

# Projeto de Interfaces WEB

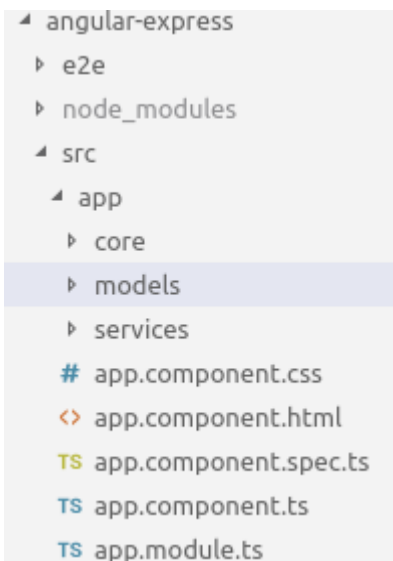
Angular e Express  
Aula 09

# Introdução

- O objetivo desse mini-projeto é:
  - Criar uma aplicação simples em Angular que insere dados em um servidor e também pega dados do servidor (um set e get)
  - Criar um servidor express simples para tratar as requisições do Angular.

# Projeto Angular

- Crie o projeto Angular:
  - A pasta core é um módulo!
  - As pastas models e services são pastas comuns.



```
angular-express
├── e2e
├── node_modules
├── src
│   ├── app
│   │   ├── core
│   │   ├── models
│   │   └── services
│   ├── app.component.css
│   ├── app.component.html
│   ├── app.component.spec.ts
│   ├── app.component.ts
│   └── app.module.ts
```

The image shows a file explorer view of an Angular project. The root directory is 'angular-express'. It contains subdirectories 'e2e', 'node\_modules', and 'src'. The 'src' directory contains 'app', 'models', and 'services'. The 'app' directory contains 'core', 'models', and 'services'. The 'models' and 'services' directories are highlighted in blue. Below the 'app' directory, there are several files: 'app.component.css', 'app.component.html', 'app.component.spec.ts', 'app.component.ts', and 'app.module.ts'. Each file has a small icon to its left: a hash for CSS, a code icon for HTML, a TypeScript icon for spec files, and a TypeScript icon for other TypeScript files.

# A pasta models

- Crie o arquivo User.ts

```
export class User{  
  firstName:string;  
  lastName:string;  
  email:string;  
  zipCode:string;  
  password:string;  
}
```

*Uma classe simples que representa um usuário. Facilita a troca de dados entre o Angular e o Express.*

# A pasta service

- Crie o arquivo user.service.js

```
import { User } from '../models/User';  
import { Injectable, OnDestroy } from '@angular/core';  
import { HttpClient } from '@angular/common/http';
```

```
@Injectable({  
  providedIn: 'root'  
})  
export class UserService{
```

```
  constructor(private http:HttpClient) {  
  }
```

```
  setUser(user:User){
```

```
    return this.http.post("http://localhost:3000/angular/express/users/register",user);  
  }
```

```
  getUser(){  
    return this.http.get("http://localhost:3000/angular/express/users/show");  
  }
```

```
}
```

*O service é responsável em se comunicar com o servidor Express.*

URL de comunicação com o Express.  
O padrão é a porta 300.

Esse “pedaço” em destaque da URL é definido no servidor Express, mais adiante.

# A pasta core - input user (.ts)

- Componente input-user

```
import { UserService } from '../services/user.service';
import { User } from '../models/User';
import { Component, OnInit } from '@angular/core';
```

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

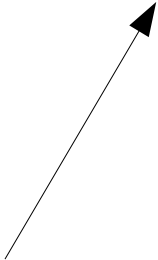
```
export class InputUserComponent {
```

```
  user: User;
```

```
  constructor(private userService: UserService) {
    this.user = new User();
  }
```

```
  onSubmit(){
    console.log(JSON.stringify(this.user));
    this.userService.setUser(this.user).subscribe(
      (res: User)=>{
        console.log(JSON.stringify(res));
      }
    );
  }
}
```

setUser recebe como parâmetro this.user, alimentado pela interface.  
A resposta (res) é usuário enviado.



# A pasta core – input user (.html)

```
<h1>Register User</h1>
<form (ngSubmit)="registerform.form.valid && onSubmit()" #registerform="ngForm" novalidate>
  <div class="form-group">
    <label for="firstName" class="control-label">First Name</label>
    <input type="text" class="form-control" name="firstName" id="firstName" [(ngModel)]="user.firstName" #firstName="ngModel"
      [ngClass]="{'is-invalid': registerform.submitted && firstName.invalid}" required>

    <div *ngIf="registerform.submitted && firstName.invalid" class="invalid-feedback">
      <div *ngIf="firstName.errors['required']">First Name is required</div>
    </div>
  </div>

  <div class="form-group">
    <label for="lastName" class="control-label">Last Name</label>
    <input type="text" class="form-control" name="lastName" id="lastName" [(ngModel)]="user.lastName" #lastName="ngModel"
      [ngClass]="{'is-invalid': registerform.submitted && lastName.invalid}" required>
    <div *ngIf="registerform.submitted && lastName.invalid" class="invalid-feedback">
      <div *ngIf="lastName.errors['required']">Last Name is required</div>
    </div>
  </div>

  <div class="form-group">
    <label for="email" class="control-label">E-mail</label>
    <input type="text" class="form-control" name="email" id="email" [(ngModel)]="user.email" #email="ngModel"
      [ngClass]="{'is-invalid': registerform.submitted && email.invalid}" required email>
    <div *ngIf="registerform.submitted && email.invalid" class="invalid-feedback">
      <div *ngIf="email.errors['required']">Email is required</div>
      <div *ngIf="email.errors['email']">Email must be a valid email address</div>
    </div>
  </div>
</div>
```

HTML com os inputs que  
serão enviados para o  
servidor Express

# A pasta core - input user (.html) (cont)

```
<div class="form-group">
  <label for="zipCode" class="control-label">Zip Code</label>
  <input type="text" class="form-control" name="zipCode" id="zipCode" [(ngModel)]="user.zipCode" #zipCode="ngModel"
    [ngClass]="{ 'is-invalid': registerform.submitted && zipCode.invalid }" required pattern="^[0-9]{5}(?-[0-9]{3})?$">
  <div *ngIf="registerform.submitted && zipCode.invalid" class="invalid-feedback">
    <div *ngIf="zipCode.errors['required']">Zip Code is required</div>
    <div *ngIf="zipCode.errors['pattern']">Pattern invalid</div>
  </div>
</div>

<div class="form-group">
  <label for="password" class="control-label">Password</label>
  <input type="password" class="form-control" name="password" id="password" [(ngModel)]="user.password" #password="ngModel"
    [ngClass]="{ 'is-invalid': registerform.submitted && password.invalid }" required minlength="6" pattern="^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])[a-zA-Z0-9]+$">
  <div *ngIf="registerform.submitted && password.invalid" class="invalid-feedback">
    <div *ngIf="password.errors['required']">Password is required</div>
    <div *ngIf="password.errors['minlength']">Password must be at least 6 characters</div>
    <div *ngIf="password.errors['pattern']">Pattern is invalid</div>
  </div>
</div>

<div class="form-group">
  <button type="submit" class="btn btn-primary">Register</button>
</div>

</form>
```



# A pasta core – display user (ts)

- Componente display-user

```
import { UserService } from '../services/user.service';  
import { User } from '../models/User';  
import { Component, OnInit } from '@angular/core';
```

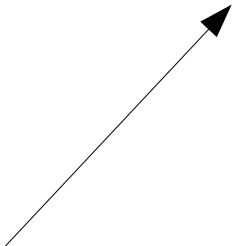
```
@Component({  
  selector: 'app-display-user',  
  templateUrl: './display-user.component.html',  
  styleUrls: ['./display-user.component.css']  
})
```

```
export class DisplayUserComponent {
```

```
  user: User = new User();
```

```
  constructor(private userService: UserService) {  
    this.userService.getUser().subscribe(  
      (res: User) => {  
        this.user = res;  
      }  
    );  
  }  
}
```

Recebe o User via getUser e atualiza a variável local this.user, a qual será exibida pelo HTML.



# A pasta core – display user (html)

```
<h1>User Information</h1>
<div>
  <div class="row">
    <label class="col-6">First Name</label>
    <div class="col-6">{{user.firstName}}</div>
  </div>
  <div class="row">
    <label class="col-6">Last Name</label>
    <div class="col-6">{{user.lastName}}</div>
  </div>
  <div class="row">
    <label class="col-6">E-mail</label>
    <div class="col-6">{{user.email}}</div>
  </div>
  <div class="row">
    <label class="col-6">Zip Code</label>
    <div class="col-6">{{user.zipCode}}</div>
  </div>
  <div class="row">
    <label class="col-6">Password</label>
    <div class="col-6">{{user.password}}</div>
  </div>
</div>
```

*Apenas exibe as informações do objeto user armazenado no componente.*

# core.module

```
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { CommonModule } from '@angular/common';
import { DisplayUserComponent } from '../display-user/display-user.component';
import { InputUserComponent } from '../input-user/input-user.component';
```

```
@NgModule({
  declarations: [DisplayUserComponent, InputUserComponent],
  exports: [DisplayUserComponent, InputUserComponent],
  imports: [
    CommonModule,
    FormsModule
  ]
})
export class CoreModule { }
```



Não esqueça e exportar os componentes  
e importar o FormsModule

# app.module

```
import { CoreModule } from './core/core.module';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HttpClientModule } from '@angular/common/http';

import { AppComponent } from './app.component';
import { InputUserComponent } from './core/input-user/input-user.component';
import { DisplayUserComponent } from './core/display-user/display-user.component';

const routes: Routes = [
  {path: '', component: InputUserComponent},
  {path: 'show', component: DisplayUserComponent},
];
```

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    CoreModule,
    RouterModule.forRoot(routes),
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



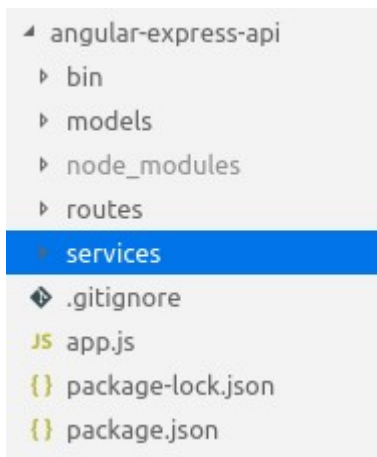
**O Servidor Express**

# O servidor Express

- O servidor Express irá gerenciar um único objeto em memória, um User.
- Para criar o projeto, vamos usar a `expresse-api`. Caso não tenha esse módulo do Node instalado, faça:
  - **`npm install -g express-generator-api`**
- Depois, pra criar o projeto, simplesmente faça:
  - **`express-api <nome_do_projeto>`**
    - cria a pasta de fato
  - **`cd <nome_do_projeto>`**
    - entra na pasta
  - **`npm install`**
    - instala as dependências na `node_modules`.

# O servidor express

- Organização do projeto:



- crie a pasta **models** e **services**.

# A pasta models.

- Crie o arquivo user.model.js

```
class UserModel{  
    constructor(firstName, lastName, login, email, zipCode, password){  
        this.login = login;  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.email = email;  
        this.zipCode = zipCode;  
        this.password = password;  
    }  
}  
  
module.exports = UserModel;
```



# A pasta services

- Crie o arquivo user.services.js

```
const UserModel = require("../models/user.model");

let user = null;

class UserService{

  static setUser(data){
    console.log("set");
    user = new UserModel(data.firstName,data.lastName,data.email,data.zipCode,data.password);
    return user;
  }

  static getUser(){
    return user;
  }

}

module.exports = UserService;
```

*Essa classe simula uma base de dados em memória.*

# Na pasta routes

- Modifique o arquivo user.routes.js

```
var express = require('express');  
var router = express.Router();  
var UserService = require('../services/user.service');
```

```
router.post('/register', async (req, res, next) => {  
  const user = await UserService.setUser(req.body);  
  return res.status(201).json(user);
```

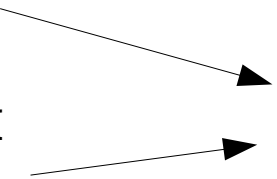
```
});
```

```
router.get('/show', async (req, res, next) => {
```

```
  const user = await UserService.getUser();  
  return res.json(user);
```

```
});
```

```
module.exports = router;
```



Coloca na resposta o resultado da chamada da URI (/register e /show).

# Em app.js

```
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var users = require('./routes/users.router');

var app = express();

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser())

app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
  next();
});

app.use('/angular/express/users', users);

module.exports = app;
```

# Inicie o Express

- `npm start`
  - dentro do diretório raiz.
  - Atenção: toda e qualquer modificação no servidor implica em reiniciar novamente o servidor.