*, ovvero adattarsi a qualsiasi tipo di dispositivo (in particolare dispositivi mobili come tablet e smartphone) per fornire un’esperienza utente coerente su tutte le piattaforme.

La scrittura del codice del file *FormFees.vue* si articola nella progettazione del template, dello script e nella gestione dello style:



## 1) Template

```
<template>
  <div :class="`${$options.name} column`" v-if="showInputs" :key="formKey">
    <InputComponent
      :key="0"
      class="col col-xs-12 col-sm-12 col-md-12"
      v-bind="inputs[0]"
      @updated="FM_updateValue"
    />
    <InputComponent
      :key="1"
      class="col col-xs-12 col-sm-12 col-md-12"
      v-bind="inputs[1]"
      @updated="FM_updateValue"
    />
    <div class="row">
      <InputComponent
        v-for="(ipt, index) in inputs.slice(2, 5)"
        :key="index"
        class="row_col col-xs-12 col-sm-12 col-md-4"
        v-bind="ipt"
        :IconColor="'primary'"
        :IconSize="'16px'"
        @updated="FM_updateValue"
      />
    </div>
    <InputComponent
      :key="5"
      class="col col-xs-12 col-sm-12 col-md-12"
      v-bind="inputs[5]"
      @updated="FM_updateValue"
    />
  </div>
</template>
```

Figura 3.2.7: sezione template di FormFees.vue

- Come suggerisce la politica di Vuejs, gli elementi del template devono essere contenuti in un unico contenitore, in questo caso un div utilizza la direttiva “v-if” per condizionare la sua visibilità alla variabile *showInputs*. Questo significa che se *showInputs* è vero, l'intero blocco div verrà visualizzato, altrimenti sarà nascosto.

- La classe del div è dinamicamente generata utilizzando l'interpolazione delle stringhe ``${options.name}`` column. Questo consente di applicare classi CSS basate sul nome del componente.
- All'interno del blocco div sono inclusi diversi componenti "InputComponent", ognuno dei quali rappresenta un campo di input.
- Ogni "InputComponent" è associato a un elemento dell'array `inputs` tramite la direttiva `v-bind`, che passa le proprietà del campo di input al componente figlio.
  - `inputs` è un array di oggetti che definisce i campi di input del form e viene utilizzato per generare dinamicamente i campi di input nell'interfaccia utente del form; ogni oggetto contiene diverse proprietà che definiscono le caratteristiche del campo di input.
  - Questa lista di oggetti viene definita in una funzione all'interno di un file mixin importato dal form corrente.
- la class degli "InputComponent" è stata impostata in modo tale che il contenuto si adatti alle diverse tipologie di dispositivi che utilizzano il sito web, facendo utilizzo delle regole di Responsive Design presenti sulla documentazione ufficiale di Quasar.
- Nel div contenente "InputComponent" che genera alcuni campi input utilizzando la direttiva `v-for` per iterare sull'array `inputs` si utilizzano delle proprietà per settare il colore e la dimensione dell'icona utilizzata; queste vengono chiamate props.
  - Le *props* sono degli attributi personalizzati che puoi registrare nel componente padre e possono essere passati al componente figlio, quindi con lo scopo di passare dati o configurazioni necessarie per il suo funzionamento.
  - Nel nostro caso, sono state aggiunte all'interno del file 'InputComponent.vue' (padre) le props 'IconSize' e 'IconColor' e, dopo essere state rese disponibili per l'utilizzo nella parte del template apposita del medesimo file, utilizzate da 'FormFees.vue' (figlio)
  - I tipi di dato accettati dalle props sono String, Number, Boolean, Array e Object.
- Quando un campo di input viene aggiornato, viene emesso l'evento `updated`, che viene catturato dal metodo "FM.updateValue".

## 2) Script

```
<script>
import { formMix } from "src/mixins/module_4/mixin.js";

export default {
  name: 'feesForm',
  mixins: [formMix],
  props: {
    formParams: { type: Object, default: () => {} },
  },
  data() {
    return {
      defaultForInput: {
        maxLength: 110,
      },
      is_fineOptions: [ 'true', 'false' ],
    }
  },
  methods: {
    async setupForm() {
      this.parametersUpdatable = [
        { name: 'invoice_cod', isRequired: true },
        { name: 'denomination', isRequired: true },
        { name: 'amount', isRequired: true },
        { name: 'due_date', isRequired: true, showDivLabel: true, isDate: true, regexPattern: this.$hs.constants.REGEX.date, prependIcon: "calendar"},
        { name: 'is_fine', isSelect: true, isRequired: true, options: this.is_fineOptions, mapOptions: true, emitValue: true },
        { name: 'description', isRichText: true },
      ];

      this.filterAttributes();

      this.formKey += 1;
    },
  },
}
</script>
```

Figura 3.2.8: sezione script di FormFees.vue

- Importazione della funzione “formMix” presente all’interno del file mixin specificato dal percorso nella figura, che gestisce la logica del modulo del form fornendo proprietà, metodi e dati utilizzati all’interno del componente.
- Assegnazione del nome (‘feesForm’) al form corrente.
- La proprietà *is\_fineOptions* contiene le opzioni disponibili per il campo di input *is\_fine*
- Il metodo *setupForm* viene utilizzato per definire i parametri dei campi di input utilizzati nel modulo. Questi parametri vengono definiti nell’array *parametersUpdatable*, che contiene oggetti con le informazioni sui singoli campi di input:

- ‘invoice\_cod’ (codice della fattura)
- ‘denomination’ (denominazione)
- ‘amount’ (ammontare)
- ‘due\_date’ (data scadenza)
- ‘is\_fine’ (è ok)
- ‘description’ (descrizione)

Tutti i campi di input tranne ‘is\_fine’ e ‘description’ sono campi obbligatori (*isRequired: true*); ‘is\_fine’ è un menù a cascata (*isSelect: true*) con delle opzioni disponibili (*options: this.is\_fineOptions*), deve emettere un valore (*emitValue: true*) e le opzioni devono essere mappate in un formato specifico (*mapOptions: true*); ‘due\_date’ mostra un’etichetta

(showDivLabel: true), accetta solo date (isDate: true), controlla se la data inserita è valida (regexPatern) e mostra un'icona prima del campo di input (prependIcon); 'description' supporta il testo formattato (es. HTML) (isRichText: true).

- Il metodo *filterAttributes* viene utilizzato per filtrare gli attributi del form e preparare gli oggetti input utilizzati per generare dinamicamente i campi di input nel template.

### 3) Style

```
<style lang="scss" scoped>
.row {
  margin-bottom: 10px;
}

.col {
  margin-right: 20px;
  margin-left: 20px;
}

.row_col {
  padding-right: 20px;
  padding-left: 20px;
}

.invoice_cod{
  font-size: 18px;
  margin-left: 20px;
}

@media (max-width: 400px) {
  .col{
    width: 100% !important;
    margin-right: 0;
    margin-left: 0;
  }
}

@media (max-width: 400px) {
  .row_col{
    width: 100% !important;
    padding-right: 0px;
    padding-left: 0px;
  }
}
</style>
```

Figura 3.2.9: sezione style di FormFees.vue

Infine, le regole di stile definite nel blocco style includono la gestione del layout responsive tramite l'utilizzo dei media query (conferendo determinate impostazioni in base alla larghezza

dello schermo) e per personalizzare gli altri elementi della pagina in modo tale da produrre il risultato finale seguente:

Figura 3.2.10: Popup per l'aggiornamento di una rata

La data riportata nel campo 'due\_date' nella Figura 3.2.8 non è formattata correttamente, in quanto nella tabella delle rate (figura 3.2.3) essa ha il costrutto 'YYYY-MM-DD' mentre il campo nel form rispetta il costrutto 'DD-MM-YYYY'; il risultato che ne consegue è che nel campo del form viene riportata la data di interesse secondo il formato rispettato dalla tabella delle rate, non facendo altro che separare le cifre con il simbolo '/' secondo il costrutto 'DD-MM-YYYY' (questo è un problema che deve essere gestito lato back-end, quindi è stato deciso di sorvolarlo).

Per quanto concerne lo sviluppo del popup per la creazione di una nuova rata, non è stato necessario creare un ulteriore form, bensì viene sfruttato il codice della pagina "FormFees.vue" portando al seguente risultato: quando viene cliccato il pulsante + in alto a destra rispetto alla tabella delle rate, si apre una finestra popup il cui contenuto rispecchia quello del popup per l'aggiornamento di una rata, visualizzando sullo schermo gli stessi campi, con la sola differenza che questi ultimi sono vuoti e pronti per essere compilati. Naturalmente, al loro riempimento verrà attivato il pulsante "Salvare" in fondo al form e sarà possibile creare una nuova rata.

Questo procedimento è reso possibile grazie al metodo 'SUB\_new' all'interno di 'HsTable.vue', il quale controlla se l'evento ricevuto alla pressione del pulsante sia uguale a "new" e, in caso affermativo, si rivolge alla funzione 'TBL\_opendForm', la quale è incaricata di impostare la variabile booleana 'opendform' a 'true' contenuta all'interno di formProps e permettendo di lanciare FormPopup esaudendo la condizione espressa nella sua direttiva v-if; successivamente, FormPopup controllerà l'argomento e riuscirà in questo caso a lanciare la pagina 'FormFees.vue' dinamicamente.

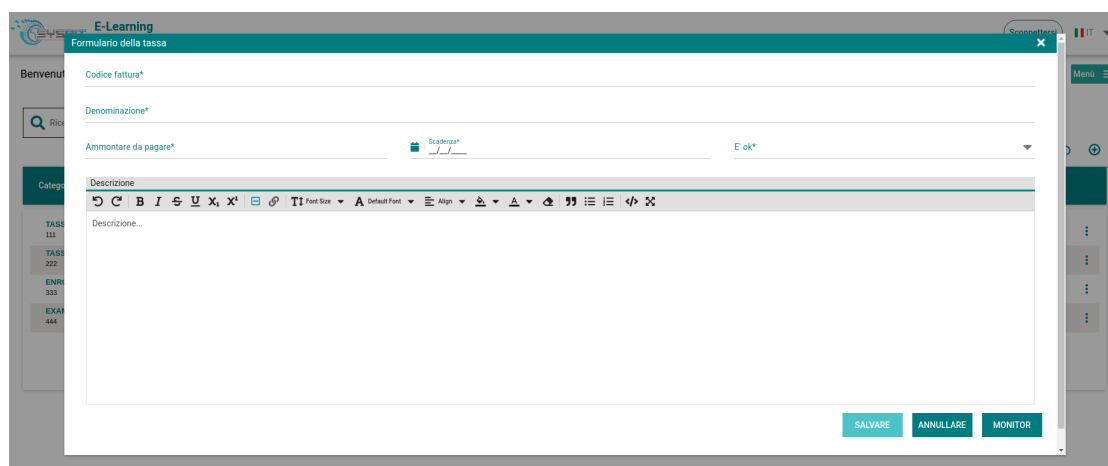


Figura 3.2.11: Popup per la creazione di una rata

Per garantire un supporto multilingua adeguato, è stato necessario creare un oggetto Javascript che contiene le traduzioni o le etichette per i campi dei form presi in analisi; ogni chiave dell'oggetto rappresenta il nome del campo del form, mentre il valore associato a ciascuna chiave è la traduzione o l'etichetta corrispondente per quel campo. Il procedimento è stato svolto per la lingua italiana, francese ed inglese nei file delle apposite sotto-directory contenute nella directory il8n:

```
feesFormTitle: "Formulario della tassa",
feesForm: {
  denomination: "Denominazione",
  amount: "Ammontare da pagare",
  due_date: "Scadenza",
  is_fine: "E' ok",
  invoice_cod: "Codice fattura",
  description: "Descrizione",
},
```

Figura 3.2.12: Traduzione italiana dei campi del form

```
feesFormTitle: "Formulaire d'impôt",
feesForm: {
  denomination: "Nom",
  amount: "Montant",
  due_date: "Expiration",
  is_fine: "C'est bien",
  invoice_cod: "Code de facture",
  description: "Description",
},
```

Figura 3.2.13: Traduzione francese dei campi del form

```
feesFormTitle: "Tax form",
feesForm: {
  denomination: "Denomination",
  amount: "Amount",
  due_date: "Due date",
  is_fine: "Is fine",
  invoice_cod: "Invoice cod",
  description: "Description",
},
```

Figura 3.2.14: Traduzione inglese dei campi del form

### 3.3 Implementazione del form di pagamento di una rata specifica

L'obiettivo riguardante la creazione del form per il pagamento è stato raggiunto prendendo come riferimento il diagramma di flusso che descrive i passaggi che portano uno studente al corretto pagamento di una rata nella piattaforma:

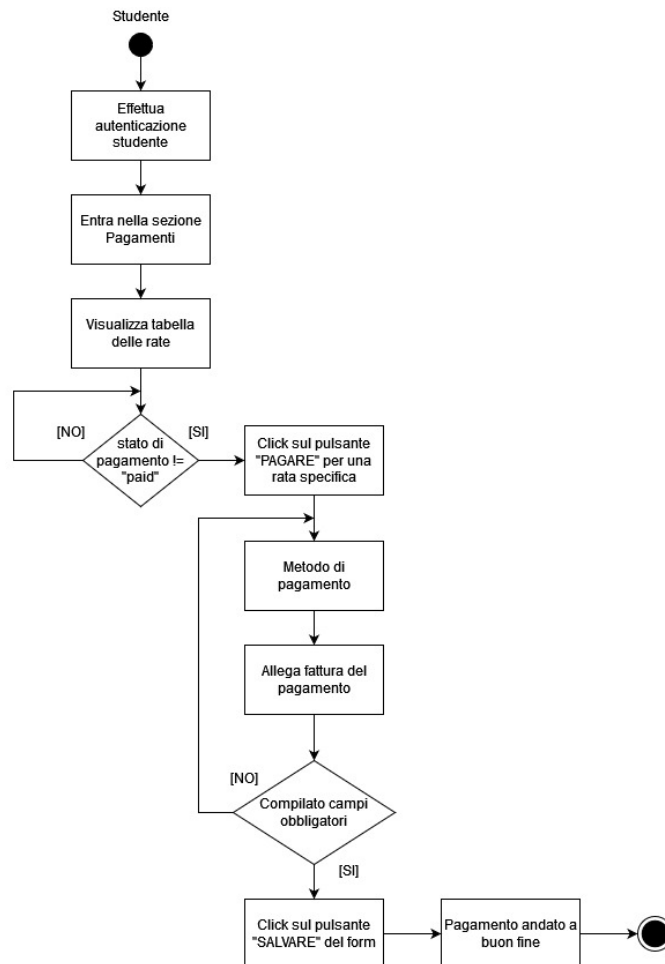


Figura 3.3.1: Diagramma di flusso per il pagamento di una rata

La realizzazione di tale diagramma è stata effettuata tramite il sito *draw.io*.

L'implementazione del form per il pagamento di una rata nasce dall'amministrazione del comportamento di diversi file tra cui "Payments.vue" e "HsTable.vue" visti in precedenza, adattandoli al contesto e implementandovi le funzionalità necessarie.

La decisione che è stata presa riguardo a questa implementazione è di ascoltare l'evento lanciato dalla pressione del pulsante "PAGARE" (tramite una funzione in HsTable) direttamente dal genitore "Payments.vue", in modo tale da non dover intaccare la sezione template di HsTable,



in quanto è un componente utilizzato in diverse parti del sito ed evitare così eventuali problemi nelle future implementazioni.

Nel file “context.js” è stata creata una funzione per la gestione degli eventi chiamata “feesContext” la quale, in base al valore del parametro `argument`, crea un array di elementi; in tal caso, se l’argomento è uguale a “fees” viene creato un array vuoto e definita una variabile “data”, inizializzata con parametri passati e alcune informazioni come:

- *to\_pay*: variabile booleana che indica se è necessario effettuare un pagamento;
- *event*: evento associato all’azione che verrà eseguita quando si interagisce con gli elementi di “feesContext”.

Nel caso corrente, se la variabile *to\_pay* agganciata alla costante `data` riporta valore “true”, allora viene inserito all’interno dell’array vuoto un array di elementi ospitante un pulsante che, se premuto, lancia l’evento “emit”, assegnandogli ulteriori caratteristiche. All’array `elements` vengono aggiunti i pulsanti definiti nella funzione “globalElements”, il cui contenuto è esplicitato nella Figura 3.2.4).

```
allContexts["feesContext"] = (params = {}) => {
  const { argument } = params;
  if (argument !== 'fees') return;
  const data = {...params, event: 'pay' };

  let OtherBtns = [];
  if(data.to_pay){
    OtherBtns.push({
      elements: [
        { event: 'emit', icon: 'fas fa-credit-card', label: i18n.t('pay'), iconColor: 'blue', data },
      ]
    })
  }
  return [
    {
      elements: globalElements(data)
    },
    ...OtherBtns
  ]
}
```

Figura 3.3.2: creazione pulsante per il pagamento di una rata

E’ importante sottolineare che la variabile *to\_pay* deriva dal progetto di back-end e assume valore “true” se lo stato di pagamento è uguale ad “unpaid” o “waiting” (contrassegnati rispettivamente da un cerchio rosso e giallo nella tabella della Figura 3.2.3) facendo comparire il pulsante in questione tra le opzioni disponibili nel menù a tendina alla pressione dei tre puntini, diversamente nel caso in cui lo stato di pagamento ha valore “paid” (quindi contrassegnato da un cerchio verde) per cui il valore di *to\_pay* risulta “false”.

Dopo la creazione del pulsante specificato, i passaggi che sono stati seguiti per lanciare il form atto al pagamento di una rata sono i seguenti:

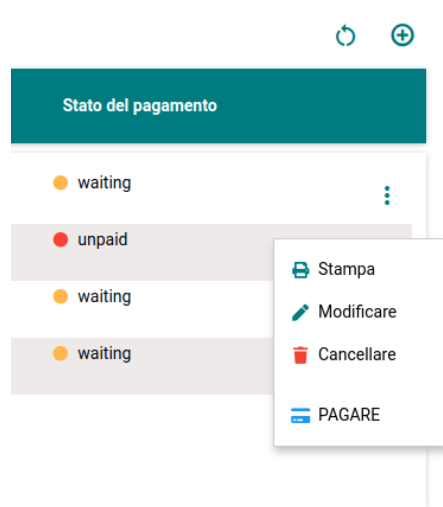


Figura 3.3.3: pulsanti disponibili per stato di pagamento “waiting” e “unpaid”

- In “HsTable.vue”, viene creato una funzione “TBL\_emit” che è incaricata di controllare se l’evento lanciato è uguale ad “emit” e, in caso affermativo, chiama il metodo `$hs.methods.COMMON_emitEvent` che accetta due argomenti: *event*, che rappresenta il tipo di evento da emettere, e *detail*, che contiene eventuali dettagli aggiuntivi associati all’evento. Se a questo metodo è stato fornito un tipo di evento, viene creato un nuovo oggetto “CustomEvent” con il tipo di evento specificato e i dettagli forniti; tale metodo verrà dispacciato sulla finestra corrente utilizzando `window.dispatchEvent`, in modo che altri componenti o parti del codice all’interno dell’applicazione possano ascoltare e reagire a quell’evento.

- Lo script del file “Payments.vue” (genitore di HsTable) viene impostato in modo tale da catturare l’evento lanciato tramite i lifecycle hooks *created*, che aggiunge un listener per l’evento personalizzato “pay” richiamando il metodo “openFormPayment” quando l’evento viene emesso, e *destroy*, che rimuove il listener per l’evento “pay”.

Nella sezione *methods*, Il metodo “openFormPayment” gestisce l’apertura del popup di pagamento, popolando le proprietà del popup con i dati forniti e impostando *openForm* su “true”, mentre *formClosed* gestisce la chiusura del popup di pagamento impostando *openForm* su “false”.

*openForm* è uno degli elementi di *formProps* (spiegato nella sezione precedente), il quale viene definito allo stesso modo del file “HsTable.vue”.

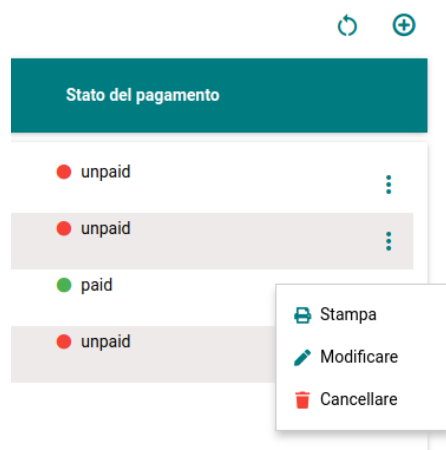


Figura 3.3.4: pulsanti disponibili per stato di pagamento “paid”

```
TBL_emit(response){
  const {event, data} = response;
  if(event !== 'emit') return

  this.$hs.methods.COMMON_emitEvent(data.event, data);
},
```

Figura 3.3.5: funzione TBL\_emit

- Nel template di “Payments.vue” viene introdotto il componente FormPopupPayments, che viene lanciato se il valore di *openForm* risulta “true”. Oltre questo, l’attributo *key* serve a garantire che il componente venga correttamente aggiornato quando cambiano le sue proprietà: in questo caso, la chiave viene costruita concatenando la stringa “Payments” con il valore di *formProps.openForm* e *formProps.id*, assicurando che, quando *formProps.openForm* o *formProps.id* cambiano, il componente FormPopupPayments verrà correttamente aggiornato; l’attributo *v-bind* collega le proprietà definite nell’oggetto *formProps* come props del componente FormPopupPayments, passando tutte le proprietà di *formProps* come props al componente figlio; l’ascoltatore dell’evento personalizzato “modal-Closed” serve per chiamare il metodo *formClosed* definito nel componente padre quando l’evento viene emesso.

FormPopupPayments ha lo stesso medesimo comportamento del componente FormPopup (spiegato nella sezione precedente) con la principale differenza che non viene effettuata l’invocazione dinamica di un form, bensì direttamente di quello adibito al pagamento di una rata, tramite la sostituzione nel template del tag “component” con il nome del form, mentre nello script è stato eliminato l’attributo *currentForm* e il lifecycle hooks *mounted* con il relativo contenuto.

```
const COMMON_emitEvent = (event = null, detail = {}) => {
  if(!event) return

  window.dispatchEvent(new CustomEvent(event, { detail }))
}
```

Figura 3.3.6: funzione COMMON\_emitEvent

```
created () {
  window.addEventListener('pay', this.openFormPayment);
},
destroy () {
  window.removeEventListener('pay', this.openFormPayment);
},
methods: {
  openFormPayment(data) {
    this.formProps['formParams'] = this.bodyProps['formParams'] || {};
    this.formProps = {
      ...this.formProps,
      ...(data.detail || {}),
    }
    this.formProps.openForm = true;
  },
  formClosed() {
    this.formProps.openForm = false;
  }
}
```

Figura 3.3.7: metodi di Payments.vue

Così facendo, FormPopupPayments sarà in grado di renderizzare il form interessato, denominato FormPayments.

Esso è stato progettato nel seguente modo:

### 1) Template

Costruzione di una tabella riportante le informazioni principali ('codice fattura', 'categoria', 'rata', 'ammontare', 'data di scadenza') inerenti ad una specifica rata da pagare sulla sinistra del popup, mentre sulla destra è stata realizzata una sezione dedicata alla selezione del metodo di pagamento e uno spazio per deporre allegati della fattura.

Importante sottolineare che la progettazione del template di tale form segue le regole di implementazione di codice utilizzate per "FormFees" spiegate nella sezione precedente.

```

<template>
  <q-page :class="`${options.name}`" padding>
    <HsTable
      v-if="renderTable"
      v-bind="tableProps"
      :bodyProps="bodyProps"
      :key="`${options.name}${renderTable}`"
    />
    <FormPopupPayments
      v-if="formProps.openForm"
      :key="`Payments${formProps.openForm}${formProps.id}`"
      v-bind="formProps"
      @modalClosed="formClosed"
    />
  </q-page>
</template>

```

Figura 3.3.8: template di Payments.vue

## 2) Script

- importa dei mixins da cui si possono ricavare le props di interesse e alcuni metodi e funzioni utili per una logica condivisa;
- Nella sezione data vengono definiti due array (*displayedFormProps* e *paymentTypesOptions*) inizializzati vuoti, *defaultForInput* che specifica la lunghezza massima del testo per i campi di input e un array denominato *Attachments* che contiene un oggetto rappresentante gli allegati associati al modulo. Oltre queste informazioni, data definisce:
  - *getCategories()*: metodo utilizzato per recuperare le categorie di pagamento tramite una chiamata API asincrona (*/schools/payment\_types*); una volta ottenuti i dati, vengono memorizzati nell'array *paymentTypesOptions* nel data del componente.
  - *setupForm()*: metodo per inizializzare il modulo di input del componente, il quale effettua una chiamata a *getCategories()* per ricevere, se necessario, le categorie di pagamento e configura i *displayedFormProps*, ovvero le proprietà che vengono renderizzate nella tabella alla sinistra del popup. Inoltre, definisce *parametersUpdatable*, quindi gli elementi che possono essere aggiornati nel modulo di input (in questo caso, l'array *parametersUpdatable* è riempito con solo il 'payment\_type\_id', che si riferisce alla select riportante le tipologie del metodo di pagamento).

Infine, filtra gli attributi del modulo di input tramite *filterAttributes()*, configura gli allegati tramite *setupAttachments()* e incrementa *formKey* per forzare il re-rendering del componente (processo di aggiornamento della visualizzazione del componente sulla pagina web in risposta a cambiamenti nei dati o nello stato del componente stesso). Questi metodi derivano dai file mixins importati.

```

data() {
  return {
    displayedFormProps: [],
    defaultForInput: {
      maxLength: 300,
    },
    Attachements: [
      {
        name: "invoice_file"
      }
    ],
    paymentTypesOptions: []
  }
},
methods: {
  async getCategories(){
    const paymentTypesResp = await this.$hs.methods.ajaxCall('schools/payment_types', "get", { hideLoader: true });
    this.paymentTypesOptions = paymentTypesResp.data;
  },
  async setupForm() {
    await this.getCategories();
    this.displayedFormProps = [
      { key: 'category', label: this.$t('category') },
      { key: 'installment', label: this.$t('installment') },
      { key: 'amount', label: this.$t('amount') },
      { key: 'due_date', label: this.$t('due_date') },
    ];
    this.parametersUpdatable = [
      { name: 'payment_type_id', isSelect: true, options: this.paymentTypesOptions, mapOptions: true, emitValue: true, prependIcon: "money-check" },
    ];
    console.log("argument: ", this.argument);
    this.filterAttributes();
    this.setupAttachments();
    this.formKey += 1;
  },
}

```

Figura 3.3.9: sezione data del tag script di FormPayments.vue

### 3) Style

Personalizzazione delle tabelle e elementi utilizzati nel template, servendosi della gestione del layout responsive tramite l'utilizzo dei media query come nel caso FormFees.

The screenshot shows a web application interface for E-Learning. The main window is titled "Formulario del pagamento". On the left, there is a sidebar with a search bar and a list of categories. The main content area is titled "Dettagli fattura" and contains a table with the following data:

Categoria	Tassa
Rata <td>Seconda rata</td>	Seconda rata
Ammontare	300 €
Data di scadenza	2023-11-20

Below the table, there is a modal window titled "Metodo di pagamento" with the following fields:

- Nome del file\*
- Link del file\*
- Tipologia PDF\*

At the bottom of the modal, there are four buttons: SALVARE, ANNULLARE, REINIZIALIZZARE, and MONITOR.

Figura 3.3.10: form per il pagamento di una rata

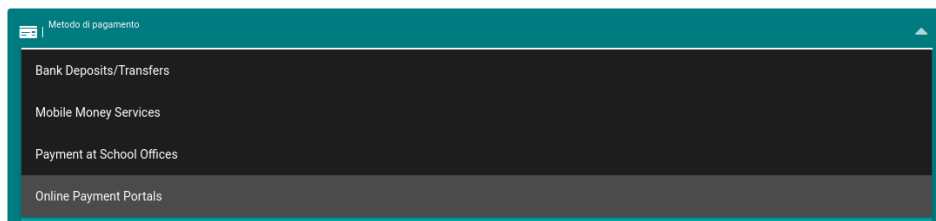


Figura 3.3.11: metodi di pagamento disponibili

Ci sono ulteriori precisazioni fondamentali da aggiungere:

- Per quanto riguarda i campi di input, per colorare il logo del metodo di pagamento e gli altri campi che si riferiscono alle informazioni dell'allegato senza prorogare tali aggiornamenti agli altri elementi dell'applicativo, quindi agendo solo per il form corrente, è stato essenziale creare delle nuove props (*dark*, *color* e utilizzando le props create in precedenza *IconColor* e *IconSize*) all'interno del componente "InputComponent" rendendole disponibili ai file che utilizzeranno tale componente (come nel caso del form per l'aggiornamento di una rata) e impostarne il comportamento desiderato all'interno del file FormPayments:

```
<InputComponent
  v-for="(ipt, index) in inputs"
  :key="index"
  class="col"
  :qsep="'bg-white'"
  :labelcolor="'white'"
  :dark="true"
  :color="'white'"
  :IconColor="'white'"
  :IconSize="'16px'"
  :argument="argument"
  v-bind="ipt"
  @updated="FM_updateValue"
/>
```

Figura 3.3.12: configurazioni campi di input nel template di FormPayments.vue

- Uno degli obiettivi consiste nel poter allegare la sola tipologia di file “pdf” per il selettore di file della fattura, senza inoltrare tali modifiche agli altri file pickers dell'applicativo. Per fare questo, è stato opportuno effettuare delle implementazioni ai componenti seguenti, i quali sono stati realizzati per rappresentare l'elemento file picker nel progetto:
  - *GlobalFilePicker*: definizione e passaggio dell'argument come props al componente figlio FilePicker;
  - *FilePicker*: definizione e passaggio dell'argument e di un array denominato types (inizializzato con il solo valore ‘pdf’) come props al componente figlio LinkFilePicker;

- *LinkFilePicker*: definizione e passaggio dell’argument e dell’array “types” come props al figlio *FormLinkFile*;
- *FormLinkFile*: nella sezione *setupForm()* dello script, viene chiamato un metodo passandogli come argomenti l’argument, l’array type e il nome del selettore (*content\_type* in questo caso).

```
const options =
  this.$hs.methods.COMMON_makeSelectOptions
  ({ name: 'content_type', argument: this.argument,
    types: this.types });
```

Questo metodo (presente nel file chiamato *functions.js* contenente diverse funzioni utilizzabili dai vari componenti) si occupa di costruire le opzioni per l’elemento di selezione in base ai parametri forniti, controllando se il nome del selettore è definito e principalmente chiamando il metodo *Common\_options* passandogli come argomenti il nome, l’argument e types.

```
const COMMON_makeSelectOptions = ( params = {} ) => {
  const { name, argument, types } = params;
  if (!name) return;
  let localOptions = [];

  let tmpOptions = COMMON_options({ name , argument, types });
  localOptions = tmpOptions.map(opt => {return {...opt, label: i18n.t(opt.label)}});

  return localOptions;
}
```

Figura 3.3.13: metodo per costruzione opzioni per una select

Tale metodo (implementato nel file *selectOptions.js*) accetta un oggetto ‘params’ come argomento composto dalle seguenti chiavi: name, argument e types. Viene effettuato nuovamente il controllo sul nome e, se definito, viene creato un array vuoto e si controlla se l’array types contiene qualcosa e che l’argument sia uguale a ‘fees’ (adattando le operazioni seguenti solo alla pagina del sito interessata); in caso affermativo, avviene un’operazione di filtraggio inserendo all’interno dell’array vuoto precedentemente creato solo gli elementi contenuti all’interno di types e il metodo restituisce l’array filtrato, altrimenti viene restituito l’intero array “CONTENT\_TYPE”, definito nel file *constants.js* (importato da *selectOptions.js*) che contiene tutti i tipi di file allegabili nell’intero sito web (‘image’, ‘video’, ‘youtube’, ‘audio’, ‘pdf’).



```
const CONTENT_TYPE = [
  { label: "IMAGE", value: 'image' },
  { label: "VIDEO", value: 'video' },
  { label: "AUDIO", value: 'audio' },
  { label: "YOUTUBE", value: 'youtube' },
  { label: "PDF", value: 'pdf' }
]
```

Figura 3.3.14: CONTENT\_TYPE

- All'interno del componente LinkFilePicker, diversamente rispetto agli altri file pickers delle altre pagine, è stato abilitato il pulsante "Cancella tutto" che compare affianco all'etichetta del file picker per la fattura nella figura 3.3.10; in pratica, dopo aver premuto il pulsante per salvare, i files vengono memorizzati in un array chiamato "localFiles" e per conferire l'abilità di cancellare tutti gli elementi da tale array al pulsante in analisi occorre controllare se l'evento scatenato è uguale a 'deleteAll' (ovvero il nome dell'evento dato a questo pulsante in fase di costruzione) e, se sì, lo svuota. In seguito sono illustrate le porzioni di codice che raffigurano il comportamento analizzato e i pulsanti principali del file picker:

```
if (data.event === 'deleteAll') {
  this.localFiles = [];
  this.opened = false;
}

buttons: [
{
  icon: "fas sysait-circle-plus-l text-white",
  tooltip: "add",
  event: 'new',
  size: "md",
},
{
  icon: "fas fa-trash-alt",
  tooltip: "deleteAll",
  textColor: "red",
  size: "sm",
  event: 'deleteAll'
}
],
```

```

import { STATUSES_ARRAY, CONTENT_TYPE } from 'src/mixins/module_1/constants.js';
let allOptions = [];

const COMMON_options = (params = {}) => {
  return allOptions[params.name](params);
}

allOptions['status'] = (params = {}) => {
  const { name } = params;
  if (name !== 'status') return;

  return STATUSES_ARRAY;
}

allOptions['content_type'] = (params = {}) => {
  const { name, argument, types } = params;
  if (name !== 'content_type') return;

  let resp = [];
  if (types?.length && argument === 'fees') {
    resp = CONTENT_TYPE.filter(item => types.includes(item.value));
    return resp;
  }

  return CONTENT_TYPE;
}

allOptions['category'] = (params = {}) => {
  const { name, argument } = params;
  let arrayOfOptions = [];
  if (name !== 'category') return;

  return arrayOfOptions;
}

export {
  COMMON_options,
}

```

Figura 3.3.15: contenuto file selectOptions.js

- Nel componente FormLinkFile, che si occupa di renderizzare le informazioni per allegare i files e i pulsanti per salvare un allegato e chiudere il file picker, è stata implementata la funzionalità per lanciare un errore in caso di campo non compilato e di disattivare il pulsante per salvare fino a quando tutti gli input non vengono compilati.

Per il pulsante è stato configurato il componente “ButtonComponent” utilizzato all’interno di FormLinkFile nella maniera seguente:

```

<ButtonComponent @click="handleClick" btnClass="bg-primary" class="q-ml-sm"
  v-bind="button" :disable="isSaveButtonDisabled && isSaveButton(button)">
  <q-tooltip v-if="button.tooltip"
    content-class="bg-primary_variant_5 text-white"
    content-style="font-size: 12px">{{ $t(button.tooltip) }}</q-tooltip>
</ButtonComponent>

```

Utilizzo la proprietà `disable` che chiama una funzione che fa ritornare “true” o “false” analizzando se i campi di input sono riempiti o meno:

```
isSaveButtonDisabled() {  
    return this.inputs.some(input => !input.model);  
}
```

Per quanto riguarda gli errori lanciati dagli input, basta aggiungere una props chiamata *rules* creata nel file “InputComponent.vue” che rappresenta un’array di funzioni di validazione, controllando se il valore inserito nel campo di input è vuoto o meno e, in base a quello, lanciare un errore colorando l’input di rosso. Un esempio per il campo di input per il nome del file:

```
{ name: "filename", model: this.form.filename, isRequired: true,  
  rules: [ val => !!val || '' ], labelKey: 'linkFileForm.filename' },
```

The screenshot shows a web form titled "Metodo di pagamento". At the top right, there is a button "Inserire il File della fattura" with a plus icon and a red error icon. Below this, there are three input fields. The first field is labeled "Nome del file\*" and is red with a red error icon. The second field is labeled "Link del file\*" and is also red with a red error icon. The third field is a dropdown menu labeled "Tipologia PDF" with a red error icon. To the right of the dropdown menu is a red button with a white 'X' icon, which is disabled.

Figura 3.3.16: errore campi di input e pulsante per salvare disabilitato

- L’ultimo passo per completare il lavoro riguarda la personalizzazione degli allegati agendo sul file “AttachmentViewer.vue”, che gestisce la raffigurazione dei files allegati una volta premuto il pulsante per salvare un elemento; l’obiettivo è stato quello di migliorare la grafica di questi ultimi e aggiungendo un’icona personalizzata (consultando il sito di FontAwesome) in base alle varie tipologie di files allegabili, producendo il risultato seguente:

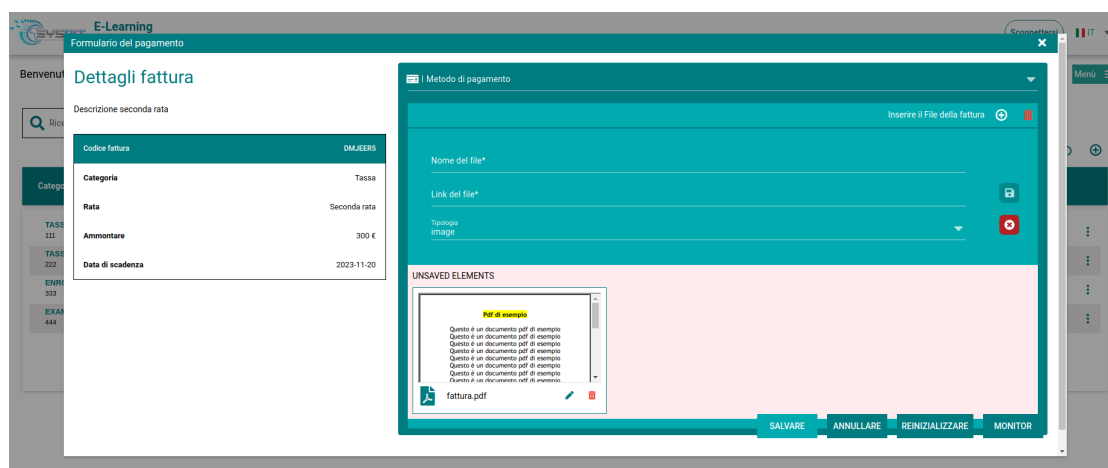


Figura 3.3.17: files allegati in FormPayments.vue

Questo è il risultato finale del lavoro del progetto. Dopo aver inserito degli allegati come nella figura qui sopra e aver cliccato il pulsante “SALVARE”, riaprendo il form si dovrebbero visualizzare come nella figura seguente:

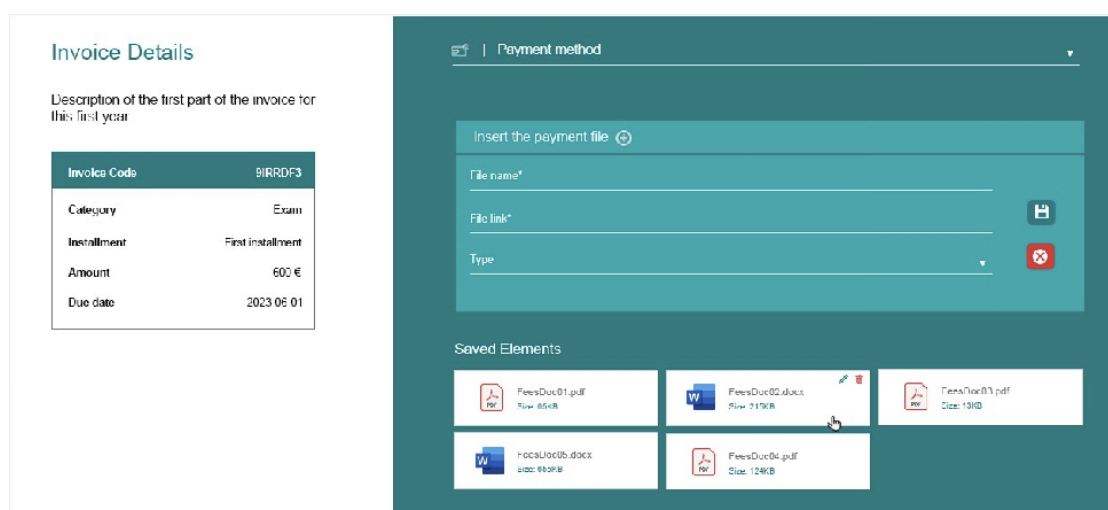


Figura 3.3.18: risultato ideale del form ultimato



## Conclusione

Lo svolgimento del progetto ha contribuito a collocare un tassello importante negli obiettivi preposti da SYSAIT; le tematiche e il forte senso di ambizione che la società sostiene hanno reso stimolante l'approccio al lavoro e il suo compimento.

Nel periodo di lavoro non sono mancati dei rallentamenti, in quanto essendo un lavoro condiviso da più persone, si sono create delle tempistiche di attesa per l'idealizzazione delle implementazioni da apportare al sito, soprattutto legate al lato design del form per il pagamento di una rata.

Nonostante alcune implementazioni mancanti lato back-end in fase di progettazione del modulo per il pagamento di una rata da parte di uno studente si può proseguire, dopo aver ultimato tali mancanze, con lo sviluppo di ulteriori moduli dell'applicativo legati all'operazione di pagamento, ad esempio per la creazione, aggiornamento e visualizzazione delle liste di fatture per gli studenti di una determinata classe, e ulteriori altri moduli che non sono ancora stati realizzati nella piattaforma.



## Sitografia

1. <https://vuejs.org/guide/introduction.html>
2. <https://www.sysaitechnology.com/home>
3. <https://www.geekandjob.com/wiki/bitbucket>
4. <https://kinsta.com/it/knowledgebase/cosa-e-postgresql/>
5. <https://it.wikipedia.org/wiki/PgAdmin>
6. <https://blog.logrocket.com/use-props-pass-data-child-components-vue-3/>
7. <https://v1.quasar.dev/introduction-to-quasar>
8. <https://v2.vuejs.org/v2/guide/mixins.html>
9. <https://quasar.dev/layout/grid/introduction-to-flexbox/>
10. <https://www.html.it/webapp/draw-io-cose-come-si-usa-template-e-integrazioni/>





## Ringraziamenti

L'ultima sezione di questa tesi è dedicata al ringraziamento delle persone che mi sono state vicine in questo capitolo importante.

Ringrazio infinitamente i miei genitori per essermi stati accanto e non aver mai dubitato delle mie capacità, soprattutto nei momenti di maggiore difficoltà, e per rendere ogni giorno della mia vita speciale.

Ringrazio la mia migliore amica Chiara, con cui condivido un forte legame, nell'aver reso questi anni memorabili condividendo molti dei miei valori e per essermi sempre stata vicina incondizionatamente, rendendola una delle persone più importanti della mia vita.

Ringrazio tutti gli amici della mia compagnia con cui ho trascorso dei bei momenti, in particolare modo al mio amico di una vita Simone e Gianluca, con cui ho stretto un grande rapporto.

Ringrazio il mio compagno di facoltà, nonché di scuola media e superiore Sebastiano per aver condiviso questo percorso accademico, supportandoci l'un l'altro; ringrazio anche le persone che ho conosciuto nel mio percorso di studi, Matteo e Lorenzo, per esserci sostenuti in questi anni.

Voglio ringraziare anche me stesso, per essere riuscito a far fronte ai problemi che mi si sono posti davanti in questa triennale, permettendomi di maturare e accrescendo la determinazione e il senso di perseveranza nel raggiungere gli obiettivi.

Infine, ringrazio la professoressa Elena Bellodi e il dottor Nguembang Arnaud Fadja per avermi accompagnato in questo ultimo passo della triennale e per l'esperienza di tirocinio formativa e impegnativa.