NestJS Cheat Sheet

Installation

```
# Install NestJS CLI globally
  npm install -g @nestjs/cli

# Or with yarn
  yarn global add @nestjs/cli

# Or with pnpm
  pnpm add -g @nestjs/cli
```

Create a new NestJS application

```
# Create a new project
  nest new project-name

# Create a new project with specific package manager
  nest new project-name --package-manager npm
  nest new project-name --package-manager yarn
  nest new project-name --package-manager pnpm
```

Generate components

```
# Generate a controller
nest g controller users

# Generate a service
nest g service users

# Generate a module
nest g module users

# Generate a resource (CRUD)
nest g resource users

# Generate a class
nest g class users/dto/create-user.dto

# Generate an interface
nest g interface users/interfaces/user.interface

# Generate a middleware
nest g middleware logger
```

Project structure

```
src/
├── app.controller.ts      # Basic controller
├── app.module.ts          # Root module
├── app.service.ts         # Basic service
└── main.ts                # Entry point
```

Providers and Dependency Injection

```
// Service example
@Injectable()
export class UsersService {
  findAll() {
    return ['user1', 'user2'];
  }
}

// Controller using the service
@Controller('users')
export class UsersController {
  constructor(private usersService: UsersService) {}

  @Get()
  findAll() {
    return this.usersService.findAll();
  }
}
```

HTTP Request decorators

```
@Get()
@Post()
@Put()
@Delete()
@Patch()
@Options()
@Head()
@All()
```

Route parameters

```
@Get(':id')
findOne(@Param('id') id: string) {
  return `This action returns a # {id} item`;
}
```

Request object

```
@Post()
create(@Body() createUserDto: CreateUserDto, @Req() request: Request) {
  // Access headers, body, or query parameters
  console.log(request.headers);
  return 'This action adds a new user';
}
```

Middleware

```
@Injectable()
export class LoggerMiddleware implements NestMiddleware {
  use(req: Request, res: Response, next: NextFunction) {
    console.log('Request...');
    next();
  }
}
```

Guards

```
@Injectable()
export class AuthGuard implements CanActivate {
  canActivate(context: ExecutionContext): boolean {
    const request = context.switchToHttp().getRequest();
    return validateRequest(request);
  }
}
```

Pipes

```
@Post()
create(@Body(new ValidationPipe()) createUserDto: CreateUserDto) {
  return 'This action adds a new user';
}
```

Exception filters

```
@Catch(HttpException)
export class HttpExceptionFilter implements ExceptionFilter {
  catch(exception: HttpException, host: ArgumentsHost) {
    const ctx = host.switchToHttp();
    const response = ctx.getResponse<Response>();
    const request = ctx.getRequest<Request>();
    const status = exception.getStatus();

    response.status(status).json({
      statusCode: status,
      timestamp: new Date().toISOString(),
      path: request.url,
    });
  }
}
```

Project setup

```
  pnpm install
```

Compile and run the project

```
# development
  pnpm run start

# watch mode
  pnpm run start:dev

# production mode
  pnpm run start:prod
```

ENV Configuration

A good approach for using this technique in Nest is to create a `ConfigModule` that exposes a `ConfigService` which loads the appropriate `.env` file. While you may choose to write such a module yourself, for convenience Nest provides the `@nestjs/config` package out-of-the box. We'll cover this package in the current chapter.

```
npm i --save @nestjs/config
```

Config app.module.ts

Typically, we'll import it into the root `AppModule` and control its behavior using the `.forRoot()` static method. During this step, environment variable key/value pairs are parsed and resolved.

`app.module.ts`

```
import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';

@Module({
  imports: [
    ConfigModule.forRoot({
            isGlobal: true,
            envFilePath: '.env',
     }),
    ],
})
export class AppModule {}
```

using `main.ts` with .env

```
import { ConfigService } from '@nestjs/config';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.useGlobalPipes(new ValidationPipe());

  const configService = app.get(ConfigService);
  const PORT = configService.getOrThrow<number>('PORT');

  await app.listen(PORT);
}
bootstrap();
```