Angular .Net Core Web api
ARCHITECTURE CRUD

Call Service
Important Notions
API Call
Folder Name
File Name
Method terms ans return
Sub-Folder Name
Image Process

FrontEnd

Services

Api MODEL

Student.service.ts

```
getStudents(): Observable <Student[]>{
  return  this.httpClient.get<Student[]>
(this.baseApiUrl+'/students');
 }
//************************************
getStudent(studentId:string): Observable
<Student>{
  return  this.httpClient.get<Student>
(this.baseApiUrl+'/students/' + studentId);
 }
```

```
updateStudent(studentId:string,StudentRequest:St
udent):Observable<Student>{
const
UpdateStudentRequest:UpdateStudentRequest={
    firstname:StudentRequest.firstname,
    lastname:StudentRequest.lastname,
postalAddress:StudentRequest.adress.postalAddre
ss
}
  return  this.httpClient.put<Student>
(this.baseApiUrl +'/students/' +
studentId,UpdateStudentRequest)
 }
```

```
uploadImage(stud
  :Observable<any>
  const formData=
  formData.appen
  return this.http
  (this.baseApiUrl
  +'/upload-image'
  {responseType:'t

 }
//***************
getImagePath(rela
  return this.base

 }
```

app

appcomponent.ts

```
@Component({
selector: 'app-root',
templateUrl:
'./app.component.html',
styleUrls: ['./app.component.css']
})
```

appcomponent.html

```
router outlet here
Main template here
<app-top-nav></app-top-nav>
<router-outlet></router-outlet>
```

app.module.ts

```
imports: [
   FormsModule,
   BrowserModule,
HttpClientModule],
schemas:[
   CUSTOM_ELEMENTS_SCHEMA
 ]
```

assets

Components

GridStudent

UI MODEL

studentGrid.component.ts

```
constructor( private studentService:StudentService) {}
displayedColumns: string[] =['firstname', 'lastname',
'dateOfBirth','email','mobile','gender','edit'];
dataSource:MatTableDataSource<Student> =new
MatTableDataSource <Student>();
@ViewChild(MatPaginator) matPaginator!:MatPaginator;
@ViewChild(MatSort) matSort!:MatSort;
this.studentService.getStudents()
.subscribe({
  next:(successResponse)=>
 {
    this.students=successResponse;
    this.dataSource= new MatTableDataSource<Student>
(this.students);
     if(this.matPaginator){
     this.dataSource.paginator=this.matPaginator; }
     if(this.matSort){
     this.dataSource.sort=this.matSort;}
 },
 error:(errorResponse)=>{
   console.log(errorResponse);}
}
);
```

studentGrid.component.html

```
<table mat-table matSort [dataSource]="dataSource" class="">

 <!-- FirstName Column -->
 <ng-container matColumnDef="firstname">
  <th mat-header-cell *matHeaderCellDef mat-sort-header> firstName
</th>
  <td mat-cell *matCellDef="let element"> {{element.firstname}} </td>
 </ng-container>

 <!-- LastName Column -->
 <ng-container matColumnDef="lastname">
  <th mat-header-cell *matHeaderCellDef mat-sort-header> lastName
</th>
  <td mat-cell *matCellDef="let element"> {{element.lastname}} </td>
 </ng-container>

   <!-- gender Column -->
   <ng-container matColumnDef="gender">
    <th mat-header-cell *matHeaderCellDef> gender </th>
    <td mat-cell *matCellDef="let element">
{{element.gender.description}} </td>
   </ng-container>
   <!-- edit Column -->
   <ng-container matColumnDef="edit">
     <th mat-header-cell *matHeaderCellDef>  </th>
     <td mat-cell *matCellDef="let element">
       <a [routerLink]="['students',element.id]">
         <mat-icon color="primary">edit</mat-icon>
       </a>
     </td>
   </ng-container>
<tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
  <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
</table>
<mat-paginator [length]="100"
[pageSize]="5"
[pageSizeOptions]="[5, 10, 25, 100]"
aria-label="Select page">
</mat-paginator>
```

View-Student

ViewStudent.component.html

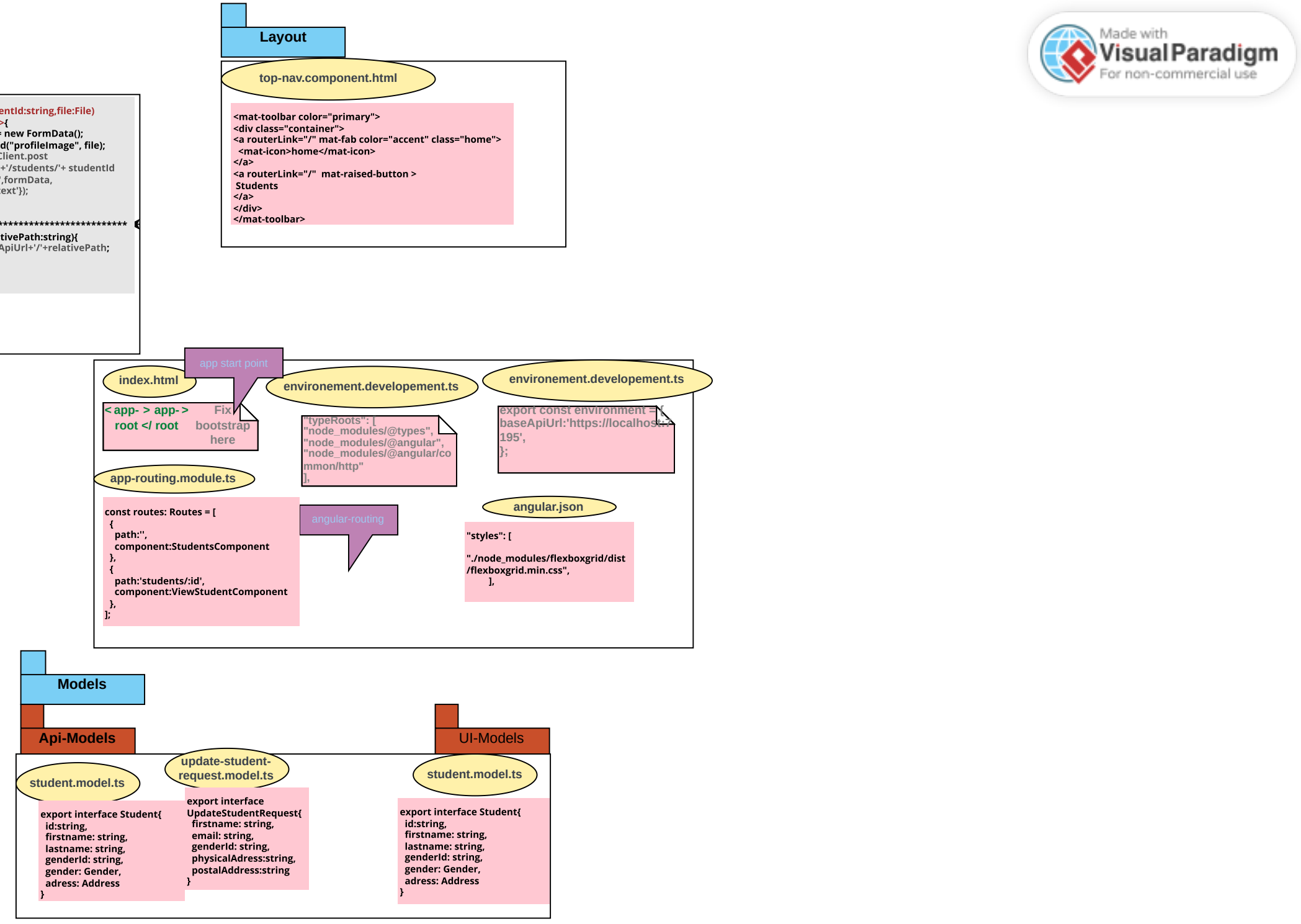
```
<div class="row center-xs">

    <div class="image-container">
      <img [src]="displayProfileImageUrl"
     alt="Profile Image">
    </div>

   </div>
//*******************************
<div class="col-xs-12 row center-xs mt-1"
*ngIf="!isNewStudent" >
      <input hidden  type="file"
#imageUpload
       (change)="uploadImage($event)" >
      <button mat-raised-button
color="primary"
      (click)="imageUpload.click()">Change
Image</button>
</div>
//*************************************
 <mat-form-field appearance="outline">
        <mat-label>First Name</mat-label>
        <input type="text" matInput
        [(ngModel)]="student.firstname"
name="firstname" required>
        <mat-error> First Name is a required
field</mat-error>
       </mat-form-field>
```

View-Student.component.ts

```
displayProfileImageUrl='';
@ViewChild('studentDetailsForm') studentDetailsForm?:
NgForm;
ngOnInit(): void {
this.route.paramMap.subscribe(
     (params )=>{
this.studentId= params.get('id');

if(this.studentId.toLowerCase()==='Add'.toLowerCase()){
          // -> new Student Functionnality
          this.setImage();
          } else{
           // -> Existing Student Functionnality
          this.studentService.getStudent(this.studentId)
           .subscribe(
             (successResponse)=>
{this.student=successResponse;
              this.setImage();
              },
              (errorResponse)=>{this.setImage()});
          }
});
}//************************************************
private setImage(): void{
if(this.student.profileImageUrl){
     //Fecth the Image by url
     this.displayProfileImageUrl=
this.studentService.getImagePath(this.student.profileIma
geUrl) ;
}
     else{
     //Display a default
    this.displayProfileImageUrl='/assets/user.png';
   }
}
```

```
uploadImage(event:
any):void{
   if(this.studentId){
    const file:File=
event.target.files[0];

this.studentService.
uploadImage(this.st
udent.id,file)
    .subscribe(

(successResponse)=>
{

this.student.profileI
mageUrl=successRes
ponse;
      this.setImage();

this.snackbar.open('
Profile Image
updated ',
       undefined,
{duration:2000});
      },

(errorResponse)=>{}
    );
   }

  }
```

Routing



Comments

Method Name and type return

Folder Name

Code related about image process

Sub-Folder Name

C# File Name

BackEnd

Controller

StudentController

```
[HttpGet]
[Route("[controller]")]
public async Task<IActionResult> GetAllStudents()
{
var students = await
StudentRepository.GetStudentsAsync();
return Ok(mapper.Map<List<Student>>(students));
}
//*************************************************
[HttpGet]
[Route("[controller]/{studentId:guid}"),
ActionName("GetStudentAsync")]
public async Task<IActionResult>
GetStudentAsync([FromRoute] Guid studentId)
{
    //Fetch Student Details
    var student = await
    StudentRepository.GetStudentAsync(studentId);
    //Return Student
        if (student == null)
        {
        return NotFound();
        }
return Ok(mapper.Map<Student>(student));
}
```

```
[HttpPut]
[Route("[controller]/{studentId:guid}")]
public async Task<IActionResult>
UpdateStudentAsync
([FromRoute] Guid studentId, [FromBody]
UpdateStudentRequest request){
if (await StudentRepository.Exists(studentId))
{
    var studentDataModel = new Student();
    // Update Details
    var updatedStudent=await
    StudentRepository.UpdateStudent
    (studentId,mapper.Map<DataModels.Student>
    (request));
        if (updatedStudent != null)
        {
        return Ok(mapper.Map
        <Student>    (updatedStudent));
        }
}
return NotFound();
}
```

```
[HttpPost]
[Route("[controller]/{studentId:guid}/upload-image")]
public async Task<IActionResult> UploadImage([FromRoute] Guid studentId,IFormFile profileImage){
//Check if student exist
var validExtensions = new List<string>{ ".jpeg",".png",".gif",".jpg"};
if (profileImage != null&& profileImage.Length >0 ){
    var extension = Path.GetExtension(profileImage.FileName);
    if (validExtensions.Contains(extension)){ if (await StudentRepository.Exists(studentId)){
    var fileName = Guid.NewGuid() +Path.GetExtension(profileImage.FileName);
    //upload the image to local storage
    var fileImagePath = await imageRepository.Upload(profileImage, fileName);
    if (await StudentRepository.UpdateProfileImage(studentId, fileImagePath)){return Ok(fileImagePath);}
    return StatusCode(StatusCodes.Status500InternalServerError, "Error uploading image");
    }
    }
return BadRequest("this is not a valid image format");
}
return NotFound();

}
```

DataModels

Student.cs

```
public class Student
{
public Guid Id {get;set;}
public string? Firstname {get;set;}
//Navigation Properties
public Gender? Gender { get; set; }
}
```

StudentAdminContext

```
public class StudentAdminContext: DbContext
{
public
StudentAdminContext(DbContextOptions<StudentAd
minContext> options)
: base(options)
{}
public DbSet<Student> Student { get; set; }
public DbSet<Gender> Gender { get; set; }
public DbSet<Adress> Adress { get; set; }
}
```

Program.cs

```
Fix Db Context here
Fix  AddScoped here
Fix AddAutoMapper here
Fix Corse Policy Here
builder.Services.AddScoped<IImageRepository,
LocalStorageImageRepository>();

app.UseStaticFiles(new StaticFileOptions
{
FileProvider = new
PhysicalFileProvider(Path.Combine(app.Environmen
t.ContentRootPath. "Ressources")),
RequestPath = "/Ressources"
});

Fix Fluent Validation here
```

appSettings.json

```
Fix Connection String here
```

Repository

IStudentRepository

```
public interface
IStudentRepository
{
Task<List<Student>>
GetStudentsAsync();
Task<Student>
GetStudentAsync(Guid studentId);
Task<Boolean> Exists(Guid
studentId);
Task<Student>
UpdateStudent(Guid
studentId.Student request);
Task<bool>
UpdateProfileImage(Guid
studentId,string profileImageUrl);


}
```

SqlStudentRepository

```
public async Task<List<Student>>
GetStudentsAsync()
{
return await
context.Student.Include(nameof(Gender))
.Include(nameof(Adress)).ToListAsync();
}
//*************************************
public async Task<Student> GetStudentAsync(
Guid StudentId)
{
return await context.Student
.Include(nameof(Gender))
.Include(nameof(Adress))
.FirstOrDefaultAsync(x =>x.Id==StudentId);
}
```

```
public async Task<bool> Exists(Guid
{
return await context.Student.AnyAs
studentId);
}
//*************************************
public async  Task<Student> Update
studentId, Student request){
var existingStudent = await
GetStudentAsync(studentId);
if (existingStudent != null)
{
existingStudent.Firstname = reques
existingStudent.Lastname = request
existingStudent.DateOfBirth = requ
existingStudent.Email = request.Ema
existingStudent.Mobile = request.M
existingStudent.GenderId = request.
existingStudent.Adress.PhysicalAdr
request.Adress.PhysicalAdress;
existingStudent.Adress.PostalAddre
request.Adress.PostalAddress;
await context.SaveChangesAsync();
return existingStudent;
}
return null;

}
```

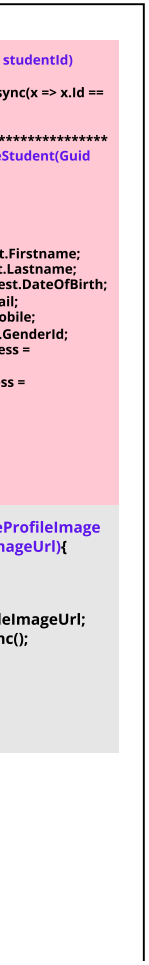
```
public async Task<bool> Update
(Guid studentId, string profileIm
var student = await
GetStudentAsync(studentId);
if (student != null){
student.ProfileImageUrl = profil
await context.SaveChangesAsyn
return true;
}
return false;
}
```

IImageRepository

```
public interface IImageRepository
  {
    Task<string> Upload(IFormFile file, string fileName);
  }
```

LocalStorageImageRepository

```
public async Task<string> Upload(IFormFile file, string fileName){
var filePath=Path.Combine(Directory.GetCurrentDirectory(),
@"Ressources\Images",fileName);
using Stream fileStream = new FileStream
(filePath, FileMode.Create);
await file.CopyToAsync(fileStream);
return GetServerRelativePath(fileName);
}//*****************************************************************
private string GetServerRelativePath(string fileName){
return Path.Combine(@"Ressources\Images", fileName);
}
```



Profiles

AutoMapperProfiles

```
public AutoMapperProfiles()
{
CreateMap<DataModels.Student, Student>()
.ReverseMap();
CreateMap<DataModels.Gender, Gender>()
.ReverseMap();
CreateMap<UpdateStudentRequest,
DataModels.Student>()
.AfterMap<UpdateStudentRequestAfterMap>();
}
```

AfterMaps

UpdateStudentRequestAfterMap

```
public class UpdateStudentRequestAfterMap :
IMappingAction<UpdateStudentRequest,
DataModels.Student>{
public void Process(UpdateStudentRequest source,
DataModels.Student destination, ResolutionContext
context){
destination.Adress = new DataModels.Adress(){
PhysicalAdress = source.PhysicalAdress,
PostalAddress = source.PostalAddress
};
}
}
```

DomainModels

UpdateStudentRequest.cs

```
public class UpdateStudentRequest
{
public string? Firstname { get; set; }
public string? Lastname { get; set; }
public string? DateOfBirth { get; set; }
public string? Email { get; set; }
public long Mobile { get; set; }
public Guid GenderId { get; set; }
public string? PhysicalAdress { get; set; }
public string? PostalAddress { get; set; }

}
```

AddStudentR

```
public class AddStudentRequest
{
public string? Firstname { get; set; }
public string? Lastname { get; set; }
public string? DateOfBirth { get; set; }
public string? Email { get; set; }
public long Mobile { get; set; }
public Guid GenderId { get; set; }
public string? PhysicalAdress { get; set; }
public string? PostalAddress { get; set; }
}
```


Validators

AddStudentRequestValidator

```
 public AddStudentRequestValidator(IStudentRepository
studentRepository)
      {
RuleFor(x => x.Firstname).NotEmpty();
RuleFor(x => x.Email).NotEmpty().EmailAddress();
RuleFor(x =>
x.Mobile).NotEmpty().GreaterThan(99999).LessThan(10000000000);
RuleFor(x => x.GenderId).NotEmpty().Must(id =>
{
var gender = studentRepository
.GetGendersAsync().Result.ToList()
.FirstOrDefault(x => x.Id == id);
if (gender != null)
{
return true;
}
return false;

}).WithMessage("Please select a valid Gender");
RuleFor(x => x.PhysicalAdress).NotEmpty();
RuleFor(x => x.PostalAddress).NotEmpty();
}
```