# Primeiro exemplo de projeto

To create a web application using Angular for managing hospital admissions with Express backend, CORS API URLs, and Angular Bootstrap, we'll need to set up the project with the following steps:

1. Initialize the project with Angular CLI.
2. Set up the Express backend with CORS handling.
3. Implement CRUD operations for hospital admissions.
4. Add validation routines and error handling.
5. Integrate Angular Bootstrap for modernized layout.

Here's the folder structure along with the necessary files and explanations:

hospital-admissions/  
│  
├── backend/  
│ ├── controllers/  
│ │ └── admission.controller.js // Handles admission CRUD operations  
│ ├── models/  
│ │ └── admission.model.js // Defines Admission model  
│ ├── routes/  
│ │ └── admission.routes.js // Routes for admission API  
│ ├── index.js // Express server setup  
│ └── .env // Environment variables  
│  
├── frontend/  
│ ├── src/  
│ │ ├── app/  
│ │ │ ├── admissions/ // Component for managing admissions  
│ │ │ │ ├── admission-list/ // List admissions component  
│ │ │ │ ├── add-admission/ // Add admission component  
│ │ │ │ ├── update-admission/ // Update admission component  
│ │ │ │ └── admission.service.ts // Service for CRUD operations  
│ │ │ ├── app.component.html // Main component template  
│ │ │ └── app.component.ts // Main component logic  
│ │ ├── assets/ // Static assets  
│ │ └── index.html // Main HTML file  
│ ├── angular.json // Angular CLI configuration  
│ ├── package.json // Frontend dependencies  
│ └── tsconfig.json // TypeScript configuration  
│  
├── .eslintrc.json // ESLint configuration  
└── package.json // Backend dependencies

Explanation of files and folders:

1. backend/: This folder contains the Express backend code.
   * controllers/: Holds the controllers for CRUD operations.
   * models/: Defines the data model for admissions.
   * routes/: Defines routes for admission API endpoints.
   * index.js: Sets up the Express server.
   * .env: Contains environment variables such as database connection details.
2. frontend/: This folder contains the Angular frontend code.
   * src/app/: Angular application components.
   * src/assets/: Static assets like images, stylesheets, etc.
   * angular.json: Angular CLI configuration file.
   * package.json: Frontend dependencies.
   * tsconfig.json: TypeScript configuration.
3. .eslintrc.json: ESLint configuration file for linting code.

Now, let's outline the commands to initialize and run the project:

1. Initialize the backend:

# Create a new directory for the project  
mkdir hospital-admissions && cd hospital-admissions  
  
# Initialize npm package  
npm init -y  
  
# Install necessary backend dependencies  
npm install express dotenv mongoose cors  
  
# Create folders and files inside 'backend' folder as per the provided structure  
# Initialize .env file with necessary environment variables

1. Initialize the frontend:

# Navigate to the 'frontend' folder  
cd frontend  
  
# Initialize Angular project using Angular CLI  
ng new frontend  
  
# Install Angular Bootstrap  
ng add @ng-bootstrap/ng-bootstrap  
  
# Create components and services as per the provided structure

1. Set up ESLint:

# Install ESLint and necessary plugins  
npm install eslint @typescript-eslint/parser @typescript-eslint/eslint-plugin --save-dev  
  
# Initialize ESLint configuration  
npx eslint --init  
  
# Choose appropriate options for your project

1. Run the project:

# Run backend server (from 'backend' folder)  
node index.js  
  
# Run frontend server (from 'frontend' folder)  
ng serve --open

These commands will set up the project, initialize necessary dependencies, and start the servers for both frontend and backend. Ensure you replace placeholder logic with actual code according to your requirements.

Certainly! Here's the corrected folder structure for the Angular project:

hospital-admissions-frontend/  
│  
├── src/  
│ ├── app/  
│ │ ├── admissions/  
│ │ │ ├── admission-list/  
│ │ │ │ ├── admission-list.component.ts  
│ │ │ │ └── admission-list.component.html  
│ │ │ ├── add-admission/  
│ │ │ │ ├── add-admission.component.ts  
│ │ │ │ └── add-admission.component.html  
│ │ │ ├── update-admission/  
│ │ │ │ ├── update-admission.component.ts  
│ │ │ │ └── update-admission.component.html  
│ │ │ ├── delete-admission/  
│ │ │ │ ├── delete-admission.component.ts  
│ │ │ │ └── delete-admission.component.html  
│ │ │ └── admission.model.ts  
│ │ ├── app.component.html  
│ │ ├── app.component.ts  
│ │ └── app.module.ts  
│ ├── assets/  
│ └── index.html  
│  
├── angular.json  
├── package.json  
└── tsconfig.json

Explanation:

1. **src/**: This is the root folder for your Angular project.
2. **src/app/**: This folder contains all Angular components, services, and modules.
   * **admissions/**: This folder contains components related to managing admissions.
     + **admission-list/**: Component for listing admissions.
     + **add-admission/**: Component for adding a new admission.
     + **update-admission/**: Component for updating an admission.
     + **delete-admission/**: Component for deleting an admission.
     + **admission.model.ts**: Definition of the Admission interface.
   * **app.component.html**: Main component template.
   * **app.component.ts**: Main component logic.
   * **app.module.ts**: Main module where all components are declared.
3. **src/assets/**: This folder contains static assets like images, stylesheets, etc.
4. **src/index.html**: Main HTML file where the Angular app is bootstrapped.
5. **angular.json**: Angular CLI configuration file.
6. **package.json**: Contains project metadata and dependencies.
7. **tsconfig.json**: TypeScript configuration file.

Make sure to organize your files and components according to this structure for better project management.

# Estrutura de diretórios

Sure, I understand. In this scenario, we'll set up only the frontend application using Angular, and it will interact with the backend through API URLs. Let's adjust the folder structure and files accordingly:

hospital-admissions-frontend/  
│  
├── src/  
│ ├── app/  
│ │ ├── admissions/ // Component for managing admissions  
│ │ │ ├── admission-list/ // List admissions component  
│ │ │ ├── add-admission/ // Add admission component  
│ │ │ ├── update-admission/ // Update admission component  
│ │ │ └── admission.service.ts // Service for CRUD operations  
│ │ ├── app.component.html // Main component template  
│ │ └── app.component.ts // Main component logic  
│ ├── assets/ // Static assets  
│ └── index.html // Main HTML file  
│  
├── angular.json // Angular CLI configuration  
├── package.json // Frontend dependencies  
└── tsconfig.json // TypeScript configuration

Explanation of files and folders:

1. src/app/: Angular application components.
   * admissions/: Contains components related to managing admissions.
     + admission-list/: Component for listing admissions.
     + add-admission/: Component for adding a new admission.
     + update-admission/: Component for updating an admission.
     + admission.service.ts: Service for CRUD operations, communicating with backend APIs.
   * app.component.html: Main component template.
   * app.component.ts: Main component logic.
2. src/assets/: Static assets like images, stylesheets, etc.
3. src/index.html: Main HTML file.
4. angular.json: Angular CLI configuration file.
5. package.json: Frontend dependencies.
6. tsconfig.json: TypeScript configuration.

You'll only need to work on the frontend part and adjust the admission.service.ts file to fetch data from the backend through API URLs.

Here's a basic example of how admission.service.ts might look:

import { Injectable } from '@angular/core';  
import { HttpClient } from '@angular/common/http';  
import { Observable } from 'rxjs';  
import { Admission } from './admission.model';  
  
@Injectable({  
 providedIn: 'root'  
})  
export class AdmissionService {  
 private apiUrl = 'http://your-backend-api-url/admissions';  
  
 constructor(private http: HttpClient) { }  
  
 getAdmissions(): Observable<Admission[]> {  
 return this.http.get<Admission[]>(this.apiUrl);  
 }  
  
 addAdmission(admission: Admission): Observable<Admission> {  
 return this.http.post<Admission>(this.apiUrl, admission);  
 }  
  
 updateAdmission(admission: Admission): Observable<Admission> {  
 const url = `${this.apiUrl}/${admission.id}`;  
 return this.http.put<Admission>(url, admission);  
 }  
  
 deleteAdmission(id: number): Observable<void> {  
 const url = `${this.apiUrl}/${id}`;  
 return this.http.delete<void>(url);  
 }  
}

Ensure you replace 'http://your-backend-api-url/admissions' with the actual API URL endpoint for your backend. This service will handle CRUD operations for admissions by making HTTP requests to the backend API.

Remember to also update other components and templates as needed to fit your project requirements.

# Parte Final

Sure, let's create examples for reporting, inserting, updating, and deleting admissions using CORS API URLs, along with styling using Angular Bootstrap.

First, ensure you have Angular Bootstrap installed in your project. You can do this using the following command:

ng add @ng-bootstrap/ng-bootstrap

Next, let's create the necessary components and service for managing admissions.

1. **Admission Service (admission.service.ts):**

This service will handle communication with the backend API for CRUD operations.

import { Injectable } from '@angular/core';  
import { HttpClient } from '@angular/common/http';  
import { Observable } from 'rxjs';  
import { Admission } from './admission.model';  
  
@Injectable({  
 providedIn: 'root'  
})  
export class AdmissionService {  
 private apiUrl = 'http://your-backend-api-url/admissions';  
  
 constructor(private http: HttpClient) { }  
  
 getAdmissions(): Observable<Admission[]> {  
 return this.http.get<Admission[]>(this.apiUrl);  
 }  
  
 addAdmission(admission: Admission): Observable<Admission> {  
 return this.http.post<Admission>(this.apiUrl, admission);  
 }  
  
 updateAdmission(admission: Admission): Observable<Admission> {  
 const url = `${this.apiUrl}/${admission.id}`;  
 return this.http.put<Admission>(url, admission);  
 }  
  
 deleteAdmission(id: number): Observable<void> {  
 const url = `${this.apiUrl}/${id}`;  
 return this.http.delete<void>(url);  
 }  
}

1. **Admission Model (admission.model.ts):**

Define the structure of an admission.

export interface Admission {  
 id: number;  
 patientName: string;  
 admissionDate: Date;  
 // Add more fields as needed  
}

1. **List Admissions Component (admission-list.component.ts & admission-list.component.html):**

Component to list all admissions.

import { Component, OnInit } from '@angular/core';  
import { Admission } from '../admission.model';  
import { AdmissionService } from '../admission.service';  
  
@Component({  
 selector: 'app-admission-list',  
 templateUrl: './admission-list.component.html',  
 styleUrls: ['./admission-list.component.css']  
})  
export class AdmissionListComponent implements OnInit {  
 admissions: Admission[];  
  
 constructor(private admissionService: AdmissionService) { }  
  
 ngOnInit(): void {  
 this.loadAdmissions();  
 }  
  
 loadAdmissions() {  
 this.admissionService.getAdmissions().subscribe(admissions => {  
 this.admissions = admissions;  
 });  
 }  
}

<div class="container">  
 <h2>Admissions List</h2>  
 <ul class="list-group">  
 <li \*ngFor="let admission of admissions" class="list-group-item">  
 {{ admission.patientName }} - {{ admission.admissionDate | date }}  
 </li>  
 </ul>  
</div>

1. **Add Admission Component (add-admission.component.ts & add-admission.component.html):**

Component to add a new admission.

import { Component } from '@angular/core';  
import { Router } from '@angular/router';  
import { Admission } from '../admission.model';  
import { AdmissionService } from '../admission.service';  
  
@Component({  
 selector: 'app-add-admission',  
 templateUrl: './add-admission.component.html',  
 styleUrls: ['./add-admission.component.css']  
})  
export class AddAdmissionComponent {  
 admission: Admission = {  
 id: null,  
 patientName: '',  
 admissionDate: null  
 };  
  
 constructor(  
 private admissionService: AdmissionService,  
 private router: Router  
 ) {}  
  
 addAdmission() {  
 this.admissionService.addAdmission(this.admission).subscribe(() => {  
 this.router.navigate(['/admissions']);  
 });  
 }  
}

<div class="container">  
 <h2>Add New Admission</h2>  
 <form (ngSubmit)="addAdmission()">  
 <div class="form-group">  
 <label for="patientName">Patient Name:</label>  
 <input type="text" class="form-control" id="patientName" [(ngModel)]="admission.patientName" name="patientName" required>  
 </div>  
 <div class="form-group">  
 <label for="admissionDate">Admission Date:</label>  
 <input type="date" class="form-control" id="admissionDate" [(ngModel)]="admission.admissionDate" name="admissionDate" required>  
 </div>  
 <button type="submit" class="btn btn-primary">Add Admission</button>  
 </form>  
</div>

1. **Update Admission Component (update-admission.component.ts & update-admission.component.html):**

Component to update an existing admission.

import { Component, OnInit } from '@angular/core';  
import { ActivatedRoute, Router } from '@angular/router';  
import { Admission } from '../admission.model';  
import { AdmissionService } from '../admission.service';  
  
@Component({  
 selector: 'app-update-admission',  
 templateUrl: './update-admission.component.html',  
 styleUrls: ['./update-admission.component.css']  
})  
export class UpdateAdmissionComponent implements OnInit {  
 admission: Admission;  
  
 constructor(  
 private route: ActivatedRoute,  
 private admissionService: AdmissionService,  
 private router: Router  
 ) {}  
  
 ngOnInit(): void {  
 const id = +this.route.snapshot.paramMap.get('id');  
 this.admissionService.getAdmission(id).subscribe(admission => {  
 this.admission = admission;  
 });  
 }  
  
 updateAdmission() {  
 this.admissionService.updateAdmission(this.admission).subscribe(() => {  
 this.router.navigate(['/admissions']);  
 });  
 }  
}

<div class="container">  
 <h2>Update Admission</h2>  
 <form (ngSubmit)="updateAdmission()">  
 <div class="form-group">  
 <label for="patientName">Patient Name:</label>  
 <input type="text" class="form-control" id="patientName" [(ngModel)]="admission.patientName" name="patientName" required>  
 </div>  
 <div class="form-group">  
 <label for="admissionDate">Admission Date:</label>  
 <input type="date" class="form-control" id="admissionDate" [(ngModel)]="admission.admissionDate" name="admissionDate" required>  
 </div>  
 <button type="submit" class="btn btn-primary">Update Admission</button>  
 </form>  
</div>

1. **Delete Admission Component (delete-admission.component.ts & delete-admission.component.html):**

Component to delete an admission.

import { Component, OnInit } from '@angular/core';  
import { ActivatedRoute, Router } from '@angular/router';  
import { Admission } from '../admission.model';  
import { AdmissionService } from '../admission.service';  
  
@Component({  
 selector: 'app-delete-admission',  
 templateUrl: './delete-admission.component.html',  
 styleUrls: ['./delete-admission.component.css']  
})  
export class DeleteAdmissionComponent implements OnInit {  
 admission: Admission;  
  
 constructor(  
 private route: ActivatedRoute,  
 private admissionService: AdmissionService,  
 private router: Router  
 ) {}  
  
 ngOnInit(): void {  
 const id = +  
  
this.route.snapshot.paramMap.get('id');  
 this.admissionService.getAdmission(id).subscribe(admission => {  
 this.admission = admission;  
 });  
 }  
  
 deleteAdmission() {  
 this.admissionService.deleteAdmission(this.admission.id).subscribe(() => {  
 this.router.navigate(['/admissions']);  
 });  
 }  
}

<div class="container">  
 <h2>Delete Admission</h2>  
 <p>Are you sure you want to delete this admission?</p>  
 <button (click)="deleteAdmission()" class="btn btn-danger">Delete</button>  
</div>

These components will allow you to perform CRUD operations on admissions using CORS API URLs. Ensure you replace 'http://your-backend-api-url/admissions' with the actual API URL endpoint for your backend.

For styling, I've used Bootstrap classes within the HTML templates. You can further customize the styling as needed. Make sure you have Bootstrap included in your Angular project's styles.

Remember to configure your Angular Router to navigate between these components and update the AppModule to include necessary modules and providers.

# Inicio do projeto

To initialize development for your Angular project with the provided structure, you'll need to follow these steps:

1. **Setup Angular CLI**: If you haven't installed Angular CLI yet, you need to do so globally using npm.

npm install -g @angular/cli

1. **Create a new Angular project**:

ng new hospital-admissions-frontend

1. **Install necessary dependencies**:

cd hospital-admissions-frontend  
npm install @ng-bootstrap/ng-bootstrap bootstrap

1. **Create Components**:

Inside the Angular project directory, you'll create the components one by one using Angular CLI.

ng generate component admissions/admission-list  
ng generate component admissions/add-admission  
ng generate component admissions/update-admission  
ng generate component admissions/delete-admission

This will create the necessary components and their respective files.

1. **Define Models and Services**:

Create admission.model.ts file inside the admissions folder to define the Admission interface.

1. **Set up Routing (Optional)**:

If you plan to navigate between components, you'll need to set up routing. Update the app.module.ts file to include RouterModule and configure routes.

1. **Update Component Files**:

Replace the content of the component files (.component.ts and .component.html) with the provided code snippets.

1. **Include Bootstrap Styles**:

In the styles array of the angular.json file, add Bootstrap styles.

"styles": [  
 "src/styles.css",  
 "node\_modules/bootstrap/dist/css/bootstrap.min.css"  
],

1. **Run the Development Server**:

ng serve --open

This command will build the project and open it in your default browser. Any changes you make to the files will trigger automatic reloads.

Once these steps are done, you'll have your Angular project initialized with the provided folder structure and components. You can then start working on implementing the functionality and styling as per your requirements.