

GROUPE 3



D'haeyere Corentin, Fuentes Gonzalez Lucas, Vandermeersch Laurent, Johnen Thomas, Thomas Loic

Introduction

Frontend :



Backend :



Base de données :



Angular CLI et Création de Composants

L'outil CLI (Command Line Interface) :

- Offre une panoplie de fonctionnalités essentielles
- Permet la création rapide de projets Angular
- Génération instantanée de composants, services, modules, etc.
- Réduit la complexité de tâches répétitives

Exemple :

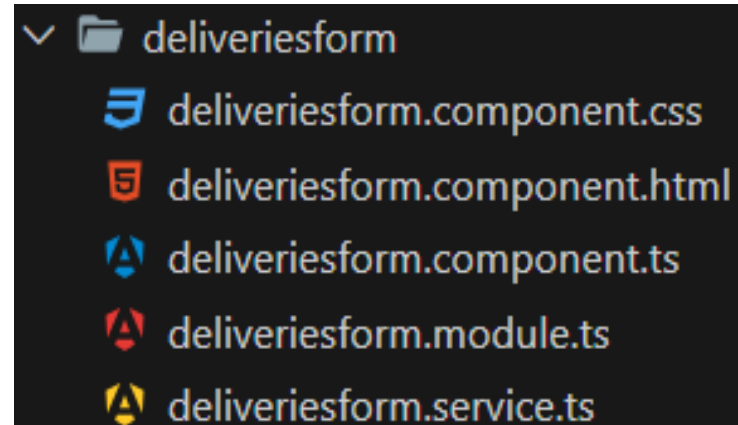
- `ng new « name project »`
- `ng g component « name component »`
- `Ng g module « name component »`
- `ng g service « name component »`

Structure d'un Composant Angular

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { DeliveriesformComponent } from './deliveriesform.component';

import { MatFormFieldModule } from '@angular/material/form-field';
import { MatInputModule } from '@angular/material/input';
import { MatButtonModule } from '@angular/material/button';
import { ReactiveFormsModule } from '@angular/forms';
import { MatSelectModule } from '@angular/material/select';
import { NavigationService } from '../services/navigation.service';

@NgModule({
  declarations: [DeliveriesformComponent],
  imports: [
    CommonModule,
    MatFormFieldModule,
    MatInputModule,
    MatButtonModule,
    ReactiveFormsModule,
    MatSelectModule
  ],
  providers: [NavigationService],
  exports: [DeliveriesformComponent],
})
export class DeliveriesformModule {}
```



```
deliveriesform
├── deliveriesform.component.css
├── deliveriesform.component.html
├── deliveriesform.component.ts
├── deliveriesform.module.ts
└── deliveriesform.service.ts
```

Gestion de la Navigation

La gestion de la navigation est simplifiée grâce au système de routage d'Angular. Les routes déclaratives offrent une navigation fluide entre les différentes parties de l'application.

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { UsersComponent } from './users/users.component';
import { LoginformComponent } from './loginform/loginform.component';
import { OrderDetailsComponent } from './order-details/order-details.component';
import { AdminComponent } from './admin/admin.component';
import { DeliveriesComponent } from './deliveries/deliveries.component';
import { UserformComponent } from './userform/userform.component';
import { AuthenticationGuard } from './authentication.guard';
import { OrderComponent } from './order/order.component';
import { AuthorizationGuard } from './authorization.guard';
import { Role } from './models/role';
import { ClientformComponent } from './clientform/clientform.component';
import { DeliveriesformComponent } from './deliveriesform/deliveriesform.component';

import { ordersDeliveryComponent } from './ordersDelivery/ordersDelivery.component';
import { ClientsComponent } from './clients/clients.component';

const routes: Routes = [
  { path: 'users', component: UsersComponent },
  { path: 'clients', component: ClientsComponent, canActivate: [AuthenticationGuard, AuthorizationGuard], data: { roles: [Role.Admin] } },
  { path: 'login', component: LoginformComponent },
  { path: '', component: DeliveriesComponent, canActivate: [AuthenticationGuard, AuthorizationGuard], data: { roles: [Role.Admin, Role.Deliverer] } },
  { path: 'admin', component: AdminComponent, canActivate: [AuthenticationGuard, AuthorizationGuard], data: { roles: [Role.Admin] } },
  { path: 'order', component: OrderComponent },
  { path: 'orders/:id', component: OrderDetailsComponent },
  { path: 'create-user', component: UserformComponent, canActivate: [AuthenticationGuard, AuthorizationGuard], data: { roles: [Role.Admin] } },
  { path: 'delivery/:deliveryId', component: ordersDeliveryComponent },
  { path: 'create-client', component: ClientformComponent, canActivate: [AuthenticationGuard, AuthorizationGuard], data: { roles: [Role.Admin] } },
  { path: 'create-delivery', component: DeliveriesformComponent, canActivate: [AuthenticationGuard, AuthorizationGuard], data: { roles: [Role.Admin] } }
];

Laurent Vandermeersch, last week • feat: navbar

Laurent Vandermeersch, last week | 1 author (Laurent Vandermeersch)
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Fondamentaux de NestJS

NestJS, inspiré par Angular et soutenu par TypeScript, offre un cadre robuste pour le développement backend. TypeScript est un Javascript plus typé. Ce qui offre une meilleure gestion des données.



```
import { Column, Entity, JoinColumn, ManyToOne, OneToMany, PrimaryGeneratedColumn } from "typeorm";
import { DeliveryState } from "src/enums/deliveryState.enum";
import { User } from "src/users/entities/user.entity";
import { Order } from "src/orders/entities/order.entity";
import { Surplus } from "src/surplus/entities/surplus.entity";

Lucas, yesterday | 1 author (Lucas)
@Entity()
export class Delivery {

  @PrimaryGeneratedColumn()
  id: number;

  @Column({ nullable: true })
  userId: number;

  @ManyToOne(() => User, user => user.deliveries, { onDelete: 'CASCADE' })
  @JoinColumn({ name: "userId" })
  user: User;

  @Column({ nullable: false, default: DeliveryState.Preparation })
  state: DeliveryState;

  @Column({ nullable: false })
  title: string;

  ⚠ @OneToMany(() => Order, order => order.delivery) Lucas, last week • refactor: fk in delivery-order
  orders: Order[];

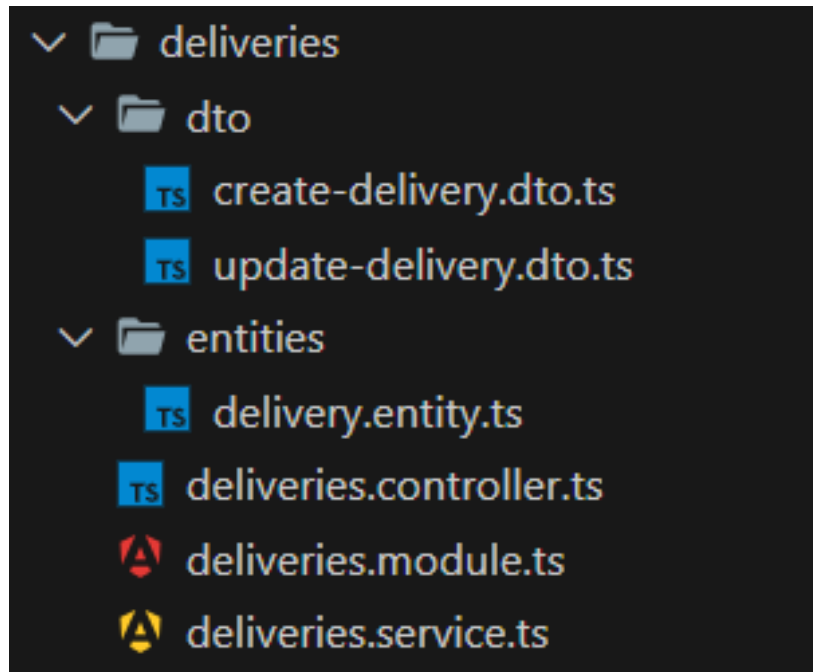
  @OneToMany(() => Surplus, surplus => surplus.delivery)
  surplus: Surplus[];
}
```

CLI de NestJS et Génération de Modules

Le CLI de NestJS est aussi un élément crucial pour démarrer rapidement un projet. Il permet la création instantanée de modules, de contrôleurs, de services, etc.

```
cdhae@NITROC-FIXE in ..\snappies-front on □ dev [!]  
x nest  
Usage: nest <command> [options]  
  
Options:  
  -v, --version          Output the current version.  
  -h, --help             Output usage information.  
  
Commands:  
  new[n [options] [name] Generate Nest application.  
  build [options] [app]   Build Nest application.  
  start [options] [app]   Run Nest application.  
  info|i                 Display Nest project details.  
  add [options] <library> Adds support for an external library to your project.  
  generate|g [options] <schematic> [name] [path] Generate a Nest element.  
    Schemas available on @nestjs/schematics collection:  
  
  name      alias      description  
  application application Generate a new application workspace  
  class      cl         Generate a new class  
  configuration config    Generate a CLI configuration file  
  controller co        Generate a controller declaration  
  decorator  d          Generate a custom decorator  
  filter     f          Generate a filter declaration  
  gateway    ga         Generate a gateway declaration  
  guard      gu         Generate a guard declaration  
  interceptor itc       Generate an interceptor declaration  
  interface  itf       Generate an interface  
  library    lib       Generate a new library within a monorepo  
  middleware mi        Generate a middleware declaration  
  module     mo        Generate a module declaration  
  pipe       pi        Generate a pipe declaration  
  provider   pr        Generate a provider declaration  
  resolver   r          Generate a GraphQL resolver declaration  
  resource   res       Generate a new CRUD resource  
  service    s          Generate a service declaration  
  sub-app    app       Generate a new application within a monorepo  
  
cdhae@NITROC-FIXE in ..\snappies-front on □ fix/authentication  
λ □
```

Structure d'un Module dans NestJS



Dans NestJS, un module est un élément fondamental qui encapsule les composants logiques de l'application. NestJS étant basé sur une structure très courante, nous avons pour chaque « ressource » un contrôleur permettant de définir les routes auxquelles les utilisateurs auront accès, un service, permettant les vérifications de type business et un module permettant les imports des autres composants de l'application.

Middleware et Guard

Les middlewares et guards jouent un rôle crucial dans le traitement des requêtes HTTP dans NestJS. Ils interviennent entre la requête et la réponse, permettant le traitement, la validation et l'autorisation.

```
1 @Injectable()
2 export class RolesGuard implements CanActivate {
3   constructor(private reflector: Reflector) {}
4
5   canActivate(context: ExecutionContext): boolean {
6     const requiredRoles = this.reflector.getAllAndOverride<Role[]>(ROLES_KEY, [
7       context.getHandler(),
8       context.getClass(),
9     ]);
10
11     if (!requiredRoles) {
12       return true;
13     }
14
15     const { user } = context.switchToHttp().getRequest();
16     return requiredRoles.some((role) => user.role === role);
17   }
18 }
```

```
1 @Injectable()
2 export class LoggerMiddleware implements NestMiddleware {
3   private logger = new Logger("HTTP");
4
5   use(request: Request, response: Response, next: NextFunction): void {
6     const { ip, method, originalUrl } = request;
7     const userAgent = request.get("user-agent") || "";
8
9     response.on("finish", () => {
10       const { statusCode, statusMessage } = response;
11
12       this.logger.log(
13         `${method} ${originalUrl} ${statusCode} ${statusMessage} - ${userAgent} ${ip}`,
14       );
15     });
16
17     next();
18   }
19 }
```

Base de Données et ORM/ODM dans NestJS

```
1  @Entity()
2  export class User {
3
4      @PrimaryGeneratedColumn()
5      id: number;
6
7      @Column({ nullable: false })
8      firstname: string;
9
10     @Column({ nullable: false })
11     lastname: string;
12
13     @Column({ unique: true, nullable: false })
14     email: string;
15
16     @Column({ nullable: false })
17     password: string;
18
19     @Column({ nullable: false, default: Role.User })
20     role: Role;
21
22     @OneToMany(() => Delivery, delivery => delivery.userId)
23     deliveries: Delivery[];
24 }
```

NestJS facilite l'intégration avec différents outils d'accès aux données, simplifiant ainsi la gestion des bases de données et des opérations liées. Dans notre cas, nous avons opté pour PostgreSQL, une base de données relationnelles que nous avons l'habitude d'utiliser. En combinaison avec, nous utilisons TypeORM, un ORM (Object-Relational Mapping) pour faciliter la gestion des données dans notre application. TypeORM permet de représenter sous forme de classes les différentes entités présentes en base de données.

Les pipelines

Nos deux applications sont déployées sur un VPS Oracle à l'aide de Docker. Les deux projets contiennent un fichier Dockerfile décrivant la marche à suivre pour créer les images. Ensuite, grâce aux Github Actions, nous avons une pipeline « Dev » et une pipeline « Prod ». Ces deux pipelines ont exactement le même comportement pour les applications. Une image Docker est build, puis envoyée sur le Docker Hub avec un tag « dev » pour les images de dev. Une connexion ssh est ensuite établie avec le VPS afin d'exécuter les commandes permettant de créer des containers et de les démarrer.

```
1 FROM node:20-alpine as build
2
3 WORKDIR /app
4
5 COPY package.json ./
6
7 RUN npm install
8
9 COPY . .
10
11 RUN npm run deploy
12
13 FROM nginx:alpine
14
15 COPY --from=build /app/dist/snappies-front /usr/share/nginx/html
16
17 COPY ./nginx.conf /etc/nginx/conf.d/default.conf
18
19 EXPOSE 80
```

Conclusion

Angular et NestJS, bien que différents dans leurs objectifs principaux, partagent des caractéristiques essentielles. Tous deux reposent sur TypeScript, offrant ainsi la robustesse du typage statique et une meilleure qualité de code. De plus, la structuration modulaire est au cœur de ces Framework, permettant une organisation efficace des composants et une réutilisation aisée du code.

Merci de nous avoir
écouté