



**Università
degli Studi
di Ferrara**

Corso di Laurea in Ingegneria Elettronica e Informatica

**API Ruby on Rails per la digitalizzazione delle tasse
scolastiche nelle scuole secondarie**

**Relatrice:
Prof. Elena Bellodi**

**Laureanda:
Marta Malagutti**

**Secondo Relatore:
Arnaud Nguembang Fadja**

Anno Accademico 2022/2023

Indice

Introduzione	5
1. Scopo del progetto	7
2. Tecnologie utilizzate	10
3. Progettazione del modulo di pagamento	12
4. Implementazione del modulo di pagamento	30
Conclusioni	47
Bibliografia	49
Sitografia	49
Ringraziamenti	51

Introduzione

L'argomento di cui si tratterà nel seguito ha le sue origini in una piattaforma educativa sviluppata da un'azienda nata recentemente, System Afrik Information Technology (SYSAIT). Il progetto al centro della tesi costituisce un contributo a tale piattaforma, in quanto si pone l'obiettivo di realizzare il modulo back-end in grado di consentire il pagamento delle tasse scolastiche da parte degli studenti, iscritti in una scuola secondaria che abbia deciso di aderire. Il nome di questa piattaforma è E-learning: si tratta di un prodotto complesso, ben riuscito e funzionante nelle parti implementate, il cui scopo è quello di fornire un contributo tangibile all'interno del sistema educativo del continente africano, sfortunatamente ancora afflitto da numerose difficoltà.

Tale progetto, però, non è stato il primo ad essere portato a termine: SYSAIT aveva già implementato alcuni tra i moduli dell'applicazione nel momento in cui è stata iniziata questa attività di tesi, pertanto il nuovo modulo è stato contestualizzato e, successivamente, realizzato.

Nel fare ciò sono state soprattutto riprese e applicate alcune delle conoscenze acquisite nell'ambito delle basi di dati, anche se si è reso necessario fare uso di nuove conoscenze, assimilate durante le prime fasi del progetto. Queste riguardano il framework Ruby on Rails e l'utilizzo di nuovi strumenti, come pgAdmin 4, Confluence e Insomnia.

Si può pertanto affermare che questo lavoro costituisca un punto di incontro tra l'ingegneria informatica e la necessità di garantire un significativo supporto all'educazione degli studenti delle scuole secondarie, in modo particolare nei paesi in Africa.

1. Scopo del progetto

Al fine di comprendere a fondo l'obiettivo di tale attività è opportuno specificare le sue origini e il contesto in cui si colloca.

SYSAIT è una giovane azienda nata con l'intento di sviluppare soluzioni digitali innovative e di qualità, adatte a migliorare la vita di tutti i giorni. Nello specifico, l'idea che si trova alla base dell'impresa è quella di favorire il successo del continente africano attraverso lo sviluppo di sistemi digitali per il miglioramento della sanità, del sistema amministrativo ed educativo, di particolare interesse per questa trattazione: al centro di quest'ultima, infatti, è presente E-learning, la piattaforma educativa che SYSAIT sta attualmente progettando. L'azienda ha deciso di intervenire secondo questa modalità, consapevole dello scenario odierno in Africa: l'archiviazione dei dati viene ancora effettuata manualmente e artigianalmente, la pianificazione e l'organizzazione delle attività sono dispendiose, il rapporto tra i genitori e gli insegnanti è distante, oltre a molte altre pratiche che non contribuiscono certamente a dare vita ad un sistema ideale per il corretto funzionamento della vita scolastica.

Proprio in risposta alle precedenti problematiche, come anticipato, è nata l'intenzione di progettare e implementare un sistema per la gestione delle scuole secondarie: si tratta della prima piattaforma globale e completamente cloud del settore, pensata per la gestione di oltre 10000 studenti e 5000 operatori, accessibile ovunque e in qualsiasi momento, multilingue, multi-struttura e multi-servizio, caratteristiche che hanno consentito a SYSAIT di acquisire oltre 100 clienti, anche di grandi dimensioni, a cui garantire servizi come la gestione dell'anagrafica, delle lezioni, delle iscrizioni e dei pagamenti.

Si compone dei seguenti moduli:

- *E-learning*, per la gestione della scuola
- *E-school*, per la presentazione, attraverso un sito, della scuola
- *E-learning-api*, l'API back-end delle due parti precedenti

In altre parole, l'applicazione web in questione include sia una vetrina della scuola sia un'applicazione di gestione basata sul web.

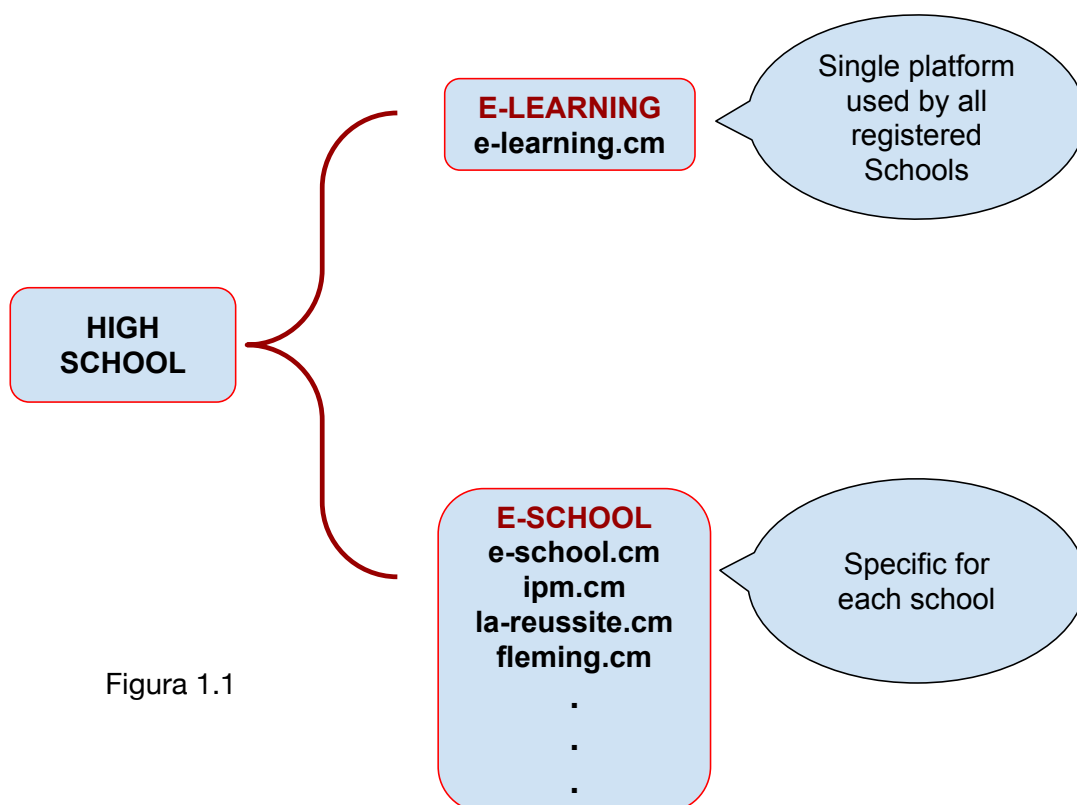


Figura 1.1

Più nel dettaglio, il modulo *E-school* ha come finalità quella di presentare l'organizzazione della scuola nel suo complesso, le attività e i club esistenti al suo interno, la lista dei contatti in caso di necessità, la sede, le ultime novità che la riguardano, la sua storia, gli esami ufficiali, le statistiche e molto altro. Oltre a tutto questo, svolge il ruolo di CMS (*Content Management System*, sistema di gestione dei contenuti).

Il modulo *E-learning*, invece, presenta numerose parti, tra cui le seguenti:

- Modulo per la gestione dell'autenticazione (amministrazione, docenti, studenti, genitori, ...)
- Modulo per la gestione del personale
- Modulo per la gestione degli studenti
- Modulo per la gestione della classe
- Modulo per la gestione del programma scolastico
- Modulo per la gestione delle presenze
- Modulo per la gestione dei compiti per casa
- Modulo per la gestione delle competizioni
- Modulo per la gestione degli esami (sequenziali e ufficiali)
- Modulo per la gestione delle pagelle
- Modulo per la gestione del pagamento delle tasse scolastiche

L'obiettivo della tesi consiste proprio nella progettazione di una base di dati per la gestione delle tasse scolastiche e dei pagamenti in una scuola superiore all'interno di *E-learning*, utilizzando le tecnologie utili per la sua integrazione nella porzione di applicazione già esistente.

In particolare, ogni studente è soggetto alle tre seguenti tasse scolastiche:

- Tassa di iscrizione: è esigibile all'atto di iscrizione ad un corso di studi secondari e la sua validità è pari all'intera durata del ciclo (è unica, perciò viene pagata una volta soltanto). Non è suddivisibile in rate e il suo importo viene definito dalla scuola; d'altra parte, la modalità di pagamento (presso la segreteria della scuola, tramite borsa di studio, tramite credito telefonico, presso la banca indicata dalla scuola oppure online) viene scelta dallo studente. La tassa deve essere pagata interamente anche nel caso in cui l'alunno si ritiri dalla scuola dopo poco; sarà costretto, inoltre, a pagare un contributo aggiuntivo nell'eventualità in cui non paghi la tassa entro la scadenza specificata dalla scuola.
- Tassa di frequenza: deve essere corrisposta annualmente ed è suddivisibile in rate, con il pagamento della prima rata all'inizio dell'anno scolastico e delle altre in base a quanto stabilito dalla scuola (ad esempio una rata per ogni trimestre). La tassa, il cui importo è definito dalla scuola, deve essere pagata interamente sia nel caso in cui l'alunno si ritiri dalla scuola sia nel caso in cui sia costretto ad interrompere la frequenza per un qualche motivo. La modalità di pagamento, invece, viene scelta dallo studente. In caso di trasferimento in un'altra scuola, lo studente pagherà le rate in base a quanto definito dalla nuova scuola. Lo studente sarà costretto a pagare un contributo aggiuntivo nel caso in cui non paghi la tassa entro la scadenza (o le scadenze, se si sceglie di suddividere la tassa in più rate) specificata dalla scuola.

- Tassa d'esame: deve essere corrisposta esclusivamente nel momento in cui viene presentata la domanda per gli esami ufficiali (ad esempio il BEPC, il *Probatoire* o la maturità) e non è suddivisibile in rate. L'importo è definito dallo Stato, mentre la modalità di pagamento è scelta, come nei due casi precedenti, dallo studente. La tassa deve essere pagata interamente anche se lo studente, per un qualche imprevisto, non è in grado di sostenere l'esame; sarà costretto, inoltre, a pagare un contributo aggiuntivo nel caso in cui non paghi la tassa entro la scadenza specificata dallo Stato.

Più che l'aspetto funzionale del progetto, risulta d'interesse e qualificante per la tesi l'integrazione e l'orchestrazione delle tecnologie (fornite da AWS, *Amazon Web Services*) per ottenere un prodotto *deployable* e funzionante. Quindi, lo sviluppo di un progetto completo di:

- Individuazione dei casi d'uso e degli attori
- Diagrammi dei casi d'uso
- Descrizione dei casi d'uso
- Diagramma Entità-Relazione
- Dizionario delle entità
- Dizionario delle relazioni
- Dizionario dei vincoli
- Schema relazionale
- Creazione delle *query* per le operazioni principali

Ad ogni modo, le funzionalità minime richieste nel complesso sono le seguenti:

- Produzione di un modello relazionale normalizzato utilizzabile
- Creazione delle tabelle e capacità di effettuare delle interrogazioni tramite *query*

2. Tecnologie utilizzate

La piattaforma si avvale di un'unica base di dati PostgreSQL.

PostgreSQL è un moderno DBMS (*Data Base Management System*, “sistema di gestione di basi di dati”), ovvero un sistema software che favorisce la definizione, la costruzione, la manipolazione e la condivisione di una base di dati, oltre alla sua protezione e manutenzione. Pubblicato nel 1996, PostgreSQL è un DBMS relazionale, vale a dire basato sul modello relazionale, e open source, facilmente gestibile attraverso uno strumento come pgAdmin 4.

Anche pgAdmin 4, per l'appunto un'applicazione per l'amministrazione di una base di dati PostgreSQL, è stato di fondamentale importanza nel frequente e progressivo monitoraggio dello stato della base di dati nella fase successiva all'installazione del progetto, effettuata al termine della progettazione del nuovo modulo. Questo programma, nel contesto di questa attività, è servito esclusivamente per la lettura delle tabelle e, raramente, per l'eliminazione e l'aggiornamento di record specifici.

Complessivamente, nell'applicazione in questione esistono un solo Server back-end, una sola piattaforma per l'amministrazione del sistema e numerosi applicativi per la presentazione delle scuole, con l'ambiente di test su Heroku, una *platform as a service* sul cloud, e AWS (*Amazon Web Services*) che viene, invece, impiegato per la produzione.

In particolare, lo sviluppo back-end trova il suo fondamento in Rails, un framework per lo sviluppo di applicazioni web scritto in Ruby, un linguaggio di programmazione interpretato ad oggetti. Precisamente, Rails segue uno specifico modello di progettazione per convenzione, denominato MVC (*Model-view-controller*), il quale si basa sulla separazione e distribuzione delle responsabilità di un'applicazione alle tre parti che la costituiscono: *Model* per la rappresentazione dei dati, *View* per la loro visualizzazione e *Controller* per la manipolazione di questi. È interessante, in aggiunta, riportare i due principi alla base di Rails:

- “*Don't Repeat Yourself*” (Non ripeterti): evitando di scrivere le stesse informazioni più volte, il codice risultante è gestibile, estensibile e, possibilmente, privo di errori.
- “*Convention Over Configuration*” (Convenzione al di sopra della configurazione): Rails preferisce adottare un determinato insieme di convenzioni, invece di richiedere la specifica di numerosi file di configurazione.

Diversamente, il front-end dell'applicazione, il quale fa uso di HTML, CSS e JavaScript, comunica tramite API (*Application programming interface*) con il back-end sui framework Vue.js e Quasar framework.

Più nel dettaglio, per la progettazione della base di dati in questione, ossia quella a supporto del pagamento delle tasse scolastiche, è stato utile comporre inizialmente una breve analisi dei requisiti, per meglio rimarcare quali fossero le caratteristiche desiderate. A partire da quest'ultima, poi, sono stati individuati gli attori e i casi d'uso, prima di passare alla creazione vera e propria dei diagrammi dei casi d'uso attraverso draw.io.

draw.io è un software di disegno grafico multiplatforma, utile, ad esempio, per la creazione di diagrammi di flusso, diagrammi UML e diagrammi Entità-Relazione. Si integra con alcuni servizi cloud per l'archiviazione come Dropbox, Google Drive e GitHub ed è anche disponibile come plugin in piattaforme come Notion e Atlassian Confluence, quella utilizzata durante l'attività di tesi. draw.io, il nome attuale di draw.io, offre numerosi strumenti in base al settore di interesse: una volta selezionato quest'ultimo, sono mostrati tutti gli elementi che potrebbero risultare utili allo scopo desiderato (la realizzazione di un diagramma EER in questo caso), come linee, frecce e forme geometriche.

A seguire è stata raggiunta la fase di progettazione concettuale, nella quale è stato prodotto il diagramma EER (*Enhanced Entity-Relationship*), accompagnato dai dizionari delle entità, delle relazioni e dei vincoli. Infine, tramite l'apposito algoritmo di mappatura nella fase di progettazione logica, è stato generato lo schema relazionale finale normalizzato.

La totalità dei passaggi elencati è stata documentata in Confluence, un software per la collaborazione all'interno di un team, grazie al quale è stato possibile tenere traccia dei meetings programmati durante il periodo di lavoro, installare la parte di progetto già esistente, impostare correttamente le variabili d'ambiente, inizializzare le risorse necessarie e ricevere commenti e suggerimenti a proposito del lavoro svolto. In particolare, tutta la documentazione e le varie informazioni sono strutturate in più parti (cartelle), le quali vengono mostrate in un elenco sulla sinistra dell'interfaccia; a loro volta, queste parti ne possono contenere altre, e così via, proprio come in un file system. Il codice vero e proprio, invece, si trova in Bitbucket, sempre di proprietà di Atlassian.

Ad accompagnare le attività successive all'installazione del progetto, chiarita per l'appunto in una parte dedicata in Confluence, è stato l'esame periodico delle cartelle del progetto grazie all'editor Visual Studio Code, precisamente *controllers* e *models* in *app* e *migrate* in *db*, generate da Rails in modo automatico.

Infine, per l'invio delle varie richieste HTTP, tramite cui verificare il corretto funzionamento delle parti implementate, è stato scelto Insomnia. Si tratta di un'applicazione che, senza dover scrivere del codice, ha lo scopo di inviare richieste con diversi metodi (sono stati utilizzati, però, solo GET e POST) ad un'API e ricevere risposte, verificando che non vi siano stati problemi nella fase di elaborazione. In questa applicazione è possibile creare delle cartelle, proprio come in Confluence, per poter meglio organizzare le richieste HTTP effettuate, le quali richiedono di scegliere il metodo, ad esempio POST, specificare il corpo se necessario, ovvero l'insieme dei parametri da inviare al Server, e l'URL della richiesta.

3. Progettazione del modulo di pagamento

Durante la prima fase del lavoro sono stati illustrati alcuni utili dettagli, a livello di progetto per l'appunto, a partire dai quali sarebbe stato opportuno iniziare a proposito del nuovo modulo, sia per garantire coerenza con il progetto esistente sia per comprenderlo più a fondo.

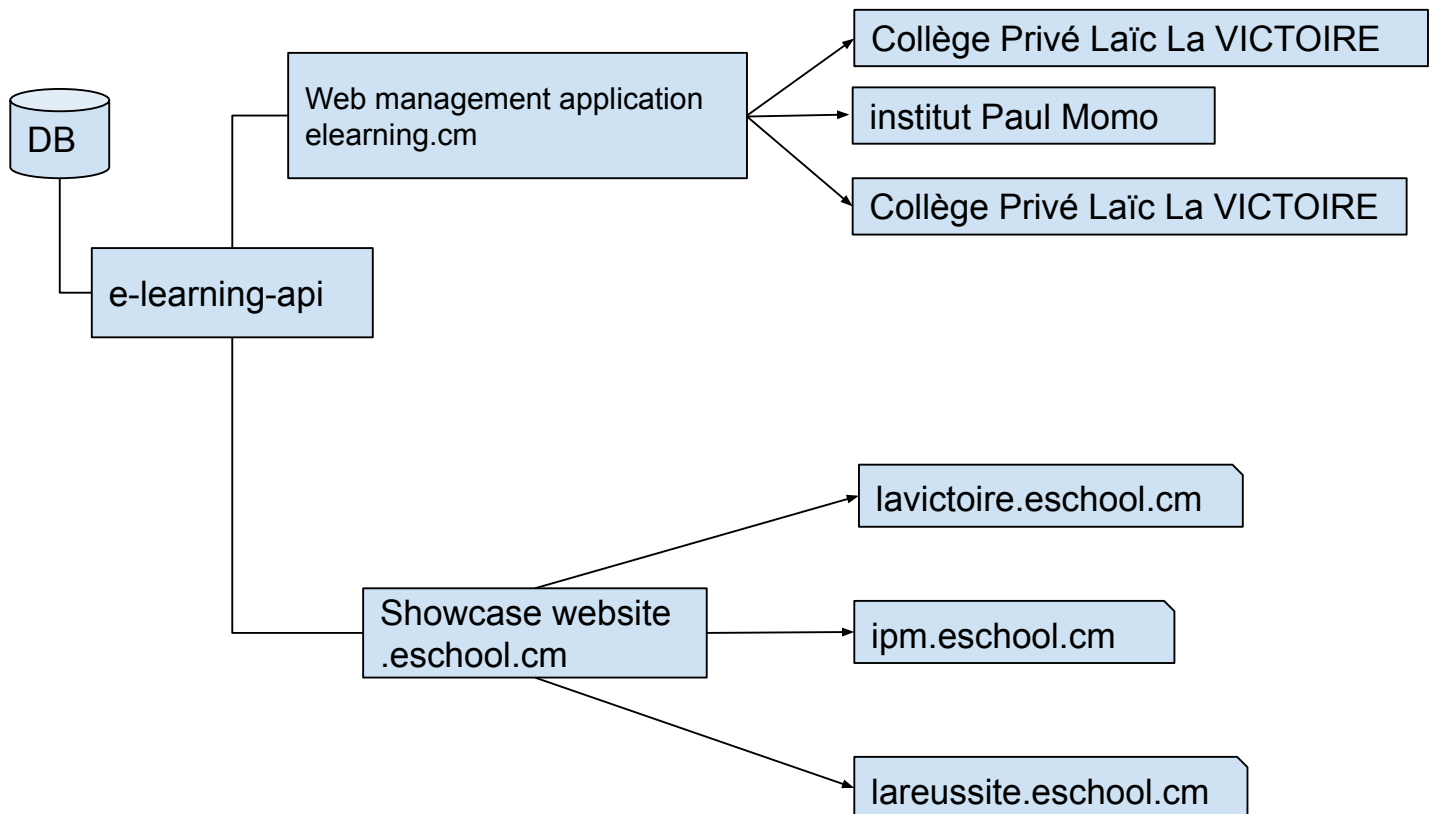


Figura 3.1

Come anticipato precedentemente, la struttura della piattaforma prevede una base di dati unica che comunica con *E-learning-api*, a sua volta in relazione con le altre due parti, *E-school* ed *E-learning*: ciascuna scuola è caratterizzata sia da un sito web sia da un'applicazione di gestione web. Nella Figura 3.1, sulla destra, si possono notare tre esempi.

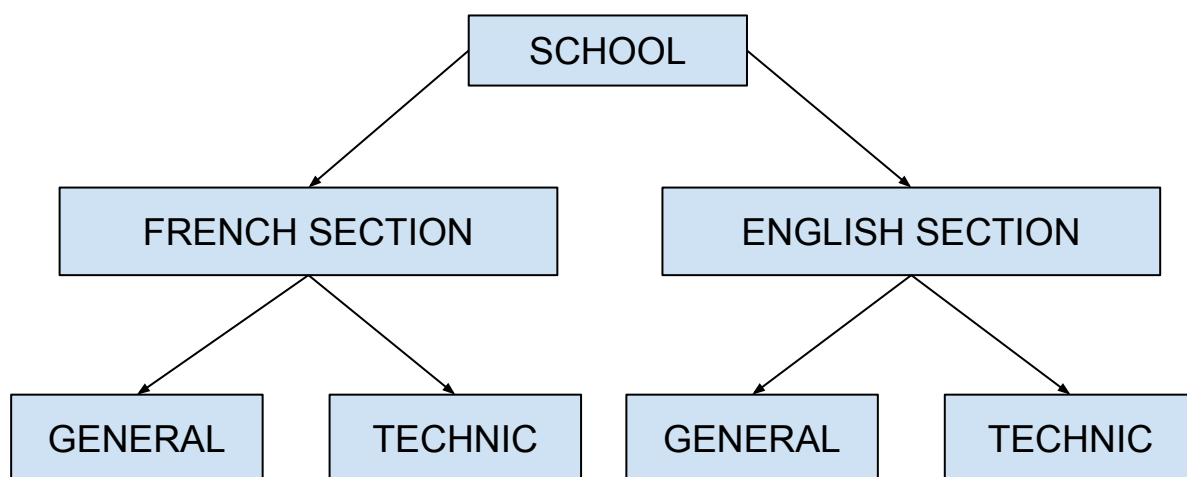


Figura 3.2

Ciascuna scuola, inoltre, prevede una sezione in cui la lingua utilizzata è il francese e un'altra in cui è l'inglese. In entrambi i casi, però, esiste una parte generale e una parte tecnica, come mostra la Figura 3.2.

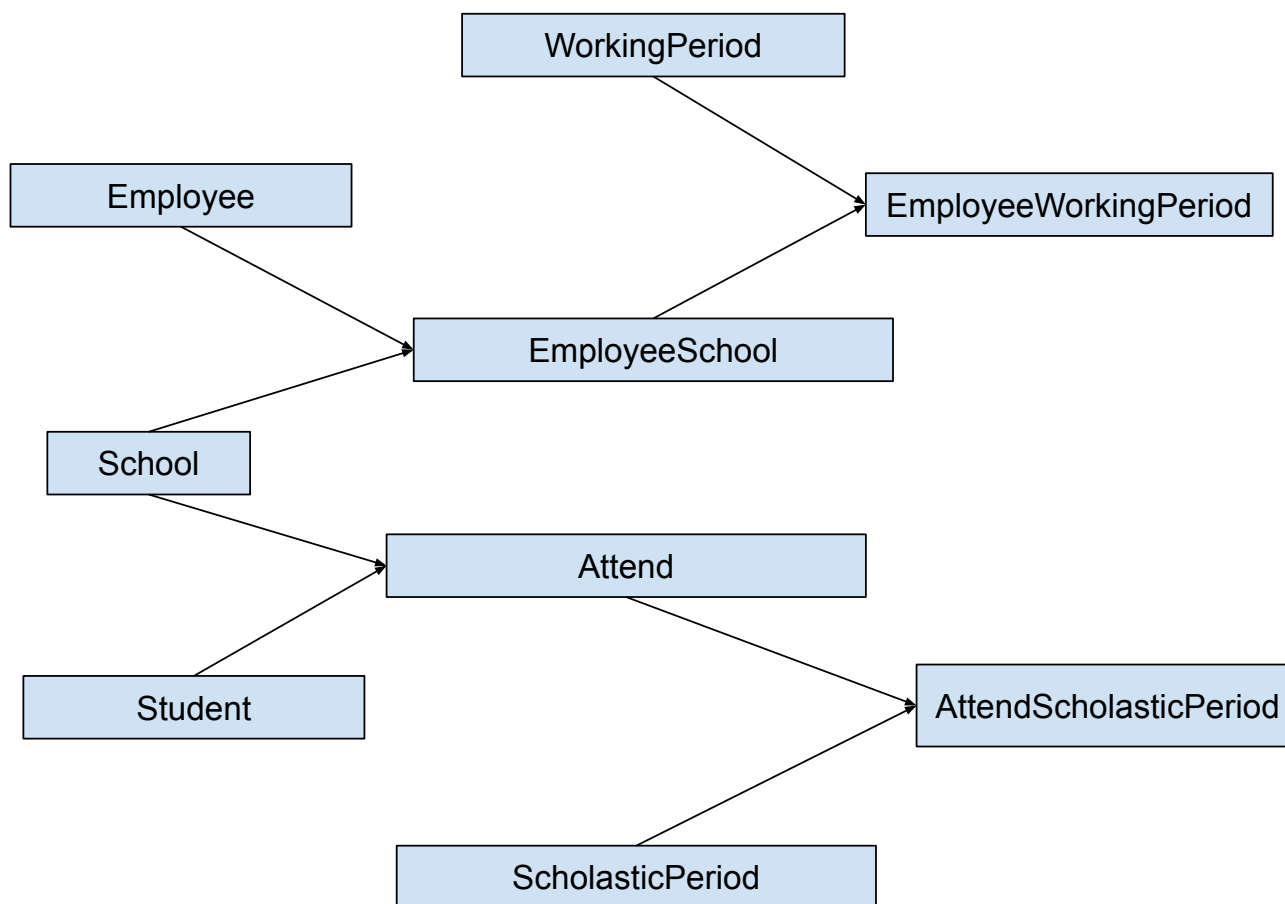



Figura 3.3

Più precisamente, è stato particolarmente d'aiuto il modello della Figura 3.3, in quanto è servito a chiarire meglio i legami tra le principali entità aventi un ruolo nella porzione di applicazione da realizzare.

- *WorkingPeriod* (un periodo lavorativo) ed *EmployeeSchool* (un impiegato della scuola) danno origine, insieme, ad una nuova entità, *EmployeeWorkingPeriod*, ovvero un impiegato della scuola collocato in un preciso periodo lavorativo
- *Employee* (un generico impiegato) e *School* (la scuola in questione) formano, insieme, un *EmployeeSchool*
- *School* e *Student* (uno studente generico) danno origine, invece, ad un *Attend*, uno studente nel contesto di una scuola che effettua lì la sua iscrizione, pagando l'apposita tassa
- *Attend* e *ScholasticPeriod* (un periodo scolastico, ad esempio un anno accademico) formano insieme un *AttendScholasticPeriod*, ovvero uno studente frequentante un preciso periodo scolastico che paga, quindi, la tassa di frequenza e quella per il sostenimento di ciascun esame

A conclusione di tale panoramica preliminare è stata presentata un'ultima immagine con l'obiettivo di definire le informazioni che ciascuno studente dovrebbe poter visualizzare all'interno della sezione "Pagamenti" nella propria area riservata.


LA REUSSITE
 Collège Privé Laïc Bilingue

Services d'enseignement en ligne

+237 000 00 00 00 | info@lareussite.com
 Accueil | **Flora Djimosimo** | Matricule N° 123456 | FR | EN

[Se déconnecter](#)
[Changer le mot de passe](#)
[Autres classes](#)

BIENVENUE DJIMO SIMO MAGGIE FLORA




Classe: T*
 Série: A4 All
 (x 2)

Enseignements

- Emploi de temps
- [Fournitures scolaires](#)
- Programmes et enseignements
- Horaires des cours
- Résultats aux concours
- Inscription aux examens officiels
- Résultats aux examens officiels
- Agenda scolaire
- Epreuves
- Cours de répétition

SÉCRETARIAT

 Horaires
 Frais de scolarité
 Utilisation des locaux
 Inscription en ligne
 Auto-certification
 Carte scolaire
[Administration transparente](#)
 Accès aux actes administratifs
 Tenue
 Boutique

Données personnelles

Cours et exercices

Séquences

Bulletin

Paiements

Documents personnels

Livres (.pdf)

Orientation

Activités parascolaire

Evaluation des activités didactique

Evaluation des infrastructures

NOTIFICATIONS 12

Fist notification
 Sjdetur adipsiong elit...

Second notification
 consectetur arehagr...

Fourth notification
 lyuktrjmk llaeorpty er...

Fifth notification
 viurtyunb h. ugylti mlaeyedg...

Autres...

Agenda scolaire

 ← SEPTEMBRE →

D	L	M	M	J	V	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Figura 3.4

Si può notare che alcune di esse sono il numero della fattura, la descrizione di ciò che si deve pagare, la data di scadenza, l'importo totale e lo stato del pagamento. Non è difficile osservare che i primi campi potrebbero essere caratteristici di una generica tassa, mentre l'ultimo potrebbe interessare, invece, il pagamento effettuato dallo studente.

Superata questa prima visione d'insieme, è stato possibile iniziare con la progettazione vera e propria del nuovo modulo, quello per il pagamento delle tasse scolastiche.

Il primo obiettivo è stato raggiunto con l'individuazione degli attori e dei casi d'uso rispettivamente, oltre alla descrizione di questi ultimi. Di seguito si riporta il contenuto del documento compilato a tal proposito.

- "Attend": uno studente in una scuola, paga la tassa di iscrizione. Include tutte le informazioni di "Student".
- "AttendScholasticPeriod": uno studente in una scuola in uno specifico periodo scolastico (ad esempio un anno scolastico con due semestri), paga la tassa di frequenza (ogni anno, anche suddivisa in più rate) e la tassa d'esame. Include tutte le informazioni di "Attend".
- "Serve": un impiegato della scuola in uno specifico periodo lavorativo. Si occupa dei pagamenti delle tasse presso la segreteria della scuola.
- "EmployeeBank": un impiegato della banca scelta dalla scuola per il pagamento delle tasse.
- "OnlineBank": la banca virtuale.
- "SchoolAdministrator": uno degli amministratori della scuola.
- "State": lo Stato, responsabile di alcune decisioni come il totale da pagare per gli esami ufficiali e la data di scadenza per tali pagamenti.

- **Studente non iscritto**

- "View amount to be paid": uno studente che desidera iscriversi deve poter vedere l'importo della tassa di iscrizione, indipendentemente dal resto
- "View due date": uno studente che desidera iscriversi deve poter vedere la data di scadenza della tassa di iscrizione, indipendentemente dal resto
- "Select payment method": uno studente che desidera iscriversi, dopo aver visionato l'importo da versare e la data di scadenza, può decidere di procedere con il pagamento, selezionando una specifica modalità
- "Payment": lo studente paga secondo il metodo selezionato
- "Pay fine": lo studente è obbligato a pagare un contributo aggiuntivo in caso di pagamento oltre la data di scadenza indicata dalla scuola
- "Proof of payment": lo studente è il destinatario della conferma del pagamento

- **Studente iscritto**

- "View total amount to be paid": uno studente frequentante deve poter vedere l'importo della tassa di frequenza, indipendentemente dal resto
- "View single installment amount": uno studente frequentante deve poter vedere l'importo di una singola tassa, nel caso in cui non sia disposto a pagare la tassa di frequenza in una volta sola

- “View due date”: uno studente frequentante deve poter vedere la data di scadenza della tassa di frequenza, indipendentemente dal resto
 - “Select payment method”: uno studente frequentante, dopo aver visionato l'importo da versare e la data di scadenza, può decidere di procedere con il pagamento, selezionando una specifica modalità
 - “Payment”: lo studente paga secondo il metodo selezionato
 - “Pay fine”: lo studente è obbligato a pagare un contributo aggiuntivo in caso di pagamento oltre la data di scadenza indicata dalla scuola
 - “Proof of payment”: lo studente è il destinatario della conferma del pagamento
-
- Impiegato della segreteria scolastica
 - “Payment”: l'impiegato che lavora presso la segreteria della scuola riceve i soldi dallo studente
 - “Proof of payment”: l'impiegato che lavora presso la segreteria della scuola fornisce allo studente la conferma dell'avvenuto pagamento e ne tiene traccia
-
- Impiegato della banca (scelta dalla scuola)
 - “Payment”: l'impiegato che lavora presso la banca selezionata dalla scuola riceve i soldi dallo studente
 - “Proof of payment”: l'impiegato che lavora presso la banca selezionata dalla scuola fornisce allo studente la conferma dell'avvenuto pagamento e ne tiene traccia
-
- Banca online
 - “Payment”: la banca online (scelta in base alle preferenze del singolo studente) riceve i soldi dallo studente
 - “Proof of payment”: la banca online fornisce allo studente la conferma dell'avvenuto pagamento e ne tiene traccia
-
- Amministratore scolastico
 - “View amount to be paid”: uno (o più) degli amministratori scolastici determina l'importo da versare in merito alla tassa di frequenza
 - “View due date”: uno (o più) degli amministratori scolastici determina la data entro cui dovrà essere versata la tassa di iscrizione o di frequenza (o una delle sue rate)
 - “Pay fine”: uno (o più) degli amministratori scolastici determina l'importo della sanzione che lo studente dovrà pagare in caso di pagamento in ritardo di una qualsiasi tassa
 - “Select payment method”: uno (o più) degli amministratori scolastici determina quali sono le possibili modalità di pagamento

- “View single installment amount”: uno (o più) degli amministratori scolastici determina l'importo da versare relativo ad una singola rata della tassa di frequenza
- “View total amount to be paid”: uno (o più) degli amministratori scolastici determina l'importo totale (non suddiviso in rate) da versare relativamente alla tassa di frequenza

- Stato

- “View amount to be paid”: lo Stato determina l'importo da pagare per poter sostenere un esame ufficiale
- “View due date”: lo Stato determina la data entro cui deve essere versata la tassa d'esame

Il secondo obiettivo è stato raggiunto, invece, tramite la creazione di tre diagrammi dei casi d'uso: il primo, *Attend.drawio* (Figura 3.5), presenta “Attend” come attore principale, mentre il secondo e il terzo, *AttendScholasticPeriod.drawio* (Figura 3.6) e *AttendScholasticPeriodExam.drawio* (Figura 3.7), sono costruiti intorno all’attore “AttendScholasticPeriod”.

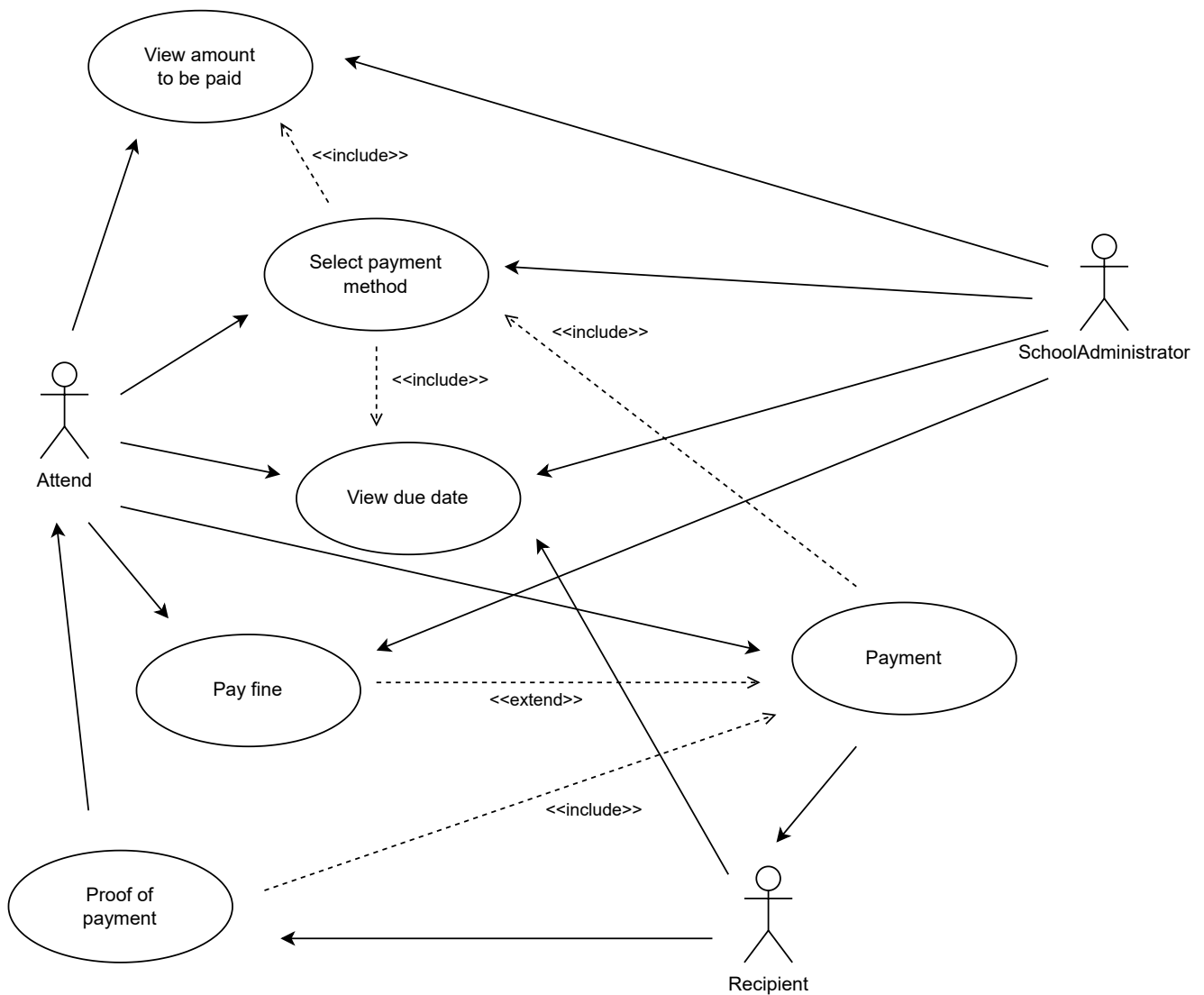


Figura 3.5 Attend.drawio

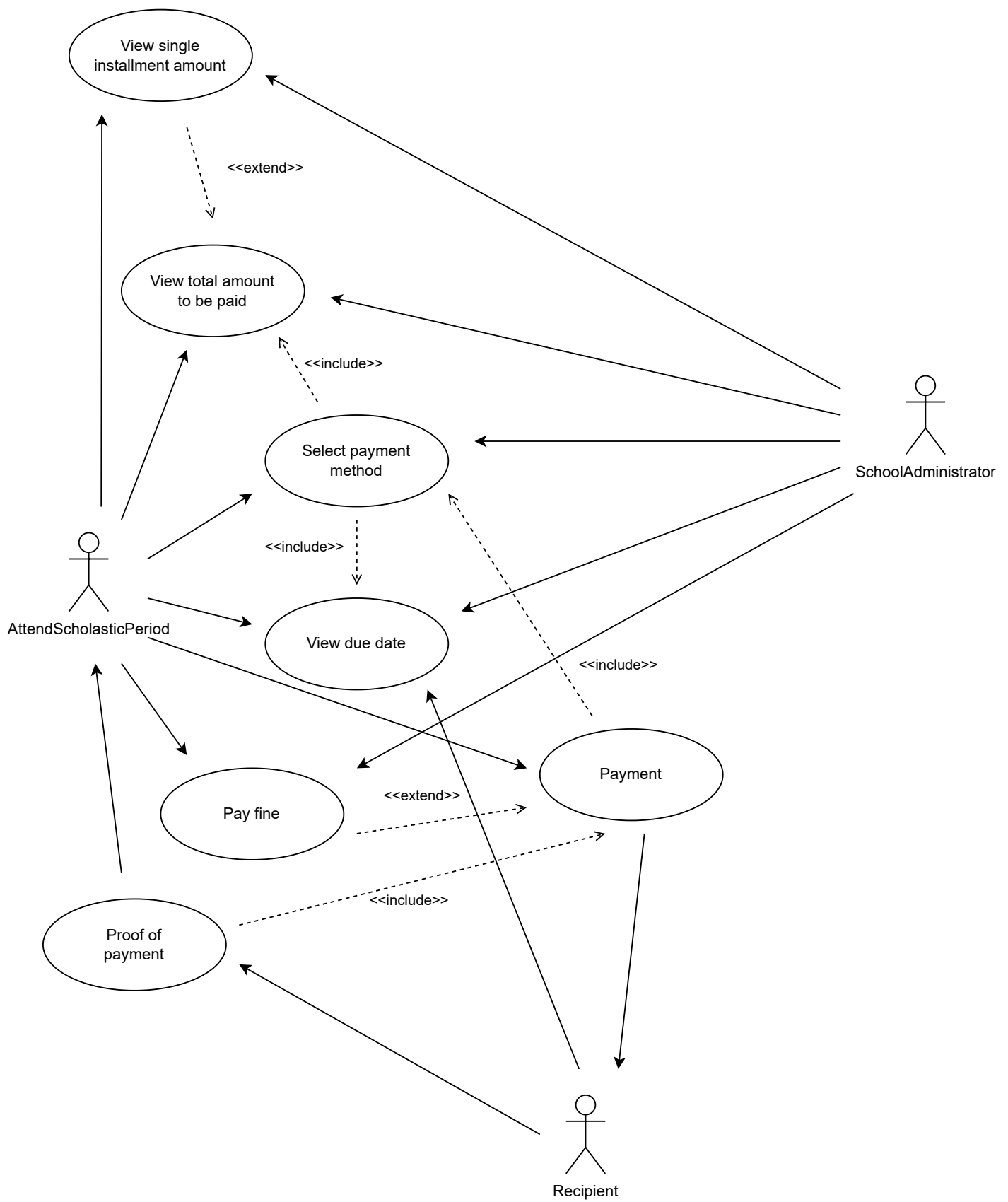


Figura 3.6 AttendScholasticPeriod.drawio

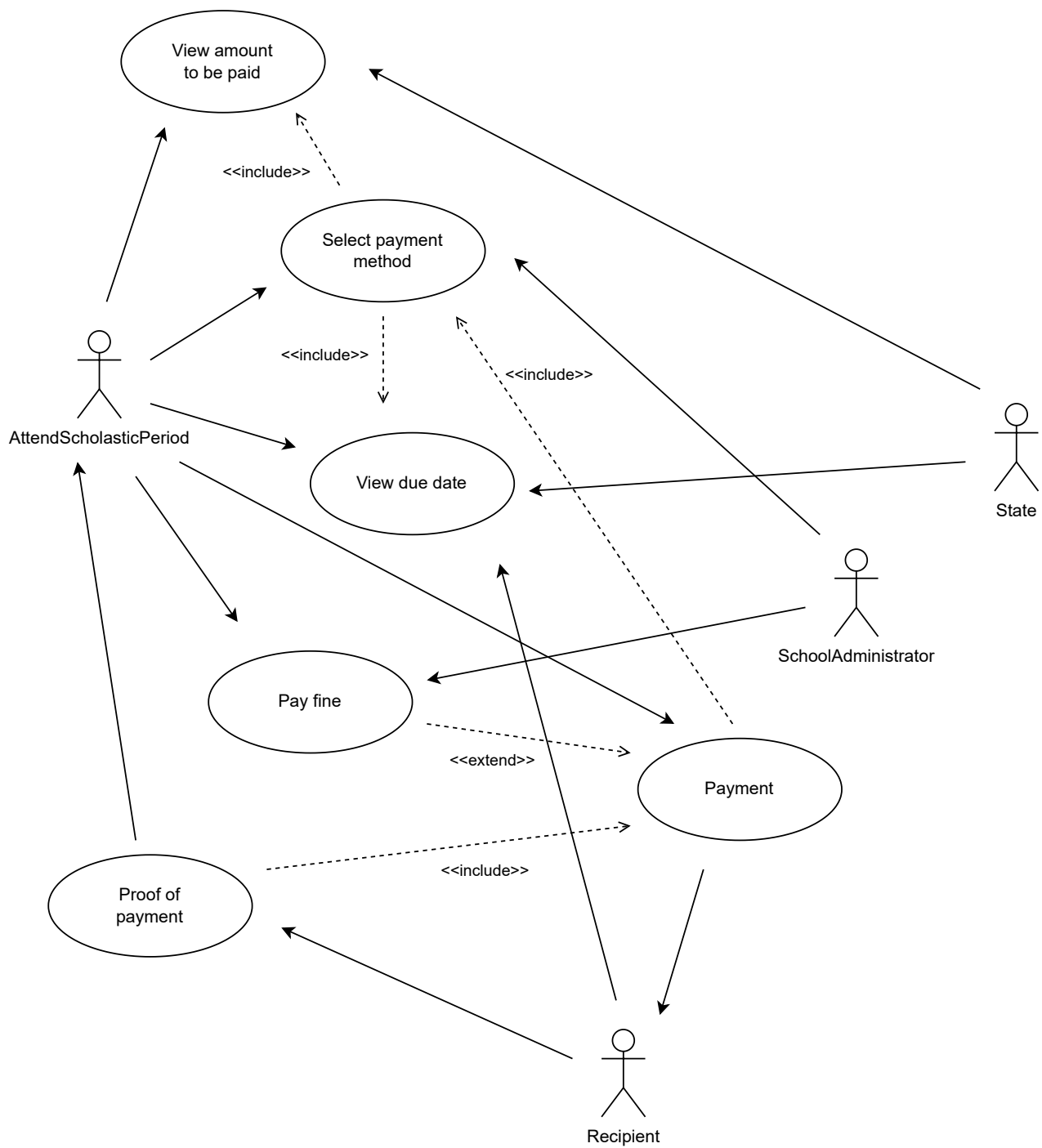


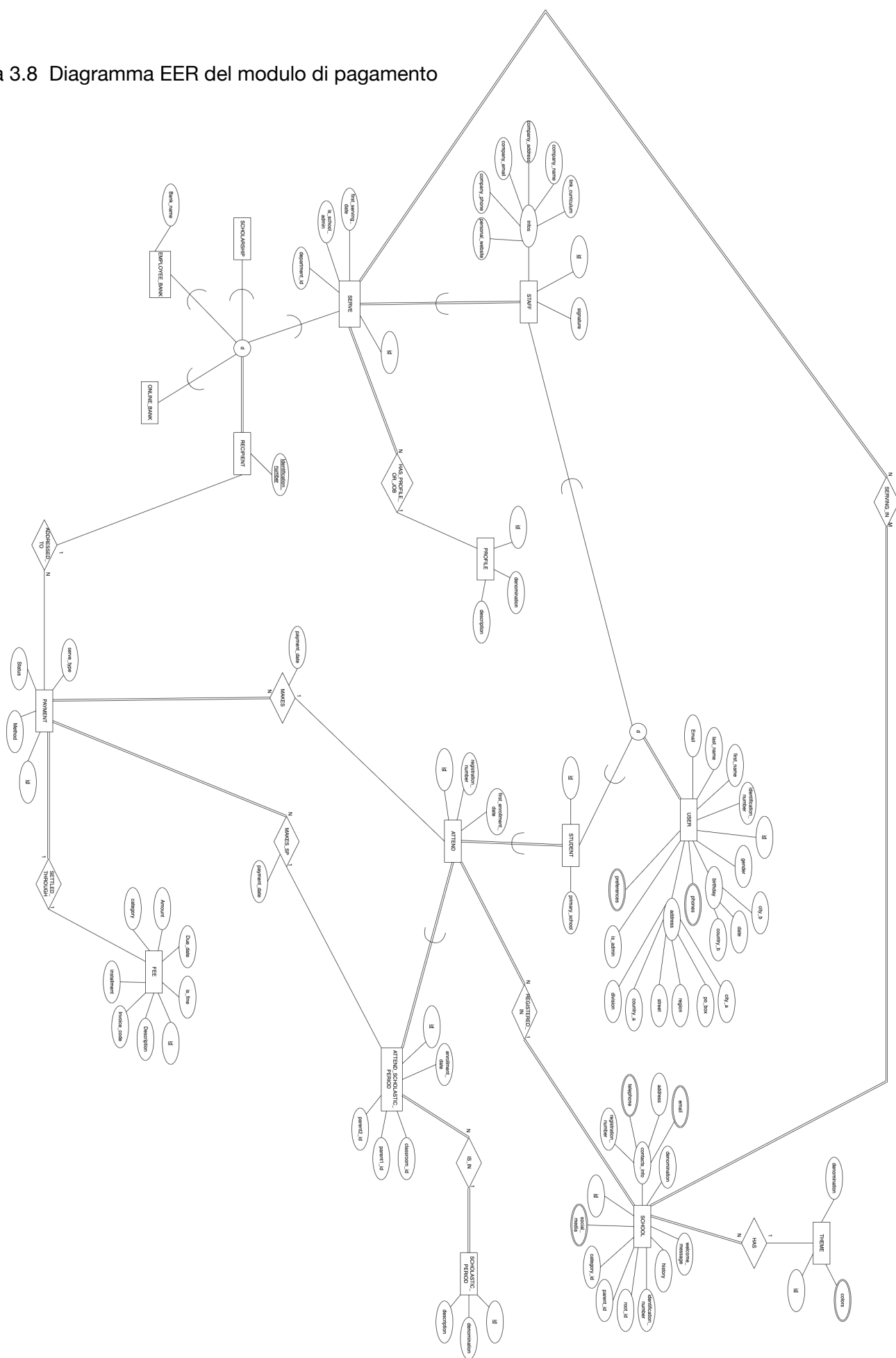
Figura 3.7 AttendScholasticPeriodExam.drawio

Relativamente ai diagrammi dei casi d'uso sono state aggiunte le note esplicative che seguono:

- Nei tre diagrammi l'attore "Recipient" rappresenta "Serve", "EmployeeBank" o "OnlineBank", in base al metodo di pagamento selezionato dallo studente
- L'attore "Recipient" deve poter visualizzare la data di scadenza di una tassa prima di occuparsi di un pagamento

Il terzo obiettivo di questa fase è stato realizzare un diagramma entità relazione, arricchito dai dizionari delle entità, delle relazioni e dei vincoli. Nel fare ciò è stato tenuto in considerazione quello che era lo stato attuale della base di dati. In particolare, sono state definite (e collegate alle entità già esistenti in precedenza) FEE, PAYMENT e RECIPIENT, specializzata in SERVE, SCHOLARSHIP, EMPLOYEE_BANK e ONLINE_BANK.

Figura 3.8 Diagramma EER del modulo di pagamento



Si riporta successivamente il contenuto dei tre dizionari appena nominati, a supporto di quanto è rappresentato in Figura 3.8.

- Entità

- USER: utente generico specializzato in STAFF e STUDENT; ha i seguenti attributi: id (la chiave primaria), identification_number (una chiave candidata), first_name, last_name, gender, birthday (city_b, date, country_b), phones, Email, address (city_a, po_box, region, street, country_a, division), is_admin, preferences
- STUDENT: un utente specifico che rappresenta uno studente; a parte quelli di USER, ha i seguenti attributi: id (la chiave primaria), primary_school
- ATTEND: uno studente che vuole iscriversi ad una scuola; a parte quelli di STUDENT, ha i seguenti attributi: id (la chiave primaria), registration_number, first_enrollment_date
- SCHOOL: la scuola nel suo complesso; ha i seguenti attributi: id (la chiave primaria), denomination, welcome_message, history, contacts_info (email, address, telephone, registration_number), social_media, identification_number, root_id, parent_id, category_id
- THEME: il tema (corrente) scelto da una scuola; ha i seguenti attributi: id (la chiave primaria), denomination, colors
- ATTEND_SCHOLASTIC_PERIOD: uno studente frequentante; a parte quelli di ATTEND, ha i seguenti attributi: id (la chiave primaria), enrollment_date, classroom_id, parent1_id, parent2_id
- SCHOLASTIC_PERIOD: un periodo scolastico specifico; ha i seguenti attributi: id (la chiave primaria), denomination, description
- FEE: una sanzione o una tassa scolastica; ha i seguenti attributi: id (la chiave primaria), Description, Invoice_code, Due_date, Amount, is_fine, category, installment
- PAYMENT: un pagamento attraverso cui una tassa è saldata; ha i seguenti attributi: id (la chiave primaria), Method, Status, serve_type
- STAFF: un utente specifico che ha una determinata posizione lavorativa (non necessariamente a scuola); a parte quelli di USER, ha i seguenti attributi: id (la chiave primaria), infos (link_curriculum, company_name, company_address, company_email, company_phone, personal_website), signature
- RECIPIENT: il destinatario del pagamento di una tassa, specializzato in: SCHOLARSHIP, SERVE, EMPLOYEE_BANK (con l'attributo Bank_name), ONLINE_BANK; ha i seguenti attributi: Identification_number (la chiave primaria); in particolare, l'entità SERVE, che è anche una specializzazione di STAFF, a parte quelli di RECIPIENT e STAFF, ha i seguenti attributi: id (la chiave primaria), first_serving_date, is_school_admin, department_id
- PROFILE: un impiego; ha i seguenti attributi: id (la chiave primaria), denomination, description

- Relazioni

- MAKES: relazione 1:N tra ATTEND e PAYMENT; la partecipazione è parziale per la prima entità e totale per la seconda; ha l'attributo payment_date
- REGISTERED_IN: relazione 1:N tra SCHOOL e ATTEND; la partecipazione è totale per entrambe le entità

- HAS: relazione 1:N tra THEME e SCHOOL; la partecipazione è parziale per la prima entità e totale per la seconda
- SERVING_IN: relazione M:N tra SCHOOL e SERVE; la partecipazione è totale per entrambe le entità
- IS_IN: relazione 1:N tra SCHOLASTIC_PERIOD e ATTEND_SCHOLASTIC_PERIOD; la partecipazione è totale per entrambe le entità
- MAKES_SP: relazione 1:N tra ATTEND_SCHOLASTIC_PERIOD e PAYMENT; la partecipazione è parziale per la prima entità e totale per la seconda; ha l'attributo payment_date
- SETTLED_THROUGH: relazione 1:1 tra FEE e PAYMENT; la partecipazione è parziale per la prima entità e totale per la seconda
- ADDRESSED_TO: relazione 1:N tra RECIPIENT e PAYMENT; la partecipazione è parziale per entrambe le entità
- HAS_PROFILE_OR_JOB: relazione 1:N tra PROFILE e SERVE; la partecipazione è parziale per la prima entità e totale per la seconda

- Vincoli

- Uno studente che vuole iscriversi ad una scuola deve pagare una tassa di iscrizione che non può essere suddivisa in rate
- Uno studente che vuole iscriversi ad una scuola deve pagare, oltre alla tassa di iscrizione, un contributo aggiuntivo se non rispetta la scadenza
- Una volta pagata la quota di iscrizione, lo studente diventa frequentante
- Ogni studente frequentante è tenuto a versare una tassa di frequenza ogni anno, all'inizio o tramite un certo numero di rate nel corso dell'anno
- Ogni studente frequentante dovrà pagare, oltre alla tassa di frequenza, un contributo aggiuntivo nel caso in cui non rispetti la scadenza (o una delle scadenze)
- Ogni studente frequentante che arriva a sostenere almeno uno degli esami ufficiali (non tutti gli studenti frequentanti arriveranno a questo punto) dovrà pagare una tassa prima di sostenere l'esame in questione
- Ogni studente frequentante che arrivi a sostenere almeno uno degli esami ufficiali dovrà pagare, oltre alla tassa d'esame, un contributo aggiuntivo se non rispetta la scadenza
- Ogni tassa, indipendentemente dalla tipologia, viene corrisposta tramite un pagamento
- Ogni pagamento, se andato a buon fine (potrebbero esserci dei pagamenti che non vanno a buon fine), ha un "destinatario": la borsa di studio (se presente), l'impiegato della banca a cui lo studente ha effettuato il pagamento, l'impiegato della scuola che può occuparsi dei pagamenti o la banca virtuale (la banca online)

Infine, a conclusione di questa sezione, si riporta lo schema relazionale ottenuto dalla traduzione del diagramma EER seguendo l'algoritmo di mappatura in otto passi che segue:

1. Traduzione di tipi di entità forti
2. Traduzione di tipi di entità deboli
3. Traduzione di tipi di associazioni binarie 1:1
4. Traduzione di tipi di associazioni binarie 1:N
5. Traduzione di tipi di associazioni binarie M:N
6. Traduzione di attributi multivalore
7. Traduzione di tipi di associazioni N-arie
8. Traduzione di specializzazioni o generalizzazioni

USER

<u>id</u>	iden tifica tion_ num ber	first_ nam e	last_ nam e	gen der	city_ b	date	cou ntry _b	Ema il	city_ a	po_ box	regi on	stre et	cou ntry _a	divis ion	is_a dmi n
-----------	---------------------------------------	--------------------	-------------------	------------	------------	------	-------------------	-----------	------------	------------	------------	------------	-------------------	--------------	------------------

THEME

<u>id</u>	denomination
-----------	--------------

SCHOOL

<u>id</u>	denomi nation	welcom e_mess age	history	addres s	registra tion_nu mber	identificat ion_n umber	root_id	parent_ id	categor y_id	theme_i d (FK)
-----------	------------------	-------------------------	---------	-------------	-----------------------------	-------------------------------	---------	---------------	-----------------	-------------------

PAYMENT

<u>id</u>	Method	Status	serve_typ e	payment_ date	recipient_ identificat ion_numb er (FK)	attend_id (FK)	attend_sc holastic_ period_id (FK)	fee_id (FK)
-----------	--------	--------	----------------	------------------	--	-------------------	---	----------------

PROFILE

<u>id</u>	denomination	description
-----------	--------------	-------------

SCHOLASTIC_PERIOD

<u>id</u>	denomination	description
-----------	--------------	-------------

RECIPIENT

<u>Identification_number</u>

FEE

<u>id</u>	Description	Invoice_code	Due_date	Amount	is_fine	category	installment
-----------	-------------	--------------	----------	--------	---------	----------	-------------

ATTEND

<u>id (*)</u>	registration_number	first_enrollment_date	school_id (FK)
---------------	---------------------	-----------------------	----------------

ATTEND_SCHOLASTIC_PERIOD

<u>id (*)</u>	enrollment_date	classroom_id	parent1_id	parent2_id	scholastic_period_id (FK)
---------------	-----------------	--------------	------------	------------	---------------------------

SERVE

<u>id (*)</u>	<u>Identification_number (*)</u>	first_serving_date	is_school_admin	profile_id (FK)	department_id
---------------	----------------------------------	--------------------	-----------------	-----------------	---------------

SERVING_IN

<u>serve_id (FK)</u>	<u>school_id (FK)</u>
----------------------	-----------------------

PHONES_USER

<u>phones</u>	<u>user_id</u> (FK)
---------------	---------------------

PREFERENCES_USER

<u>preferences</u>	<u>user_id</u> (FK)
--------------------	---------------------

COLORS_THEME

<u>colors</u>	<u>theme_id</u> (FK)
---------------	----------------------

SOCIAL_MEDIA_SCHOOL

<u>social_media</u>	<u>school_id</u> (FK)
---------------------	-----------------------

EMAIL_SCHOOL

<u>email</u>	<u>school_id</u> (FK)
--------------	-----------------------

TELEPHONE_SCHOOL

<u>telephone</u>	<u>school_id</u> (FK)
------------------	-----------------------

STAFF

<u>id</u> (*)	link_curriculum	company_name	company_address	company_email	company_phone	personal_website	signature
---------------	-----------------	--------------	-----------------	---------------	---------------	------------------	-----------

STUDENT

<u>id</u> (*)	primary_school
---------------	----------------

SCHOLARSHIP

<u>Identification_number</u> (*)

EMPLOYEE_BANK

<u>Identification_number</u> (*)

Bank_name

ONLINE_BANK

<u>Identification_number</u> (*)

In corrispondenza di alcuni attributi è presente, tra parentesi tonde, la sigla FK (*Foreign Key*): ciò significa che tale attributo è una chiave esterna; vicino ad altri, invece, il simbolo “*”, sempre tra parentesi tonde, indica che l’attributo in questione punta alla chiave primaria della superclasse.

4. Implementazione del modulo di pagamento

L'ultima parte della creazione del nuovo modulo è iniziata con il set-up del progetto tramite i seguenti step:

1. Creare un account su Bitbucket, un servizio di hosting web-based
2. Generare e aggiungere una chiave SSH nelle impostazioni di Bitbucket, per evitare le richieste di password quando viene eseguito il push del codice in Bitbucket
3. Clonare il progetto all'interno del proprio computer
A. `git clone git@bitbucket.org:SYS_AIT/elearning-api.git`
4. Entrare nel repository
A. `cd elearning-api`
5. Spostarsi nel proprio *branch* (ad esempio con il comando `git checkout feature/fee-payment`)
6. Creare un file chiamato ".env" nella root directory del progetto e inserire al suo interno le variabili d'ambiente fornite
7. Creare la base di dati
A. `rails db:create`
8. Inizializzare la base di dati e creare le tabelle
A. `rails db:migrate`
B. `rails db:migrate RAILS_ENV=test`
9. Installare le dipendenze
A. `bundle install`
10. Eseguire il progetto localmente
A. `rails server -p 3002`

Una volta completata la configurazione del progetto sono state create alcune risorse iniziali attraverso i passaggi che seguono. È importante sottolineare che i comandi riportati devono essere lanciati da riga di comando e, inoltre, è opportuno assicurarsi che questi, una volta eseguiti, abbiano comportato l'effettiva creazione di un record nella base di dati. Questo si può verificare tramite pgAdmin 4.

1. Creare un utente amministratore, considerato l'autore delle risorse successive
 - A. Aprire il file `elearning-api/lib/tasks/create_user.rake`
 - B. Modificare il contenuto del file con i propri dati personali
 - C. Lanciare il comando `rake create_user`
2. Creare un tema, ovvero una combinazione di colori
 - A. Aprire il file `elearning-api/lib/tasks/create_theme.rake` e modificare la denominazione e i colori di tale tema
 - B. Lanciare il comando `rake create_theme`
3. Creare la scuola radice, vale a dire la scuola con numero identificativo uguale a 00000000000
 - A. Nel file `.env` porre la variabile `ADMIN_USER` uguale al valore dell'id dell'utente amministratore (ad esempio quello definito in precedenza) che andrà a creare la scuola radice
 - B. Lanciare il comando `rake create_root_school`
4. Creare una scuola
 - A. Nel file `.env` porre la variabile `ADMIN_USER` uguale al valore dell'id dell'utente amministratore (ad esempio quello definito in precedenza) che andrà a popolare la scuola
 - B. Nel file `.env` impostare opportunamente la variabile `NUMBER_OF_RECORDS` in base al numero di scuole che si desidera creare
 - C. Aprire il file `elearning-api/lib/tasks/rake/populate_schools.rake` e modificarne il contenuto con le informazioni di una nuova scuola
 - D. Lanciare il comando `rake populate_schools`
 - E. È da notare che, per ciascuna scuola, `root_id` e `parent_id` sono posti uguali all'id della scuola radice, creata in un punto precedente; inoltre, `theme_id` è posto uguale all'id del primo tema aggiunto alla base di dati, vale a dire quello creato inizialmente
5. Creare un periodo scolastico
 - A. Aprire `elearning-api/lib/tasks/rake/populate_scholastic_periods.rake` e modificarlo con le informazioni riguardanti un nuovo periodo scolastico
 - B. Nel file `.env` impostare opportunamente le variabili `SCHOOL_ID`, ad esempio uguale a 2, `ADMIN_USER`, ad esempio uguale a 1 e `NUMBER_OF_RECORDS`
 - C. Lanciare il comando `rake populate_scholastic_periods`
6. Creare uno studente, ovvero uno "user", un "attend" e un "attend_scholastic_period"
 - A. Aprire il file `elearning-api/lib/tasks/populate_students.rake` e modificarne il contenuto con le informazioni di un nuovo studente. È importante compilare correttamente `sign_up_params` e `student_params`
 - B. Nel file `.env` impostare opportunamente le variabili `SCHOOL_ID`, ad esempio uguale a 2, `SCHOLASTIC_PERIOD_ID`, ad esempio uguale a 1, `ADMIN_USER`, ad esempio uguale a 1 e `NUMBER_OF_RECORDS`, uguale a 1 per creare un solo studente
 - C. Lanciare il comando `rake populate_students`

In seguito a questa prima fase preparatoria sono stati lanciati i comandi per generare i componenti necessari all'introduzione del nuovo modulo, grazie ad una particolare modalità di Rails chiamata *scaffolding*. In generale, uno *scaffold* comprende un modello, una relativa migrazione della base di dati, un controllore, le viste e un insieme di test per ciascuno di questi elementi. Sempre all'interno della directory `elearning-api`, la cartella in cui si trova il progetto completo, sono stati eseguiti i seguenti comandi:

```
bin/rails g scaffold Fee Description:text Invoice_code:integer
Due_date:date Amount:float is_fine:boolean category:string
installment:integer
```

```
bin/rails g scaffold Recipient Identification_number:string
```

```
bin/rails g scaffold Payment Method:text Status:string serve_type:text
payment_date:date recipient:references attend:references
attend_scholastic_period:references fee:references
```

```
bin/rails g scaffold Scholarship recipient:references
```

```
bin/rails g scaffold EmployeeBank Bank_name:string recipient:references
```

```
bin/rails g scaffold OnlineBank recipient:references
```

In un secondo momento, tramite una migrazione definita appositamente, è stata aggiunta una relazione di tipo M:N tra *Serve* e *Recipient*, supportata da un opportuno endpoint nel controllore di *Serve*, con lo scopo di creare una relazione tra i due oggetti e di impostare `is_school_admin` uguale a `true`: solo chi possiede il titolo di amministratore, infatti, può occuparsi dei pagamenti delle tasse.

```
def set_school_admin
  serve = Serve.find(params[:id])
  recipient = Recipient.find(params[:recipient_id])
  serve.update(is_school_admin: true)
  serve.recipients << recipient
  if serve.save
    render json: { message: "Relation between Serve and Recipient has
been successfully created" }
  else
    render json: { error: "Failed to create relation between Serve and
Recipient" }, status: :unprocessable_entity
  end
end
```

Poi, attraverso il comando `make migrations` sono state eseguite le sette migrazioni generate per la creazione delle nuove tabelle.

Una volta creati gli elementi su cui operare è stato anche aggiunto un metodo al modello `fee.rb` per la generazione automatica della colonna `Invoice_code`, la quale deve contenere stringhe uniche, casuali e composte da sei cifre:

```
before_create :generate_unique_invoice_code

def generate_unique_invoice_code
  loop do
    code = rand(100000..999999)
    unless Fee.exists?(Invoice_code: code)
      self.Invoice_code = code
      break
    end
  end
end
```

Successivamente sono iniziate le simulazioni tramite Insomnia: sempre all'interno della directory `elearning-api` è stato lanciato il comando `make dev` per avviare Puma, un Server web distribuito con Rails in ascolto sulla porta 3002 (dovrebbe essere, in realtà, la porta 3000 ma per questo progetto è stata scelta una porta differente), mentre in Insomnia, per ciascuna creazione, è stato generato un nuovo messaggio di richiesta HTTP con il metodo POST. Si riportano, in ordine, i passaggi effettuati a tal proposito e alcuni esempi di supporto.

1. Creazione di tasse delle tre tipologie, ovvero di iscrizione, di frequenza e d’esame

Ad esempio, nel caso della tassa di iscrizione:

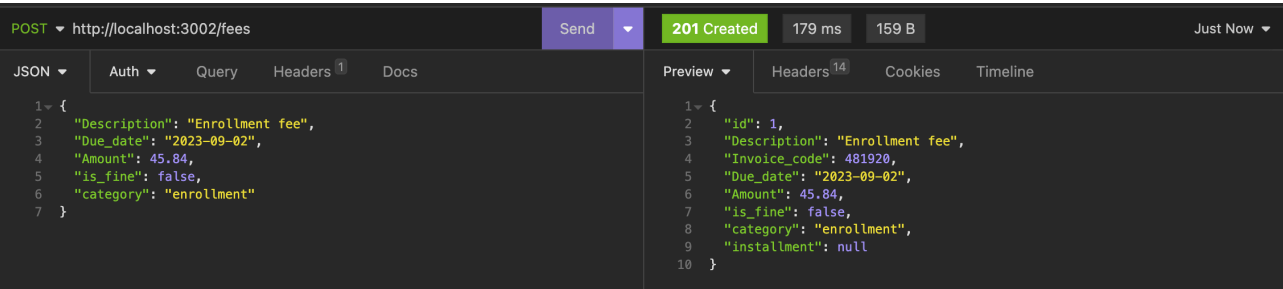


Figura 4.1. Messaggio di richiesta a sinistra e quello di risposta a destra in Insomnia

	id [PK] bigint	Description text	Invoice_code integer	Due_date date	Amount double precision	is_fine boolean	category character varying	installment integer
1	1	Enrollment fee	481920	2023-09-02	45.84	false	enrollment	[null]
2	2	Tuition fee	976978	2024-06-01	289.8	false	tuition	[null]
3	3	Exam fee	826852	2024-05-25	30.51	false	exam	[null]
4	4	Fine for late payment of the enrollment fee	574231	2023-09-07	7.63	true	enrollment	[null]
5	5	Enrollment fee	339498	2023-09-02	45.84	false	enrollment	[null]
6	6	Tuition fee	586971	2024-06-01	289.8	false	tuition	[null]
7	7	Fine for late payment of the tuition fee	395396	2024-06-07	7.63	true	tuition	[null]
8	8	Exam fee	869359	2024-05-25	30.51	false	exam	[null]
9	9	Enrollment fee	774950	2023-09-02	45.84	false	enrollment	[null]
10	10	Tuition fee	881902	2024-06-01	289.8	false	tuition	[null]
11	11	Exam fee	115746	2024-05-25	30.51	false	exam	[null]
12	12	Fine for late payment of the exam fee	348225	2024-05-27	7.63	true	exam	[null]

Figura 4.2. La tabella “fees” in pgAdmin4

2. Creazione delle quattro tipologie di “recipient”, ovvero “employee_bank”, “online_bank”, “scholarship” e “serve”

Questo particolare step ha richiesto l'utilizzo dei cosiddetti *nested attributes* di Rails, in modo da creare automaticamente il “recipient” generale al momento della creazione di una sua specializzazione: un “recipient” particolare è anche un “recipient” generale per definizione. È stato necessario, quindi, aggiungere la riga

```
accepts_nested_attributes_for :recipient
```

in ciascuno dei quattro modelli elencati sopra e la parte

```
recipient_attributes: [:id, :Identification_number]
```

all'interno del metodo `permit`, utilizzato alla fine di ciascun controllore per consentire solo determinati parametri (quelli specificati) di una richiesta HTTP.

Ad esempio, nel caso del primo tipo di “recipient” elencato, ovvero “employee_bank”:

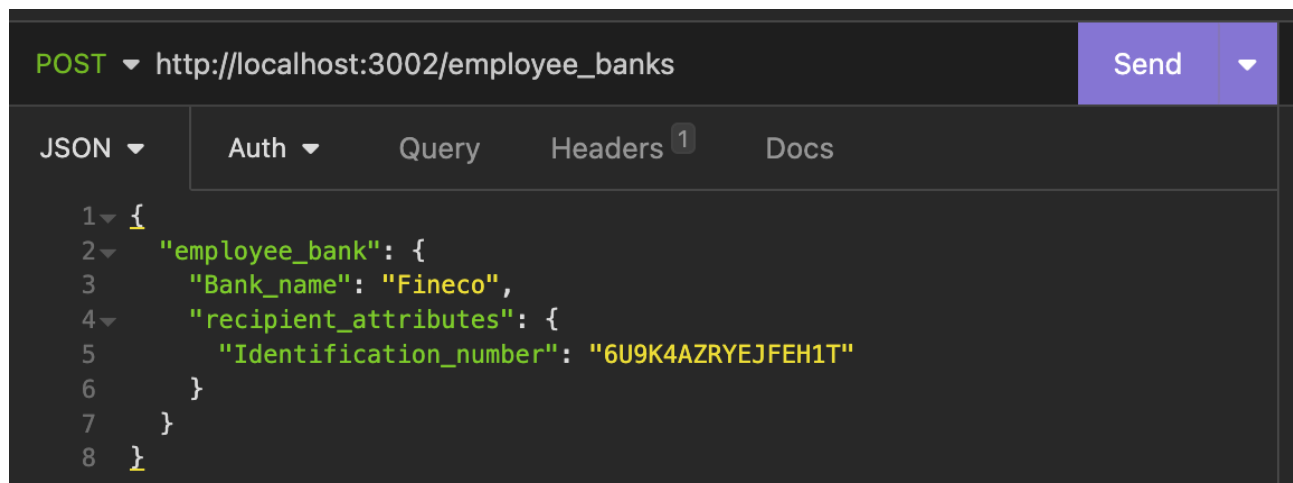


Figura 4.3 Messaggio di richiesta in Insomnia

	id [PK] bigint	Bank_name character varying	recipient_id bigint	created_at timestamp without time zone (6)	updated_at timestamp without time zone (6)
1	3	UniCredit	6	2023-09-05 15:21:02.610493	2023-09-05 15:21:02.610493
2	4	Fineco	7	2023-09-05 15:21:30.950012	2023-09-05 15:21:30.950012

Figura 4.4 La tabella “employee_banks” in pgAdmin 4: si noti la riga 2





	id [PK] bigint 	created_at timestamp without time zone (6) 	updated_at timestamp without time zone (6) 	Identification_number character varying 
1	3	2023-09-05 15:01:05.676375	2023-09-05 15:01:05.676375	4N9K8SPRYEJFTH2T
2	4	2023-09-05 15:14:13.545397	2023-09-05 15:14:13.545397	4N9K8SPRYEEDH2T
3	5	2023-09-05 15:14:53.115215	2023-09-05 15:14:53.115215	4N7R8SPRYEEGAP2T
4	6	2023-09-05 15:21:02.607681	2023-09-05 15:21:02.607681	6F9K8AZRYEJFTH1T
5	7	2023-09-05 15:21:30.948328	2023-09-05 15:21:30.948328	6U9K4AZRYEJFEH1T

Figura 4.5. La tabella “recipients” in pgAdmin 4: si noti la riga 5

3. Creazione di tre studenti

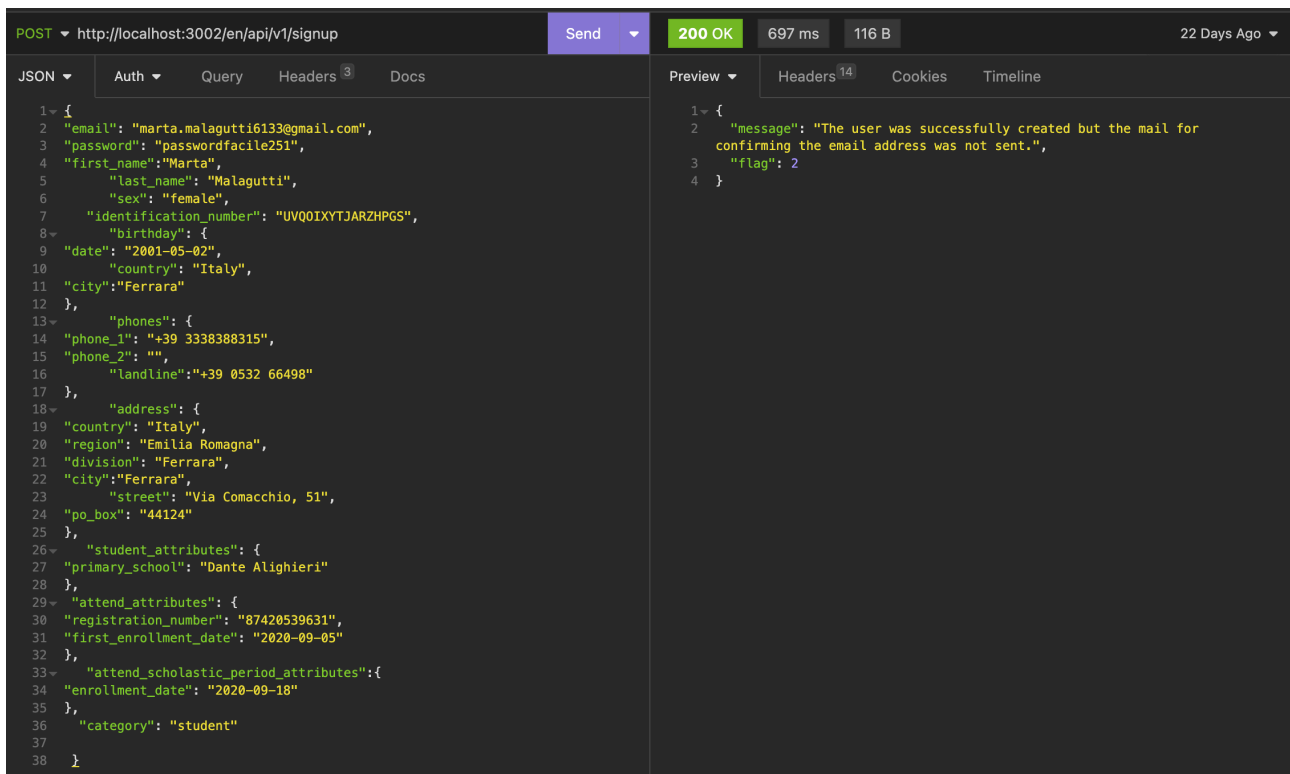


Figura 4.6 Messaggio di richiesta a sinistra e quello di risposta a destra in Insomnia

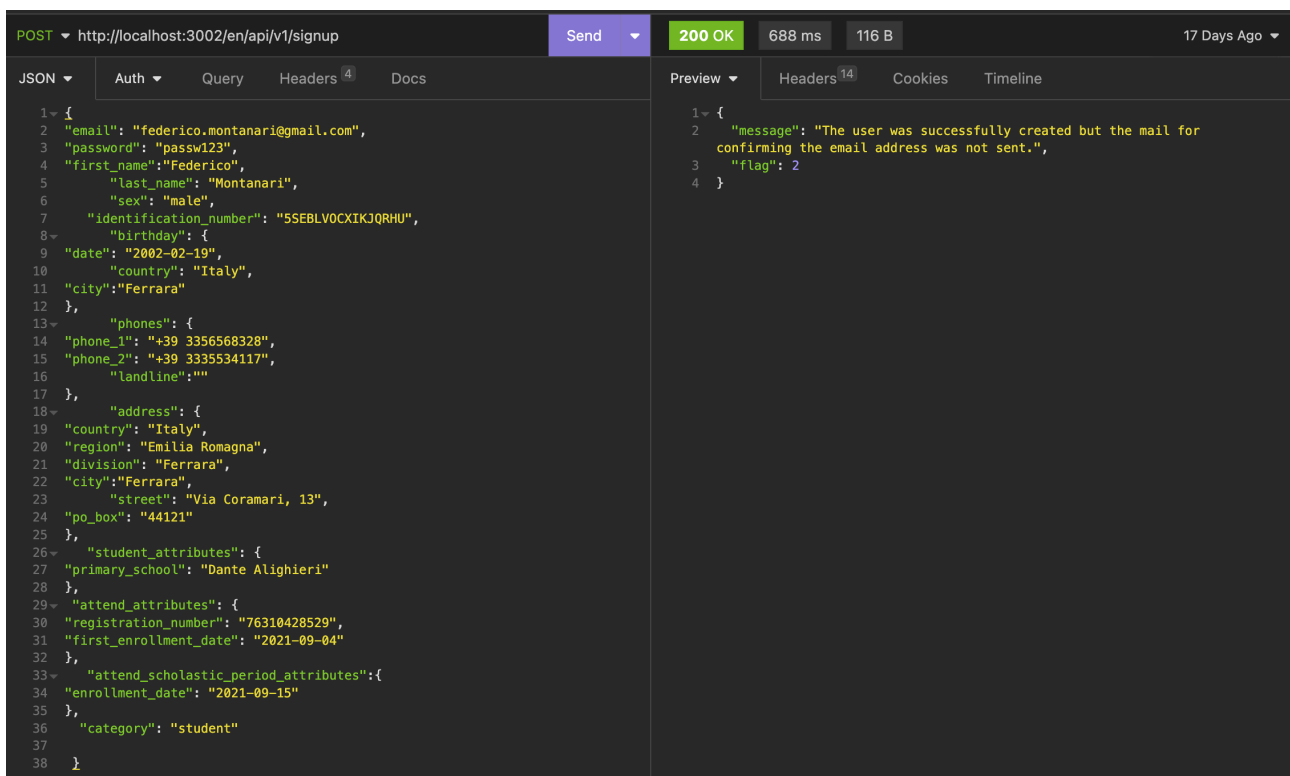


Figura 4.7 Messaggio di richiesta a sinistra e quello di risposta a destra in Insomnia

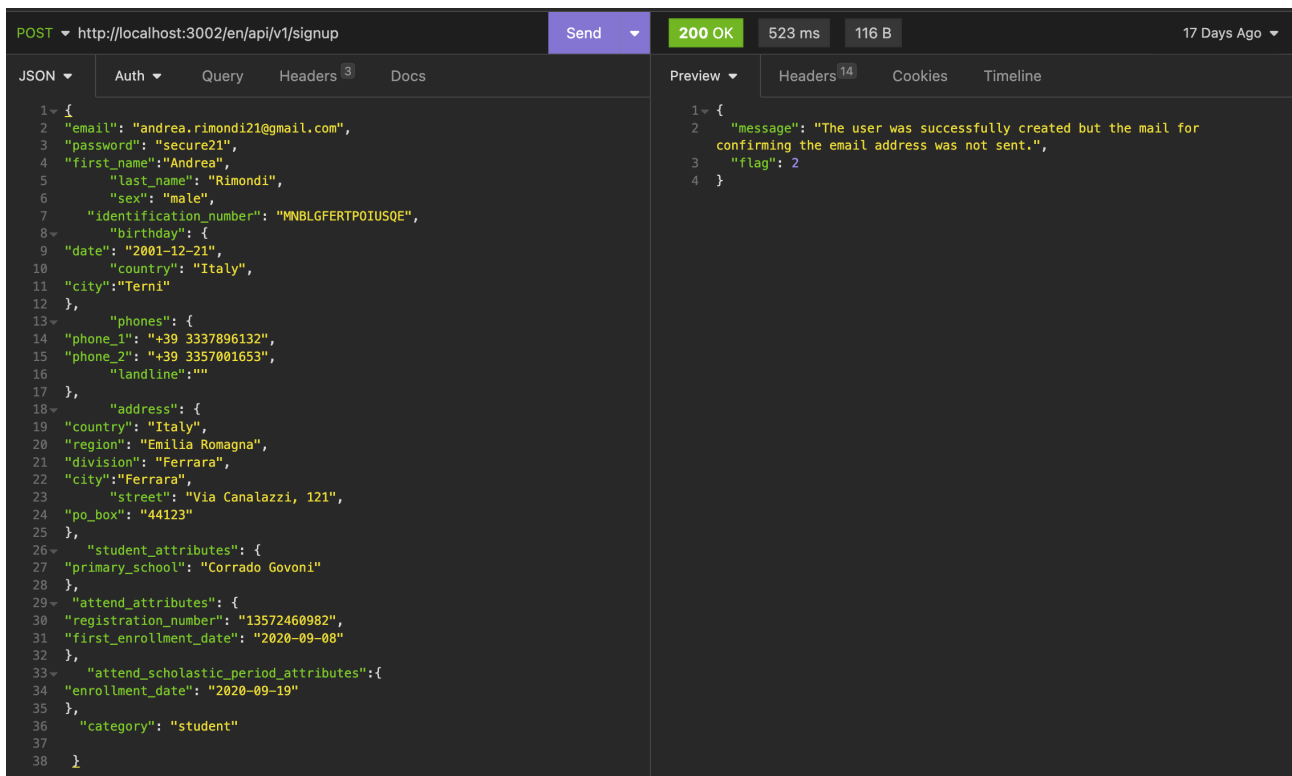


Figura 4.8 Messaggio di richiesta a sinistra e quello di risposta a destra in Insomnia

Come si può osservare dalle figure 4.6, 4.7 e 4.8 è stato sufficiente inserire nel body delle richieste HTTP i parametri consentiti.

17	18	marta.malagutti6133@gmail.com	\$2a\$12\$xB5jEPaDWcpEnmUNb2h0uYH.Qsnp7Nrvjwbxw66JzxW.5dPly5I2	[null]
18	19	federico.montanari@gmail.com	\$2a\$12\$cnaMT2B7cEFZZMWhid9r40Hskq3LaweJ8/cpGzoA./zJbnzB7LOZG	[null]
19	20	andrea.rimondi21@gmail.com	\$2a\$12\$7N/vn2JaZw66C55yx4P33e1fm/nxCHZNSSvrwhk1ggYYE..9oSMx...	[null]

Figura 4.9 Le ultime tre righe della tabella “users” in pgAdmin 4. Sono visibili sono le prime quattro colonne della tabella, ovvero id, email, encrypted_password e reset_password_token

17	17	Dante Alighieri
18	18	Dante Alighieri
19	19	Corrado Govoni

Figura 4.10 Le ultime tre righe della tabella “students”. Le due colonne della tabella sono id e primary_school

	id [PK] bigint	school_id bigint	student_id bigint	registration_number character varying	first_enrollment_date date	created_at timestamp without time zone (6)	updated_at timestamp without time zone (6)
1	1	2	14	[null]	[null]	2023-08-28 16:08:41.242029	2023-08-28 16:08:41.242029
2	2	2	15	[null]	[null]	2023-08-28 16:17:33.045126	2023-08-28 16:17:33.045126
3	3	3	17	87420539631	2020-09-05	2023-08-28 16:36:12.887503	2023-08-28 16:36:12.887503
4	4	3	18	76310428529	2021-09-04	2023-09-02 14:09:49.856819	2023-09-02 14:09:49.856819
5	5	3	19	13572460982	2020-09-08	2023-09-02 14:18:34.428892	2023-09-02 14:18:34.428892

Figura 4.11 La tabella “attends”: si notino le ultime tre righe

	id [PK] bigint	scholastic_period_id bigint	attend_id bigint	enrollment_date date	created_at timestamp without time zone (6)	updated_at timestamp without time zone (6)	classroom_id bigint
1	1	1	2	[null]	2023-08-28 16:17:33.118768	2023-08-28 16:17:33.118768	[null]
2	2	1	3	2020-09-18	2023-08-28 16:36:12.988971	2023-08-28 16:36:12.988971	[null]
3	3	1	4	2021-09-15	2023-09-02 14:09:49.943533	2023-09-02 14:09:49.943533	[null]
4	4	1	5	2020-09-19	2023-09-02 14:18:34.455567	2023-09-02 14:18:34.455567	[null]

Figura 4.12. La tabella “attend_scholastic_periods”. Non sono visibili le ultime due colonne, ovvero parent1_id e parent2_id

Dalle precedenti quattro figure è possibile notare come la creazione di un solo studente introduca un nuovo record in numerose tabelle della base di dati.

4. Pagamento delle diverse tasse per gli studenti creati precedentemente

In questa sezione si suppone che i tre studenti paghino le tre tasse previste dal regolamento della scuola, secondo diverse modalità; tuttavia, ciascuno di essi paga una tassa in ritardo, di conseguenza è costretto a pagare un contributo aggiuntivo. In particolare, il primo studente paga la tassa di iscrizione in ritardo, il secondo quella di frequenza e il terzo quella d'esame.

Si suppone, inoltre, che i pagamenti vengano creati in anticipo dalla scuola, per poi essere completati nel momento in cui uno studente versa effettivamente il denaro. È stato necessario, quindi, rendere opzionali alcune colonne della tabella "payments" (quando la scuola crea un pagamento non può conoscere in anticipo alcune delle informazioni richieste) e aggiungere un endpoint al controllore di Payment:

```
before_action :set_payment, only: [:show, :update, :destroy, :complete_payment]

def complete_payment
  begin
    if params[:Method] == 'Serve'
      if ["Cash", "Credit transfer"].include?(params[:serve_type])
        @payment.complete_pay(params[:Method], params[:recipient_id], params[:serve_type])
      else
        render json: { error: "serve_type must be Cash or Credit transfer" }, status: :unprocessable_entity
      end
    else
      @payment.complete_pay(params[:Method], params[:recipient_id])
    end
    render json: { message: "Payment has been successfully completed" }
  rescue => exception
    render json: { error: "Failed to complete the selected payment" }, status: :unprocessable_entity
  end
end
```

Nel caso in cui l'attributo "Method" del pagamento sia uguale a "Serve", si verifica che l'attributo "serve_type" (obbligatorio in questa specifica situazione) sia impostato a "Cash" oppure a "Credit transfer"; se è così, viene effettivamente completato il pagamento con i dati non ancora inseriti, vale a dire "Method", "serve_type" e "recipient_id". Se, invece, "Method" non dovesse essere uguale a "Serve", verrebbe completata l'operazione aggiungendo solamente "Method" e "recipient_id", lasciando "serve_type" al suo valore di default, ovvero nil. Infatti, se la persona addetta al pagamento è un impiegato (o impiegata) della scuola in questione, l'attributo "serve_type", il cui scopo è quello di specificare se sono stati utilizzati dei contanti oppure il credito telefonico dello studente, può essere solamente "Cash" o "Credit transfer". Se, però, "Method" non dovesse essere uguale a "Serve" allora l'unica possibilità per "serve_type" sarebbe il valore nil.

I metodi `set_payment` (generato automaticamente da Rails) e `complete_pay` (all'interno del modello `Payment`), invece, sono definiti nel modo seguente:

```
def set_payment
  @payment = Payment.find(params[:id])
end
```

```
def complete_pay(method, recipient_id, serve_type = nil)
  self.update(Method: method, serve_type: serve_type,
recipient_id: recipient_id, Status: 'Completed', payment_date:
Date.today)
end
```

	id [PK] bigint	Method text	Status character varying	serve_type text	payment_date date	recipient_id bigint	attend_id bigint	attend_scholastic_period_id bigint	fee_id bigint
1	1	Online	Completed	[null]	2023-09-30	5	3	2	1
2	2	Online	Completed	[null]	2023-09-30	5	3	2	2
3	3	Online	Completed	[null]	2023-09-30	5	3	2	3
4	4	Online	Completed	[null]	2023-09-30	5	3	2	4
5	5	At the bank	Completed	[null]	2023-09-30	6	4	3	5
6	6	At the bank	Completed	[null]	2023-09-30	6	4	3	6
7	7	At the bank	Completed	[null]	2023-09-30	6	4	3	8
8	8	At the bank	Completed	[null]	2023-09-30	6	4	3	7
9	9	Serve	Completed	Cash	2023-09-30	7	5	4	9
10	10	Serve	Completed	Credit transfer	2023-09-30	7	5	4	10
11	11	Serve	Completed	Cash	2023-09-30	7	5	4	11
12	12	Serve	Completed	Credit transfer	2023-09-30	7	5	4	12

Figura 4.13 La tabella “payments”: non sono visibili le ultime due colonne, ovvero `created_at` e `updated_at`

5. Implementazione di un endpoint che restituisca i pagamenti effettuati in tempo e quelli in ritardo in `payments_controller.rb`

```
def get_payments_by_status
  if params[:status] == 'early'
    @payments = Payment.early
  elsif params[:status] == 'late'
    @payments = Payment.late
  else @payments = Payment.all
  end
  render json: @payments
end
```

Per realizzare questo endpoint è stato necessario aggiungere il parametro `status` alla lista dei parametri permessi, così da poter specificare il tipo dei pagamenti desiderati all'interno della richiesta HTTP. Nel caso in cui non venga precisata questa informazione, verranno restituiti tutti i pagamenti. Oltre a questo sono stati aggiunti due *scope* al modello `Payment`, ovvero due metodi che eseguono un compito che può essere necessario ripetere in più punti del progetto (all'interno dell'endpoint considerato in questo caso specifico)

```
scope :early, -> { where("Due_date >= payment_date") }
scope :late, -> { where("Due_date < payment_date") }
```

È utile ricordare che, per tutti e tre gli endpoint definiti, è stato necessario apportare delle modifiche a `routes.rb`, in modo da renderli utilizzabili. Infatti, `routes.rb` stabilisce una corrispondenza tra le parti implementate (i tre endpoint) e gli URL impiegati in `Insomnia` per effettuare le richieste HTTP.

Terminata questa prima fase di simulazioni con Insomnia, è stata avviata l'ultima: è stata generata, in primo luogo, una nuova tassa di frequenza, questa volta suddivisa in tre rate (si sottolinea nuovamente che il numero di rate viene deciso dalla scuola).

13	13	First installment of tuition fee	818747	2023-12-01	182.69	false	tuition	1
14	14	Second installment of tuition fee	306065	2024-03-01	76.26	false	tuition	2
15	15	Third installment of tuition fee	546786	2024-06-01	30.5	false	tuition	3

Figura 4.14 La ultime tre righe della tabella “fees” in pgAdmin 4

Successivamente in Insomnia sono stati simulati i pagamenti effettuati dai tre studenti già creati precedentemente. Più precisamente: il primo studente paga le prime due rate della tassa di frequenza mediante trasferimento del credito telefonico e l'ultima in contanti, il secondo studente paga le prime due rate online e la terza in contanti, il terzo studente paga tutte e tre le rate grazie alla sua borsa di studio.

13	13	[null]	Uncompleted	[null]	[null]	[null]	3	2	13
14	14	[null]	Uncompleted	[null]	[null]	[null]	3	2	14
15	15	[null]	Uncompleted	[null]	[null]	[null]	3	2	15

Figura 4.15 Dopo la creazione dei pagamenti corrispondenti alle tre rate della tassa di frequenza per il primo studente vengono aggiunte (una per volta) le righe sopra riportate nella tabella “payments”; le colonne mostrate in figura sono: id, Method, Status, serve_type, payment_date, recipient_id, attend_id, attend_scholastic_period_id e fee_id

The screenshot shows a REST client interface with a POST request to `http://localhost:3002/payments`. The request body is a JSON object with the following structure:

```

1 {
2   "attend_id": 3,
3   "attend_scholastic_period_id": 2,
4   "fee_id": 13
5 }

```

The response status is **201 Created** with a response time of 354 ms and a body size of 786 B. The response body is a JSON object with the following structure:

```

1 {
2   "id": 13,
3   "Method": null,
4   "Status": "Uncompleted",
5   "serve_type": null,
6   "payment_date": null,
7   "recipient_id": null,
8   "attend_id": 3,
9   "attend_scholastic_period_id": 2,
10  "fee_id": 13,
11  "recipient": null,
12  "attend": {
13    "monitor": {
14      "id": 24,
15      "create_who_fullname": "Marta Malagutti",
16      "update_who_fullname": "Marta Malagutti",
17      "status": 4,
18      "start_date": null,
19      "end_date": null
20    }
21  },
22  "attend_scholastic_period": {
23    "monitor": {
24      "id": 25,
25      "create_who_fullname": "Marta Malagutti",
26      "update_who_fullname": "Marta Malagutti",
27      "status": 4,
28      "start_date": null,
29      "end_date": null
30    }
31  },
32  "id": 2,
33  "scholastic_period_id": 1,
34  "attend_id": 3,
35  "enrollment_date": "2020-09-18",
36  "classroom_id": null
37  },
38  "fee": {
39    "id": 13,
40    "Description": "First installment of tuition fee",
41    "Invoice_code": 818747,
42    "Due_date": "2023-12-01",
43    "Amount": 182.69,
44    "is_fine": false,
45    "category": "tuition",
46    "installment": 1

```

Figura 4.16 Messaggio di richiesta (a sinistra) e quello di risposta (a destra) in merito alla creazione del pagamento della prima rata della tassa di frequenza per il primo studente

The screenshot shows a REST client interface with a POST request to `http://localhost:3002/payments/13/complete_payment`. The request body is a JSON object with the following structure:

```

1 {
2   "Method": "Serve",
3   "serve_type": "Credit transfer",
4   "recipient_id": 7
5 }

```

The response status is **200 OK** with a response time of 62.2 ms and a body size of 53 B. The response body is a JSON object with the following structure:

```

1 {
2   "message": "Payment has been successfully completed"
3 }

```

Figura 4.17 Messaggio di richiesta (a sinistra) e quello di risposta (a destra) in merito al completamento del pagamento della prima rata della tassa di frequenza per il primo studente (Figura 4.16)

13	13	Serve	Completed	Credit transfer	2023-09-30	7	3	2	13
14	14	Serve	Completed	Credit transfer	2023-09-30	7	3	2	14
15	15	Serve	Completed	Cash	2023-09-30	7	3	2	15

Figura 4.18 Dopo aver completato i pagamenti delle tre rate della tassa di frequenza per il primo studente vengono opportunamente aggiornate le righe mostrate in Figura 4.15

Successivamente sono stati ripetuti questi ultimi passaggi (creazione delle rate, creazione dei pagamenti corrispondenti e completamento di questi ultimi) per il secondo e per il terzo studente, concludendo così anche l'ultima fase di questo progetto.

Conclusioni

Sulla base di quanto descritto in queste pagine è possibile desumere che il lavoro svolto, sebbene circoscritto, abbia costituito un tassello in più in quella che è una piattaforma così complessa come quella in fase di costruzione da SYSAIT.

Per portare a termine questo progetto sono state affrontate numerose problematiche, qualunque fosse la specifica situazione, soprattutto a causa del fatto che la soluzione più adeguata è spesso stata trovata in stati avanzati del compito in questione: talvolta questo ha comportato la modifica degli step precedenti ad esso.

A questo punto, nonostante alcune mancanze rimaste in sospeso (ed eventualmente ancora da individuare) come una gestione più precisa degli errori e la possibilità di pagare due o anche più rate attraverso un pagamento unico, è possibile procedere con l'implementazione della parte front-end, così come con la progettazione e l'implementazione di un modulo non ancora presente nell'applicazione, proprio come lo era il modulo riservato ai pagamenti prima di questa attività di tesi.

Bibliografia

1. Ramez A. Elmasri, Shamkant B. Navathe. Sistemi di basi di dati. Fondamenti e complementi. Settima edizione, 2018, Pearson Addison Wesley

Sitografia

1. https://guides.rubyonrails.org/getting_started.html
2. https://guides.rubyonrails.org/active_record_basics.html
3. https://guides.rubyonrails.org/active_record_migrations.html
4. https://guides.rubyonrails.org/active_record_validations.html
5. https://guides.rubyonrails.org/association_basics.html
6. https://guides.rubyonrails.org/active_record_querying.html
7. https://guides.rubyonrails.org/action_controller_overview.html
8. <https://guides.rubyonrails.org/routing.html>
9. https://guides.rubyonrails.org/command_line.html
10. <https://dev.to/nemwelboniface/api-with-rails-7-ngh>
11. https://guides.rubyonrails.org/api_app.html
12. <https://api.rubyonrails.org/classes/ActiveRecord/NestedAttributes/ClassMethods.html>
13. <https://en.wikipedia.org/wiki/Diagrams.net>
14. <https://www.sysaitechnology.com/home>

Ringraziamenti

Quest'ultima pagina ho scelto di dedicarla alle persone che, direttamente o indirettamente, hanno contribuito a rendere più felice la mia vita e, di conseguenza, più facile affrontare questi ultimi tre anni.

In primis vorrei ringraziare con tutto il cuore la mia famiglia, la quale ha saputo sempre supportarmi, aiutarmi e spingermi ad essere la versione migliore di me e senza la quale non sarebbe stato possibile nulla di tutto questo.

Un ringraziamento speciale lo rivolgo a Federico che con il suo amore, la sua sincerità, la sua pazienza e la sua allegria ha reso gli ultimi quattro anni nella mia vita indimenticabili e pieni di gioia.

Ringrazio profondamente tutti i miei amici e amiche di una vita ormai: Chiara e Margherita, le prime due amiche che abbia mai avuto, Alice e Martina, insieme dall'inizio della scuola elementare, Giovanni, un punto di riferimento dalla scuola media, Lisa, unite grazie ad un viaggio e inseparabili da allora, Andrea e Anastasia, dal primo anno di scuola superiore due presenze irrinunciabili, fondamentali e caotiche nella mia vita.

Vorrei ringraziare tanto anche le persone conosciute proprio grazie all'università e quelle conosciute in tempi più recenti, le quali hanno reso divertente e più leggero questo percorso: Vincenzo e Maria Teresa, Mattia e Riccardo, Marta e Riccardo, Simone, Alessandro e Marcello. Mi ritengo fortunata ad avere incontrato persone così speciali in un momento della vita così delicato e importante.

Infine, ringrazio la professoressa Elena Bellodi e il dottor Arnaud Nguembang Fadja che con la loro gentilezza e professionalità hanno saputo accompagnarmi negli ultimi mesi, quelli più intensi, di questo lungo e faticoso percorso triennale.