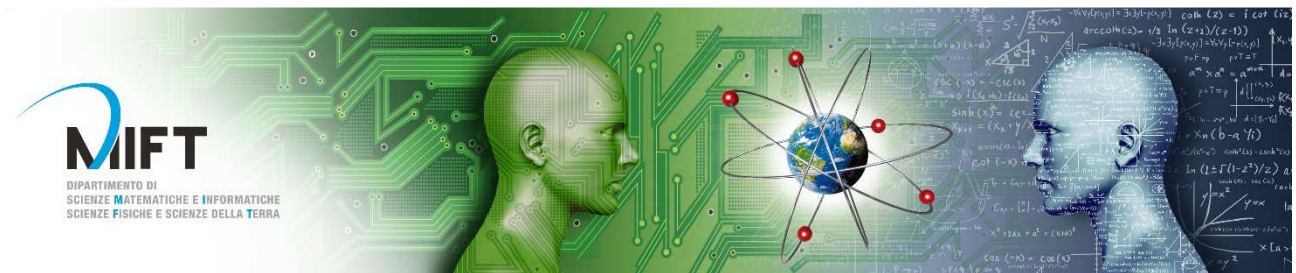




# Università degli Studi di Messina



Relazione progetto programmazione III.  
Sistema di segnalazione per i problemi nei  
locali dell'Ateneo.

Docente:  
Andrea Nucita.

Studenti:  
Emanuele Infortuna.  
Giuseppe Primerano.

## Indice:

1. Presentazione.
2. Modelli per la gestione dei dati.
  - 2.1 Modello concettuale.
  - 2.2 Modello logico.
3. Descrizione tabelle.
4. Laravel.
  - 4.1 Migration.
  - 4.2 Controller.
  - 4.3 Model.
  - 4.4 Route.
  - 4.5 Seeder.
5. Implementazioni software.
6. Mappa sito web.
  - 6.1 Homepage.
  - 6.2 Mappa.
  - 6.3 Login.
  - 6.4 Area riservata direttori.
  - 6.5 Area riservata tecnici.
7. RESTful API's.

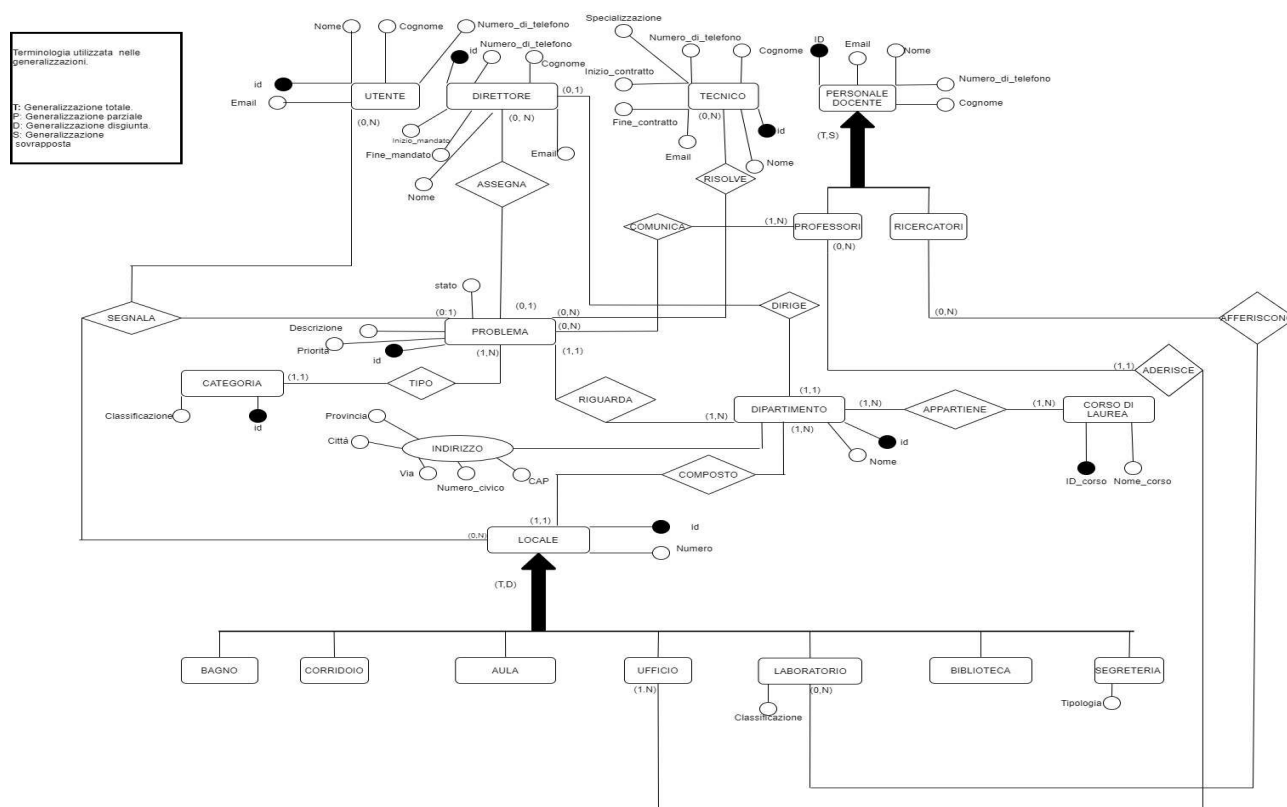
## 1. Presentazione.

L'obiettivo di questo progetto è quello di voler realizzare un sito web in cui tutti gli utenti che vengono a contatto con i locali dell'Ateneo possano segnalare dei problemi riscontrati all'interno degli stessi. I problemi segnalati verranno inseriti in un opportuno database e potranno essere visualizzati dai direttori di dipartimento che avvertiranno i tecnici che lavorano per l'Università per fare in modo che i problemi vengano risolti. L'utilizzo del nostro sito web fa sì che gli utenti aiutino i direttori dei diversi dipartimenti a venire a conoscenza di tutte le problematiche presenti nei locali di loro competenza e inoltre aiuterà i direttori a gestirle. Un altro fine che si pone il nostro progetto è di migliorare i prodotti e i servizi all'interno degli spazi universitari, creando un sistema di gestione delle segnalazioni e un sistema di gestione per le mansioni che dovranno essere svolte dai tecnici.

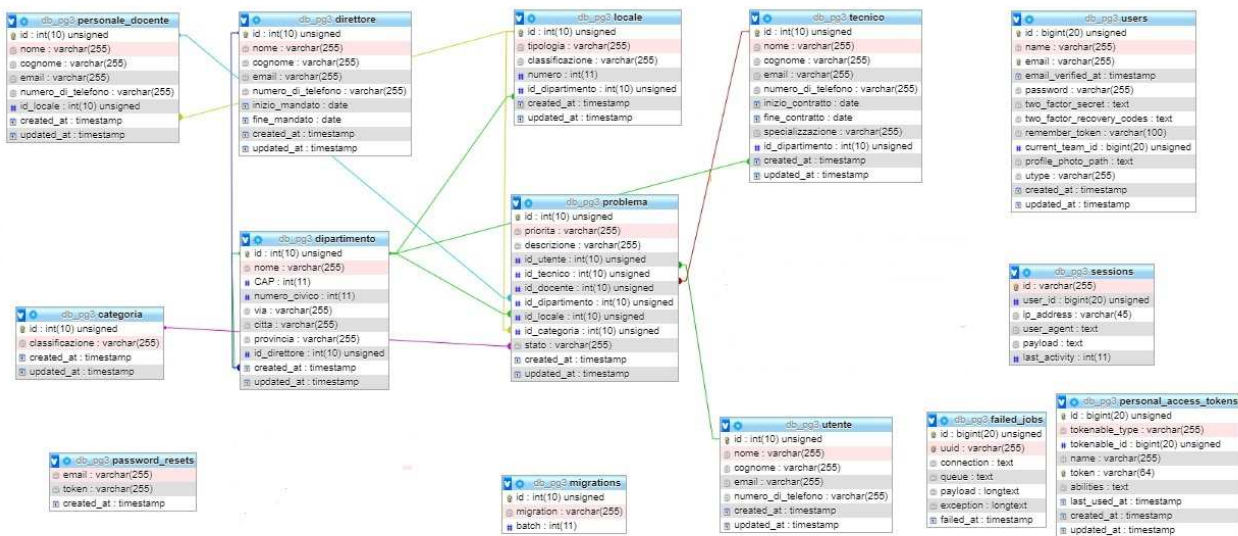
## 2. Modelli per la gestione dei dati.

Nei paragrafi seguenti verranno illustrati i modelli che abbiamo utilizzato nella nostra applicazione per la gestione dei dati.

### 2.1 Modello concettuale.



## 2.2 Modello logico



## 3. Descrizione tabelle.

Piccola precisazione preliminare; gli attributi created\_at e updated\_at vengono creati in modo automatico da Laravel durante la migrazione delle tabelle che avviene mediante il comando:

```
php artisan migrate
```

Di seguito abbiamo indicato con il colore rosso le primary key e in verde le foreign key.

### UTENTE:

Tale entità rappresenta, l'utente che effettuerà la segnalazione.

#### Descrizione attributi.

**Id:** Identificativo univoco di ogni utente.

**NOME:** Identifica il nome dell'utente.

**COGNOME:** Identifica il cognome dell'utente.

**EMAIL:** Identifica l'indirizzo di posta elettronica di ciascun utente che fa una segnalazione.

**NUMERO DI TELEFONO:** rappresenta il numero di telefono dell'utente.

**CREATED\_AT:** rappresenta l'ora e la data della creazione.

**UPDATED\_AT:** rappresenta l'ora e la data della modifica di un eventuale segnalazione.

### PERSONALE DOCENTE:

Tale entità rappresenta colui che ha il compito di insegnare all'interno dell'Università. E ha la possibilità di segnalare un eventuale problema

#### Descrizione attributi

**Id:** Identificativo univoco di ogni docente.

**NOME:** Identifica il nome del docente.

**COGNOME:** Identifica il cognome del docente.

**EMAIL:** Identifica l'indirizzo di posta elettronica di ciascun docente che fa una segnalazione.

**NUMERO DI TELEFONO:** rappresenta il numero di telefono del docente.

**ID\_LOCALE:** rappresenta la foreign key per indicare il locale (ufficio o laboratorio) a cui ha accesso ogni docente.

**CREATED\_AT:** rappresenta l'ora e la data della creazione.

**UPDATED\_AT:** rappresenta l'ora e la data della modifica.

## **DIRETTORE:**

Tale entità rappresenta il gestore del sistema che stiamo creando.

### *Descrizione attributi*

**Id:** Identificativo univoco di ogni direttore.

**NOME:** Identifica il nome del direttore.

**COGNOME:** Identifica il cognome del direttore.

**EMAIL:** Identifica l'indirizzo di posta elettronica di ciascun direttore che fa una segnalazione.

**NUMERO DI TELEFONO:** rappresenta il numero di telefono del direttore.

**INIZIO\_MANDATO:** indica l'inizio del mandato di un direttore.

**FINE\_MANDATO:** indica la fine del mandato di un direttore.

**CREATED\_AT:** rappresenta l'ora e la data della creazione.

**UPDATED\_AT:** rappresenta l'ora e la data della modifica.

## **TECNICO.**

Tale entità rappresenta il personale responsabile che sarà incaricato, da parte del direttore, a risolvere le problematiche segnalate dagli utenti.

### *Descrizione attributi*

**Id:** Identificativo univoco di ogni tecnico.

**NOME:** Identifica il nome del tecnico.

**COGNOME:** Identifica il cognome del tecnico.

**EMAIL:** Identifica l'indirizzo di posta elettronica di ciascun tecnico.

**NUMERO DI TELEFONO:** rappresenta il numero di telefono del tecnico.

**INIZIO\_CONTRATTO:** rappresenta la data in cui il tecnico ha iniziato a lavorare per l'Università.

**FINE\_CONTRATTO:** rappresenta la data in cui il tecnico finirà di lavorare per l'Università.

**SPECIALIZZAZIONE:** indica la specializzazione del tecnico.

**ID\_DIPARTIMENTO:** foreign key usata per indicare per quale dipartimento lavora il tecnico.

**CREATED\_AT:** rappresenta l'ora e la data della creazione.

**UPDATED\_AT:** rappresenta l'ora e la data della modifica.

## **PROBLEMA:**

Al suo interno troviamo tutte le segnalazioni fatte dagli utenti o docenti.

### *Descrizione attributi.*

**Id:** Identificativo univoco di un problema.

**PRIORITA':** indica la priorità di un problema definita su tre parametri (alta, media e bassa).

**DESCRIZIONE:** rappresenta la descrizione di un problema.

**ID\_UTENTE:** foreign key usata per indicare l'utente che ha effettuato la segnalazione.

**ID\_TECNICO:** foreign key usata per indicare il tecnico che dovrà risolvere il problema.

**ID\_DOCENTE:** foreign key usata per indicare il docente che ha effettuato la segnalazione.

**ID\_DIPARTIMENTO:** foreign key usata per indicare in quale dipartimento è stato segnalato il problema.

**ID\_LOCALE:** foreign key usata per indicare il locale in cui è stato riscontrato un problema.

**ID\_CATEGORIA:** foreign key usata per indicare a quale categoria appartiene un problema segnalato.

**STATO:** indica in che stato si trova la risoluzione di un problema, impostato su tre parametri (incompleto, parzialmente completo e completo).

**CREATED\_AT:** rappresenta l'ora e la data della creazione.

**UPDATED\_AT:** rappresenta l'ora e la data della modifica.

## CATEGORIA.

Tale entità rappresenta le diverse categorie del problema, nella nostra applicazione sono quattro (elettrico, sanitario, informatico e strutturale).

### *Descrizione attributi*

**Id:** Identificativo univoco per una categoria.

**CLASSIFICAZIONE:** Indica il nome della categoria, abbiamo impostato quattro parametri (elettrico, sanitario, informatico e strutturale).

**CREATED\_AT:** rappresenta l'ora e la data della creazione.

**UPDATED\_AT:** rappresenta l'ora e la data della modifica.

## DIPARTIMENTO:

Al suo interno troviamo tutti i dipartimenti dell'ateneo.

### *Descrizione attributi*

**Id:** Identificativo univoco di un dipartimento.

**NOME:** indica il nome del dipartimento.

**CAP:** indica il codice di avviamento postale di un dipartimento.

**NUMERO\_CIVICO:** indica il numero civico di un dipartimento.

**VIA:** indica la via del dipartimento.

**CITTA:** indica la città in cui si trova un dipartimento.

**PROVINCIA:** indica la provincia in cui si trova un dipartimento.

**ID\_DIRETTORE:** foreign key usata per indicare il direttore di un dipartimento.

**CREATED\_AT:** rappresenta l'ora e la data della creazione.

**UPDATED\_AT:** rappresenta l'ora e la data della modifica.

## LOCALE:

Al suo interno troviamo tutti i locali presenti in ciascun dipartimento.

### *Descrizione attributi*

**Id:** Identificativo univoco di un locale.

**TIPOLOGIA:** indica la tipologia del locale. Ad esempio aula, ufficio, segreteria ecc.

**CLASSIFICAZIONE:** indica la classificazione di un locale. Ad esempio aula relax, aula studio, ecc.

**NUMERO:** indica il numero del locale.

**ID\_DIPARTIMENTO:** foreign key usata per indicare il dipartimento a quale afferisce un locale.

**CREATED\_AT:** rappresenta l'ora e la data della creazione.

**UPDATED\_AT:** rappresenta l'ora e la data della modifica.

## 4.1 Laravel.

Nel nostro progetto abbiamo deciso di utilizzare un framework PHP, più precisamente Laravel (versione 8). In questo paragrafo andremo a elencare quelle che sono state le motivazioni per cui abbiamo deciso di utilizzare questo framework, inoltre andremo a spiegare brevemente le componenti principali di Laravel, spiegazione che sarà poi ripresa nei paragrafi successivi nel momento in cui andremo ad analizzare nel dettaglio la nostra applicazione web.

Iniziamo elencando di seguito le motivazioni per cui abbiamo deciso di utilizzare Laravel:

- Laravel rende i siti web molto performanti e ci permette di capire e studiare il mondo dello sviluppo ad “oggetti”.
- Ha molte funzionalità come la migrazione del database, il seeding del database, ecc.
- Funziona su basi modulari con molte funzioni di build disponibili. Ciò significa che le applicazioni possono essere costruite rapidamente senza bisogno di ore di lavoro e molte righe di codice.
- Possiede dei test integrati per garantire che le nuove modifiche apportate all'applicazione non interrompano inaspettatamente l'applicazione stessa.
- Laravel ha una fiorente e utile comunità di sviluppatori.
- Laravel fornisce una configurazione immediata per il sistema di autenticazione e autorizzazione. Grazie a questo è possibile avere nella nostra applicazione autenticazione e autorizzazione sicura.
- Laravel offre una sicurezza delle applicazioni web molto forte. Utilizza un meccanismo di password con funzioni di hash in modo che la password non venga mai salvata come testo normale nel database. Utilizza anche "Bcrypt Hashing Algorithm" per generare una password crittografata. Inoltre, questo framework utilizza istruzioni SQL create appositamente per prevenire gli attacchi di SQL injection.
- Guardando gli ultimi dati forniti dal web, Laravel si posiziona al primo posto tra i framework PHP più utilizzati nel 2020, quindi una sua accurata conoscenza offre molte opportunità di lavoro.

Ora analizzeremo brevemente quelle che sono le caratteristiche principali di Laravel. Partiamo da **MVC** (Model View Controller) è il modello di Laravel che garantisce chiarezza di logica e di presentazione. Questo supporto all'architettura aiuta a migliorare le prestazioni, consentendo una migliore documentazione e dispone di molte funzioni integrate. Il **model** si occupa di gestire la logica dei dati della richiesta effettuata dall'utente. Quindi il model



interagisce con il database e si occupa della validazione, salvataggio, modifica ecc. Esso fornisce i metodi per accedere ai dati utili all' applicazione web. Le **view** forniscono gli elementi dell'interfaccia grafica che arrivano dal model in una forma utile per l'utente finale. Infine abbiamo i **controller** che ricevono le richieste da parte dell'utente e chiedono informazioni al model e alle view.

Un altro elemento interessante è sicuramente il Laravel **artisan** (artigiano) è uno strumento integrato a riga di comando che viene utilizzato per creare la struttura del database, costruire la migrazione e fare altre azioni. Inoltre, può essere usato per generare immediatamente i file MVC di base tramite la riga di comando e gestire tali risorse e le rispettive configurazioni. È anche un comando che consente di eseguire la maggior parte di quelle attività di programmazione noiose e ripetitive che molti sviluppatori PHP evitano di eseguire manualmente.

Infine il framework Laravel offre l'ORM (Object Relational Mapping) che consente ai programmatori di applicazioni Web di eseguire query sul database con la sintassi PHP anziché scrivere codice SQL, può essere visto come un traduttore che si occupa di tradurre da linguaggio "OO" a linguaggio relazionale. L'ORM è un'interfaccia che fornisce, tramite programmazione a oggetti, tutti i servizi inerenti alla persistenza dei dati astruendo le caratteristiche del DBMS usato; significa che possiamo gestire le entità e le relazioni di un DBMS tramite PHP in modo agevole scrivendo poche righe di codice, avendo una portabilità superiore e scrivendo un'applicazione più stratificata per quanto riguarda la persistenza dati a vantaggio della modularità del sistema. L'ORM usato in Laravel viene chiamato Eloquent. Facendo uso dell'ORM ad ogni tabella del nostro DB corrisponderà un modello che ci permetterà di agire su di essa direttamente da riga di comando (usando artisan) o direttamente dal codice, senza interpellare il DB stesso.

## 4.1 Migration

Il primo componente di Laravel che abbiamo utilizzato sono state appunto le "migration" consentono di modificare e condividere lo schema, esse sono una classe speciale che contengono un insieme di azioni da eseguire sul database, come la creazione o la modifica di una tabella. Le migration assicurano che il database venga impostato in modo identico ogni volta che si crea una nuova istanza della web application. Grazie alle migration si può gestire l'intera struttura del database dell'applicazione, si presentano come dei file PHP che creiamo per costruire e modificare il database della nostra applicazione nel tempo.



Per creare una “migration” si usa nel terminale il seguente comando:

```
php artisan make:migration CreateProblema
```

Eseguendo il comando, la “migration” sarà inserita nella directory “database/migrations”. Ogni “migration” ha un nome che contiene un timestamp che permette a Laravel di individuare l’ordine delle “migrations”.

La migration si presenta in questo modo:

```
class CreateProblema extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('problema', function (Blueprint $table) {
            $table->increments('id');
            $table->string('priorita');
            $table->string('descrizione');
            $table->integer('id_utente')->unsigned()->nullable();
            $table->foreign('id_utente')->references('id')->on('utente');
            $table->integer('id_tecnico')->unsigned()->nullable();
            $table->foreign('id_tecnico')->references('id')->on('tecnico')->onUpdate('CASCADE');
            $table->integer('id_docente')->unsigned()->nullable();
            $table->foreign('id_docente')->references('id')->on('personale_docente')->onDelete('CASCADE')->onUpdate('CASCADE');
            $table->integer('id_dipartimento')->unsigned()->nullable();
            $table->foreign('id_dipartimento')->references('id')->on('dipartimento');
            $table->integer('id_locale')->unsigned()->nullable();
            $table->foreign('id_locale')->references('id')->on('locale');
            $table->integer('id_categoria')->unsigned()->nullable();
            $table->foreign('id_categoria')->references('id')->on('categoria');
            $table->string('stato')->default('Incompleto');
            $table->timestamps();
        });
    }
}
```

All’interno di ogni migration troviamo due metodi ovvero:

- 1) up(): questo metodo permette di aggiungere una nuova tabella, aggiungere nuove colonne o nuovi indici al database;
- 2) down(): questo metodo permette di eliminare le precedenti modifiche.

Abbiamo creato una migration per ogni tabella del nostro database.

## 4.2 Controller.

Nel framework MVC, la lettera "C" sta per Controller, funge da guida del traffico tra le view e i model. Invece di definire tutta la logica di gestione delle richieste nei file di instradamento, abbiamo organizzato questo concetto utilizzando le classi controller. I controller possono raggruppare la logica di gestione delle richieste correlate in una singola classe, essi vengono memorizzati nella directory app/Http/Controllers. Un controller rappresenta anche una semplice classe che presenta diversi metodi pubblici.

Un esempio di controller presente nella nostra applicazione web è:

```
class TecnicoController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $id_tecnico = M_tecnico::select('id')
            ->where('email', '=', Auth::user()->email )->first();

        $list_problemi = M_segna1azione::All()
            ->where('id_tecnico', '=', $id_tecnico->id);

        return view('dashboard', ['list_problemi'=>$list_problemi, 'layout'=>'index']);
    }
}
```

I controller sono dotati delle seguenti funzioni **index**, **create**, **update**, **save**, **destroy** ecc. esse garantiscono il corretto svolgimento delle operazioni CRUD sul database, infatti noi ci siamo serviti delle suddette funzioni per generare tutte le operazioni della nostra web application.

Il comando per creare un controller è:

```
php artisan make:controller TecnicoController
```

## 4.3 Model.

Nel framework MVC, la lettera “M” sta per Model. In Laravel, model è una classe che rappresenta la struttura logica e la relazione della tabella dati sottostante. In Laravel, ciascuna tabella del database ha un "Modello" corrispondente che ci consente di interagire con quella tabella. I modelli ti danno la possibilità di recuperare, inserire e aggiornare le informazioni nella tua tabella dati. Tutti i modelli Laravel sono archiviati nella directory principale dell'app.

Per ogni tabella del nostro database abbiamo creato un model usando il comando:

```
php artisan make:model M_tecnico
```

Ogni Model ha la seguente struttura:

```
class M_tecnico extends Model
{
    use HasFactory;

    protected $table="tecnico";
    protected $fillable = [
        'id',
        'nome',
        'cognome',
        'email',
        'numero_di_telefono',
        'inizio_contratto',
        'fine_contratto',
        'specializzazione',
        'id_dipartimento'
    ];
}
```

## 4.4 Route.

Tutte le route Laravel sono definite nei file route, esse si trovano nella `routes/directory`. Questi file vengono caricati automaticamente dal framework. Il file `web.php` definisce i percorsi della nostra applicazione web., mentre il file `api.php` definisce tutti i percorsi della nostra RESTful API.

Ecco come si presentano i file nominati sopra.

`web.php`:

```
Route::get('/', [SegnalazioneController::class, 'create']);
Route::get('/', [SegnalazioneController::class, 'index']);
Route::post('/store', [SegnalazioneController::class, 'store']);

Route::get('/admin/dashboard', [DirettoreController::class, 'index'])->name('admin.dashboard');
Route::get('/admin/dashboard/edit/{id}', [DirettoreController::class, 'edit']);
Route::post('/admin/dashboard/update/{id}', [DirettoreController::class, 'update']);

Route::get('/user/dashboard', [TecnicoController::class, 'index'])->name('dashboard');
Route::get('/user/dashboard/edit/{id}', [TecnicoController::class, 'edit']);
Route::post('/user/dashboard/update/{id}', [TecnicoController::class, 'update']);

Route::get('/mappa', [MappaController::class, 'index']);
```

`api.php`:

```
/*
|-----
| API Routes
|-----
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/

Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
    return $request->user();
});

Route::ApiResource('/segnalazione', 'App\Http\Controllers\ApiSegnalazioneController');
```

Inoltre utilizzando il comando `php artisan route:list -c` abbiamo la possibilità di visualizzare in forma tabellare tutte le route che fanno parte della nostra applicazione web. Come si può vedere dall'immagine seguente la tabella mette a disposizione tutti gli URI (Uniform Resource Identifier) e i metodi HTTP ad essi associati.

```
$ php artisan route:list -c
```

Method	URI	Action
GET HEAD	/	App\Http\Controllers\SegnalazioneController@index
GET HEAD	admin/dashboard	App\Http\Controllers\DirettoreController@index
GET HEAD	admin/dashboard/edit/{id}	App\Http\Controllers\DirettoreController@edit
POST	admin/dashboard/update/{id}	App\Http\Controllers\DirettoreController@update
GET HEAD	api/segnalazione	App\Http\Controllers\ApiSegnalazioneController@index
POST	api/segnalazione	App\Http\Controllers\ApiSegnalazioneController@store
GET HEAD	api/segnalazione/{segnalazione}	App\Http\Controllers\ApiSegnalazioneController@show
PUT PATCH	api/segnalazione/{segnalazione}	App\Http\Controllers\ApiSegnalazioneController@update
DELETE	api/segnalazione/{segnalazione}	App\Http\Controllers\ApiSegnalazioneController@destroy
GET HEAD	mappa	App\Http\Controllers\MappaController@index
POST	store	App\Http\Controllers\SegnalazioneController@store
GET HEAD	user/dashboard	App\Http\Controllers\TecnicoController@index
GET HEAD	user/dashboard/edit/{id}	App\Http\Controllers\TecnicoController@edit
POST	user/dashboard/update/{id}	App\Http\Controllers\TecnicoController@update

## 4.5 Seeder.

Laravel include un metodo semplice per eseguire il seeding del database utilizzando classi seed. Tutte le classi seed vengono memorizzate nella cartella database/seeder. Per generare un seeder si utilizza il seguente comando:

```
php artisan make:seeder CreatePersonaleDocenteSeeder
```

Una classe seeder per impostazione predefinita è presente un solo metodo: run. Questo metodo viene chiamato quando viene eseguito il comando artisan db:seed. All'interno del metodo run, abbiamo inserito i dati per passarli al nostro database. In questo modo abbiamo evitato d'inserire i dati nell'altro metodo ovvero quello che si basa sulle query. Per eseguire il seeding del database abbiamo usato il comando:

```
php artisan db:seed
```

La motivazione per cui abbiamo usato i seeder è che abbinati alle migration ci permettono di sfruttare al meglio le potenzialità di entrambi.

Ecco come si presenta un seeder:

```

class CreatePersonaleDocenteSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $pdcente = [
            [
                'nome' => 'Alessandro',
                'cognome' => 'Arangio',
                'email' => 'alessandro.arangio@unime.it',
                'id_locale' => 983
            ],
            [
                'nome' => 'Antonino',
                'cognome' => 'Baglio',
                'email' => 'antonino.baglio@unime.it',
                'id_locale' => 985
            ],
        ],
    }
}

```

.

.

.

```

        foreach ($pdcente as $key => $value) {
            M_personale_docente::create($value);
        }
    }
}

```

Mediante l'utilizzo della funzione riportata sopra facciamo l'inserimento dei dati all'interno del database.

## 5. Implementazioni software.

Fatta questa breve premessa sul framework utilizzato vorremmo aggiungere che per lo sviluppo della nostra applicazione ci siamo avvalsi dell'uso del template engine di Laravel, conosciuto meglio come **blade**. Esso ci permette di scrivere php nel template e presenta delle strutture di controllo. Tutti i file sono riconoscibili dall'estensione **blade.php**. Mediante blade abbiamo potuto sviluppare la nostra applicazione usando Html e CSS.

Ci siamo avvalsi del software XAMPP ed in particolare abbiamo utilizzato alcuni dei suoi componenti tra cui:

- Web server: Apache HTTP server;

- DBMS: PHPmyAdmin.

Infine l'intero progetto è stato editato e sviluppato utilizzando l'editor di testi Visual Studio Code.

Per il test dell'API create ci siamo serviti di Postman.

## 6. Mappa sito web.

Nei paragrafi che seguono andremo ad illustrare la nostra applicazione web, analizzeremo inoltre quelle che sono state le nostre scelte dal punto di vista della programmazione. S'informa che la configurazione per la connessione al database è presente nel file **.env** di Laravel.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=db_pg3
DB_USERNAME=root
DB_PASSWORD=
```



## 6.1 Homepage.

The screenshot displays the homepage of the University of Messina. At the top left is the university's logo and name. At the top right are links for 'LOGIN' and 'MAPPA'. The central focus is a white contact form with the following fields: 'NOME' (text input), 'COGNOME' (text input), 'EMAIL' (text input with a hint), 'NUMERO DI TELEFONO' (text input), 'PRIORITA'' (dropdown menu), 'DESCRIZIONE' (text area), 'CATEGORIA' (dropdown menu), 'DIPARTIMENTO' (dropdown menu), and 'LOCALE' (dropdown menu). At the bottom of the form are two buttons: 'Inserisci' (blue) and 'Reset' (yellow). The background features a geometric wireframe pattern.

Questa è la pagina iniziale della nostra applicazione web, pagina con cui s'interfaceranno tutti coloro che vogliono usufruire del servizio da noi creato. Come si può vedere questa pagina è composta da un **form HTML** in cui gli utenti hanno la possibilità d'inserire e di inviare la segnalazione del problema riscontrato, una volta premuto il bottone "*inserisci*" la segnalazione verrà inviata al server e salvata all'interno della tabella **PROBLEMA** nel database da noi creato. Il form è composto da nove campi;

- I primi due sono del tipo:

```
<input type='text'
```

- Il terzo è del tipo:

```
<input type='email'
```

- Il quarto è del tipo:

```
<input type='text'
```

- Il quinto campo è del seguente tipo:

```
<label>PRIORITA'</label><br>
<select class="form-control" name="priorita" required >
  <option value="" disabled selected>Inserisci Priorità</option>
  <option value="Alta">Alta</option>
  <option value="Media">Media</option>
  <option value="Bassa">Bassa</option>
</select>
```

- Il sesto campo è del seguente tipo:

```
<label>DESCRIZIONE</label>
<textarea class="form-control" name="descrizione" rows="3" cols="30" placeholder="Inserisci Descrizione" required></textarea>
```

- Il settimo campo è del seguente tipo:

```
<label>CATEGORIA</label>
<select class="form-control" name="classificazione" required>
  <option value="" disabled selected>Inserisci Categoria</option>
  @foreach($cat as $c)
    <option value="{{ $c->id }}">{{ $c->classificazione }}</option>
  @endforeach
</select>
```

- L'ottavo campo è del seguente tipo:

```
<label>LOCALE</label>
<select class="form-control" name="locale" required>
  <option value="" disabled selected>Inserisci Locale</option>
  @foreach($locale as $l)
    <option>{{ $l->tipologia }} {{ $l->classificazione }} {{ $l->numero }}</option>
  @endforeach
</select>
```

Per distinguere le segnalazioni fatte dai semplici utenti da quelle fatte dai docenti abbiamo creato questa funzione, la si può trovare all'interno del controller [SegnalazioneController](#). Ecco un piccolo snippet:

```
$verify_email = M_personale_docente::select('email')
->where('email', '=', $request->get('email'))->first();

if($verify_email == null ){
```

Grazie ad essa ogniquale volta che viene inserita una segnalazione viene fatto un controllo sull'email inserita nel form (obbligatoriamente istituzionale nel caso dei docenti); se quest'ultima è già presente nella tabella PERSONALE DOCENTE viene dato per scontato che si tratta di un docente e viene modificata la foreign key con l'id del docente che ha effettuato la segnalazione, altrimenti la segnalazione risulta essere stata fatta da un utente qualsiasi e viene inserito l'id utente nella foreign key dedicata.

Il metodo che abbiamo deciso di usare per inviare i dati al server è il metodo **POST**.

## 6.2 Mappa.

In alto a destra della nostra Homepage è presente il simbolo di Google Maps, cliccandovi sopra l'utente verrà indirizzato nella seguente pagina:



Questa pagina indica la posizione di tutti i dipartimenti del nostro ateneo, l'utente cliccando su un marcatore potrà vedere il nome del dipartimento e soprattutto il numero totale di problemi segnalati all'interno di esso. Per creare questa pagina abbiamo usato le API di Google Maps, il codice è presente all'interno del file **"mappa.blade.php"**. C'è da fare una precisazione riguardo l'utilizzo di queste API infatti come si può vedere nella foto in alto la mappa risulta oscurata ciò è dovuto al fatto che Google non fornisce più gratuitamente

le proprie API, quindi noi non avendo un indirizzo di fatturazione abbiamo dovuto tenere la mappa in questo modo, dal punto di vista del funzionamento non abbiamo riscontrato nessun tipo di problema.

“In determinate circostanze, potrebbe essere visualizzata una mappa oscurata o un'immagine "negativa" di Street View, contrassegnata dal testo "solo a scopo di sviluppo". Questo comportamento in genere indica problemi con una chiave API o con la fatturazione. Per poter utilizzare i prodotti di Google Maps Platform, la fatturazione deve essere abilitata sul tuo account e tutte le richieste devono includere una chiave API valida.”

(Fonte [developers.google.com](https://developers.google.com/maps/documentation/faq)).

La funzione che ci permette di contare il numero di problemi all'interno di un dipartimento è inclusa all'interno del controller [MappaController](#).

Infine in alto a sinistra è presente l'immagine di una casetta, inserita per fare in modo che l'utente cliccandovi sopra possa ritornare all'Homepage.

## 6.3 Login.

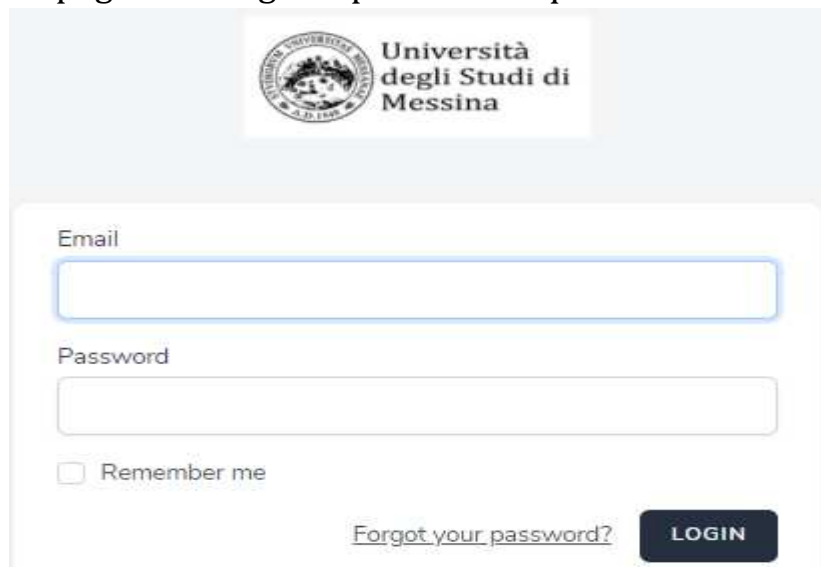
Sempre nell'Homepage in alto a destra è presente il login, esso deve essere utilizzato dai tecnici oppure dai direttori per entrare all'interno delle loro aree riservate. Laravel rende l'implementazione dell'autenticazione molto semplice. In effetti, quasi tutto è configurato in modo automatico. Il file di configurazione dell'autenticazione si trova in [config/auth.php](#), che contiene diverse opzioni ben documentate per modificare il comportamento dei servizi di autenticazione. Il comando che abbiamo usato è il seguente:

```
composer require laravel/jetstream

// Install Jetstream with the Livewire stack...
php artisan jetstream:install livewire
```

Laravel fornisce una session che mantiene lo stato dell'utente autenticato questo grazie alla memorizzazione della sessione e i cookie. Il login fornito da Laravel offre oltre al normale servizio di autenticazione anche delle feature come la modifica del proprio profilo, l'inserimento della foto di profilo e il recupero della password dimenticata che sono stati usati nella nostra applicazione web. Inoltre la grafica del login è stata fatta grazie a Jetstream, uno scaffolding dell'applicazione dal design accattivante. Jetstream è progettato utilizzando Tailwind CSS e offre la possibilità di scegliere tra impalcature Livewire o Inertia nel nostro caso abbiamo optato per Livewire.

La pagina del login si presenta in questo modo:



Jetstream fornisce la tabella **users** in cui sono presenti i dati di tutti coloro che dovranno interagire con il login. Nella tabella nominata in precedenza abbiamo inserito un ulteriore attributo ("utype") utilizzato per distinguere l'accesso dei tecnici (USR) da quello dei direttori (ADM), ecco la tabella:

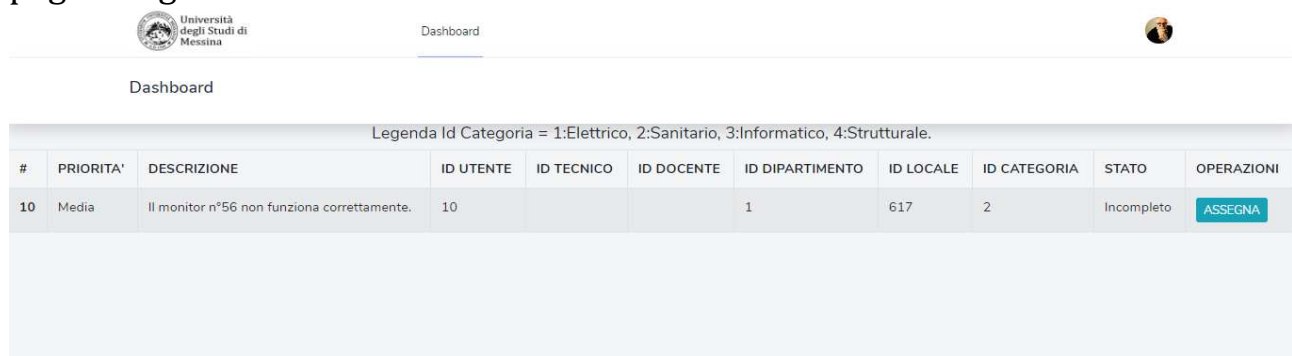
```
class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->foreignId('current_team_id')->nullable();
            $table->text('profile_photo_path')->nullable();
            $table->string('utype')->default('USR')->comment('ADM per Direttori, USR per Tecnici');
            $table->timestamps();
        });
    }
}
```

Inoltre Laravel semplifica la protezione dell'applicazione dagli attacchi CSRF (Cross-Site Request Forgery), esso evita l'esecuzione di comandi non autorizzati per conto di un utente autenticato. Laravel genera automaticamente un "token" CSRF per ogni sessione utente attiva gestita dall'applicazione. Il token `@csrf`

all'interno della nostra applicazione lo si può infatti trovare nel form presente nel file "form-tecnico.blade.php" e in "form-select.blade.php."

## 6.4 Area riservata direttori.

Il direttore una volta inserite le proprie credenziali viene indirizzato nella pagina seguente:

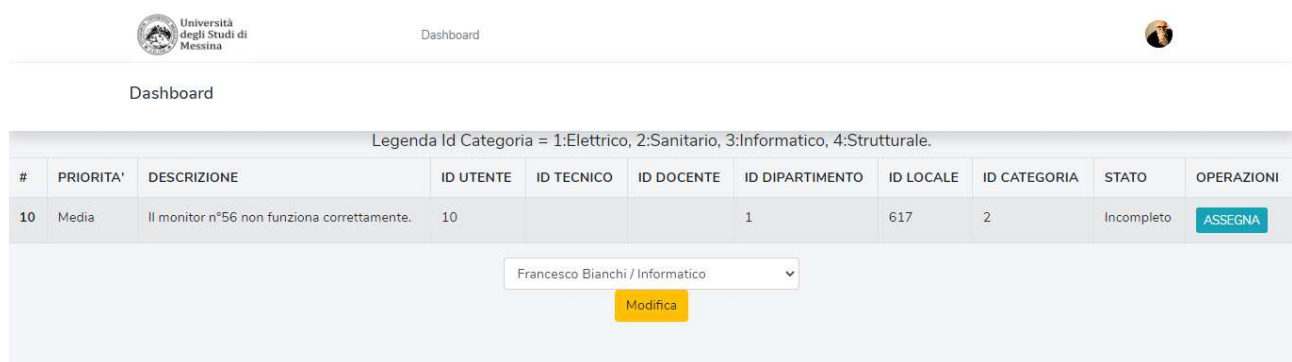


The screenshot shows the Director's Dashboard for the University of Messina. At the top, there is a header with the university logo, the word "Dashboard", and a user profile icon. Below the header, the word "Dashboard" is repeated. A legend indicates the categories: 1: Elettrico, 2: Sanitario, 3: Informatico, 4: Strutturale. The main content is a table with the following data:

#	PRIORITA'	DESCRIZIONE	ID UTENTE	ID TECNICO	ID DOCENTE	ID DIPARTIMENTO	ID LOCALE	ID CATEGORIA	STATO	OPERAZIONI
10	Media	Il monitor n°56 non funziona correttamente.	10			1	617	2	Incompleto	<a href="#">ASSEGNA</a>

Qui il direttore ha la possibilità di visualizzare una tabella che rappresenta in modo dettagliato tutti i problemi che sono stati segnalati dagli utenti e che sono presenti all'interno del suo dipartimento.

Al direttore cliccando su "assegna" apparirà un dropdown menù in cui potrà scegliere il tecnico (facente parte del suo dipartimento) a cui lui potrà assegnare un problema tra quelli presenti nella tabella per procedere così nella risoluzione dello stesso, successivamente la pagina si presenterà così:



The screenshot shows the Director's Dashboard with the same table as before. The "ASSEGNA" button has been clicked, and a dropdown menu is now visible, showing "Francesco Bianchi / Informatico". Below the dropdown is a yellow "Modifica" button.

Una cosa che va detta è che il direttore può assegnare ad un singolo tecnico fino ad un numero di cinque problemi, infatti abbiamo creato una funzione apposita che andrà a contare il numero di problemi assegnati ad ogni tecnico. Nel caso in cui un tecnico abbia raggiunto la soglia massima di problemi assegnati al direttore apparirà un banner che impedirà l'assegnazione di un problema.

## 6.5 Area riservata ai tecnici.

Utilizzando il link “login” in alto a sinistra della nostra Homepage e inserendo le proprie credenziali anche i tecnici avranno la possibilità di entrare nella propria area riservata che si presenterà in questo modo:

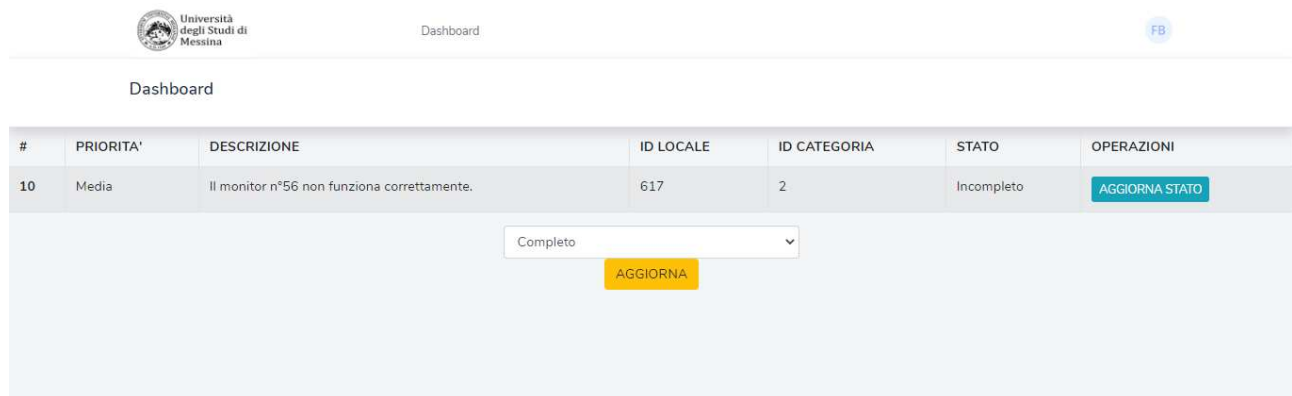


The screenshot shows the top navigation bar with the University of Messina logo, a 'Dashboard' link, and a user profile icon labeled 'FB'. Below this, the word 'Dashboard' is centered. A table with the following columns is displayed: '#', 'PRIORITA'', 'DESCRIZIONE', 'ID LOCALE', 'ID CATEGORIA', 'STATO', and 'OPERAZIONI'. The first row contains the data: '# 10', 'PRIORITA' Media', 'DESCRIZIONE Il monitor n°56 non funziona correttamente.', 'ID LOCALE 617', 'ID CATEGORIA 2', 'STATO Incompleto', and 'OPERAZIONI' with a teal 'AGGIORNA STATO' button.

#	PRIORITA'	DESCRIZIONE	ID LOCALE	ID CATEGORIA	STATO	OPERAZIONI
10	Media	Il monitor n°56 non funziona correttamente.	617	2	Incompleto	AGGIORNA STATO

In questa pagina il tecnico avrà la possibilità di visualizzare in forma tabellare tutte le mansioni, ovvero i problemi da risolvere che gli sono stati assegnati dai direttori. Cliccando su “aggiorna stato” il tecnico troverà un dropdown menù in cui modificherà lo stato di risoluzione del problema, impostato su tre scelte (completo, parzialmente completo e incompleto), così facendo verrà inviata una richiesta al server che aggiornerà il valore del campo **stato** nella tabella PROBLEMA del nostro database e farà in modo che nell’area riservata del direttore il problema appena risolto non sia più visualizzabile.

La pagina dopo aver cliccato su bottone “aggiorna stato” si presenta così:



The screenshot shows the same dashboard as before, but with an additional dropdown menu below the table. The dropdown is currently set to 'Completo' and has a yellow 'AGGIORNA' button below it.

#	PRIORITA'	DESCRIZIONE	ID LOCALE	ID CATEGORIA	STATO	OPERAZIONI
10	Media	Il monitor n°56 non funziona correttamente.	617	2	Incompleto	AGGIORNA STATO

Completo ▼

AGGIORNA



## 7. RESTful API's.

Nella nostra applicazione web abbiamo deciso di utilizzare questo tipo di ragionamento. Infatti, quando si crea un'API, potrebbe essere necessario un livello di trasformazione che si trova tra i modelli Eloquent e le risposte JSON che vengono effettivamente restituite agli utenti dell'applicazione. Le classi di risorse di Laravel ci hanno consentito di trasformare in modo espressivo e semplice i nostri model e le raccolte di modelli in formato JSON.

Per generare una classe di risorse, abbiamo usato il comando:

```
php artisan make:resource ApiResource
```

Per impostazione predefinita, le risorse verranno inserite nella directory `app/Http/Resources` dell'applicazione.

Le ApiResources si presentano in questo modo:

```
class ApiResource extends JsonResponse
{
    /**
     * Transform the resource into an array.
     *
     * @param \Illuminate\Http\Request $request
     * @return array
     */
    public function toArray($request)
    {
        return parent::toArray($request);
    }
}
```

Successivamente alla creazione delle ApiResources abbiamo creato l'API controller mediante l'apposito comando Artisan.

Il controller `ApiSegnalazioneController` che si trova all'interno dell'omonimo file con estensione `php` si presenta in questo modo:

```

class ApiSegnalazioneController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $cat = M_categoria::all();
        $dipa = M_dipartimento::all();
        $locale = M_locale::all();

        return ApiResource::collection([$cat, $dipa, $locale]);
    }
}

```

Qui abbiamo la funzione index essa ha il compito di restituire tutti i valori presenti all'interno del Model. Successivamente grazie all'uso del return tutti i valori dei model in questione vengono restituiti in formato JSON. Come si può vedere dall'immagine abbiamo usato una funzione di Laravel ovvero **"collection"** essa serve per restituire una raccolta di risorse o una risposta impaginata.

Successivamente andiamo ad analizzare funzione store.

In questa funzione per prima cosa andiamo a verificare l'email inserita mediante ([\\$verify\\_email](#)), nel caso in cui non si trovi un riscontro di email all'interno del Model **M\_personale\_docente** la funzione capirà che la segnalazione è stata fatta da un utente quindi oltre a popolare la tabella PROBLEMA inserendo l'id utente all'interno del campo dedicato, verrà popolata anche la tabella UTENTE e tutto ciò viene fatto all'interno del ciclo condizionale if-else.

Nella nostra web app andiamo a prendere i valori di categoria, dipartimento e locale mediante una "select" qui nella creazione delle API abbiamo adottato un meccanismo diverso in quanto Postman non permette di selezionare i dati dal database mediante dropdown menù. Quindi andiamo a selezionare l'id di una categoria, di un dipartimento o di un locale in base a ciò che andremo a scrivere all'interno di value (campo di Postman) nella key (altro campo di Postman) id\_categoria , id\_dipartimento e id\_locale. Vogliamo altresì aggiungere che chiunque in futuro volesse servirsi di questa API per creare una propria applicazione deve essere a conoscenza che dovrà implementare i dropdown menù per inserire le foreign key.

Postman è la piattaforma per lo sviluppo ed il test di API esso consente innanzitutto di costruire molto velocemente richieste HTTP di una certa complessità gestendo per noi l'azione HTTP e tutti i parametri con un'interfaccia

molto chiara e facile da usare. Permette inoltre di visualizzare, i dati in risposta in formato XML, JSON o raw data.

Come si può vedere nelle seguenti immagini la nostra API funziona correttamente infatti viene dato come risposta HTTP il valore 201, significa che i dati passati attraverso l'API vengono salvati nel database.

Di seguito presentiamo un 'immagine in cui è racchiusa la nostra funzione store.

```
/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    $segnalazione = new M_segnalazione();
    $utente = new M_utente();

    $verify_email = M_personale_docente::select('email')
    ->where('email', '=', $request->email)->first();

    if($verify_email == null )
    {
        $utente->nome = $request->nome;
        $utente->cognome = $request->cognome;
        $utente->email = $request->email;
        $utente->numero_di_telefono = $request->numero_di_telefono;
        $segnalazione->priorita = $request->priorita;
        $segnalazione->descrizione = $request->descrizione;
        $salva_utente = $utente->save() && $utente->addres()->save($segnalazione);
    }else{
        $id_docente = M_personale_docente::select('id')
        ->where('email', '=', $request->email )->first();
        $segnalazione->priorita = $request->priorita;
        $segnalazione->descrizione = $request->descrizione;
        $salva_utente = $id_docente->addpersonaledocente()->save($segnalazione);
    }

    $categoria = M_categoria::select('id')
    ->where('classificazione', '=', $request->id_categoria)->first();

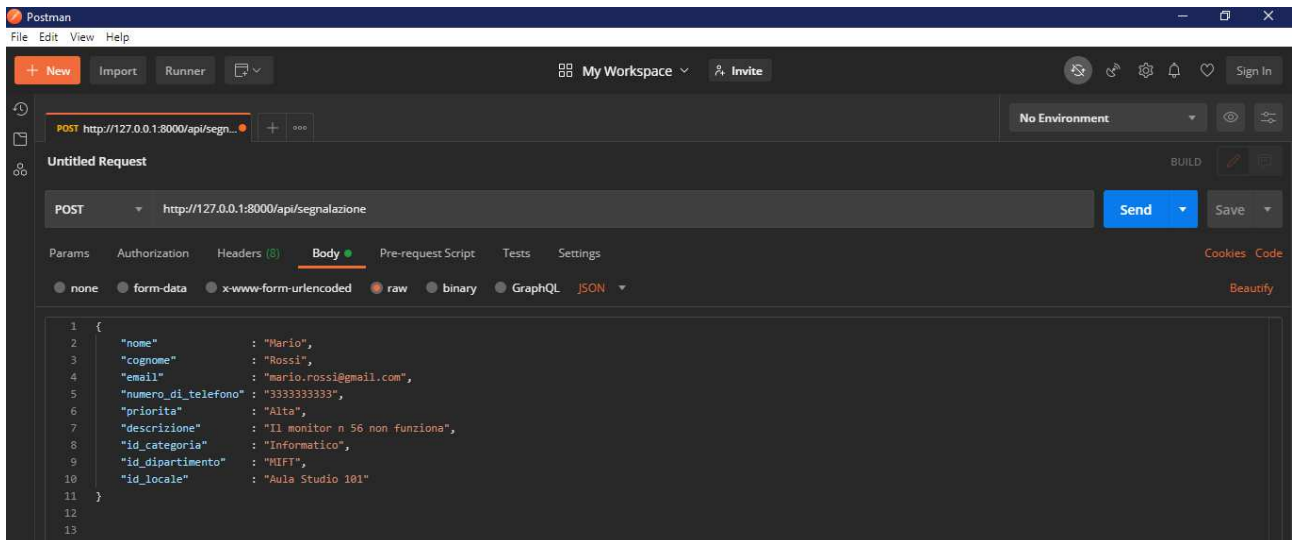
    $dipartimento = M_dipartimento::select('id')
    ->where('nome', '=', $request->id_dipartimento)->first();

    $arr_locale = explode(' ', $request->id_locale);
    if(count($arr_locale) == 2){
        $aula = M_locale::select('id')
        ->where('tipologia', '=', $arr_locale[0], 'and')
        ->where('numero', '=', $arr_locale[1], 'and')
        ->where('id_dipartimento', '=', $dipartimento->id)->first();
    }else{
        $aula = M_locale::select('id')
        ->where('tipologia', '=', $arr_locale[0], 'and')
        ->where('classificazione', '=', $arr_locale[1], 'and')
        ->where('numero', '=', $arr_locale[2], 'and')
        ->where('id_dipartimento', '=', $dipartimento->id)->first();
    }

    if( $segnalazione->save() &&
        $salva_utente &&
        $categoria->addcategory()->save($segnalazione) &&
        $dipartimento->adddipartimento()->save($segnalazione) &&
        $aula->addlocale()->save($segnalazione) )
    {
        return new ApiResource($segnalazione, $utente);
    }
}
```

Invece qui abbiamo inserito le operazioni svolte mediante Postman.

Qui possiamo vedere come avviene l'inserimento della nuova segnalazione tramite API. Tutto ciò che viene inserito verrà inviato e immagazzinato nel nostro database.



Invece qui abbiamo il file di tipo JSON di risposta con i dati inseriti

