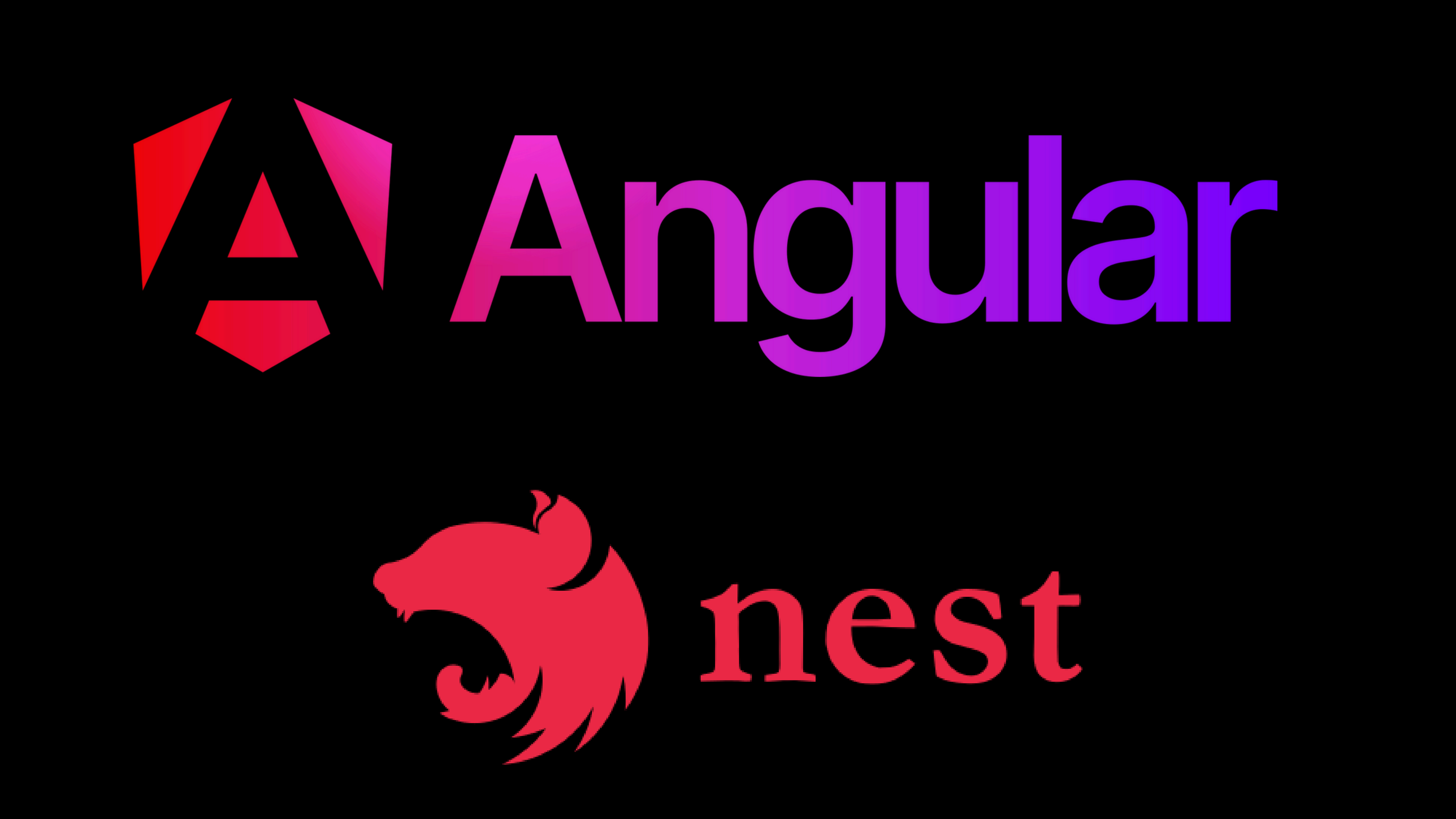

Angular
nest

Wer sind wir?

Hannes Frey **Anton Gerdts** **Urs Weniger**

https://github.com/ursweniger/angular-workshop

Agenda

Fundamentals und Setup

Setup

Middleware und Routing

MongoDB Anbindung

NestJS

Was ist NestJS?

- Framework zur Entwicklung von Server-side Anwendungen
- Open-Source
- Sprache: TypeScript
- Verwendet Express (optional Fastify)
- Architektur stark and Angular angelehnt

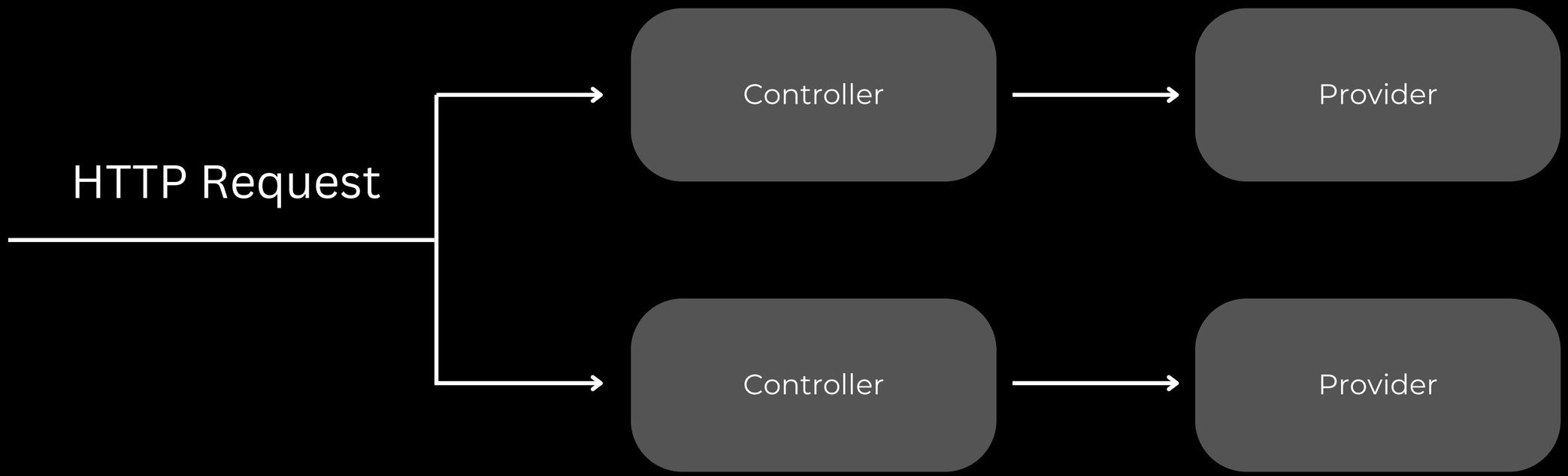
Warum NestJS?

- Skalierbarkeit durch modularen Aufbau
- Dependency Injection ermöglicht lose Kopplung, bessere Testbarkeit
- Dekoratoren, die Metadaten hinzufügen und die Lesbarkeit erhöhen
- Integration von Mikroservices (WebSocket, Apache Kafka, Redis...)
- Einfache Erstellung von REST APIs

Architektur

- Module:
  - Zusammenfassen von Komponenten, Controllern und Providern.
- Controller:
  - Definieren Routen/Endpunkte für die Anwendung.
- Provider:
  - Können mithilfe von Dependency Injection (DI) in Controller oder andere Provider injiziert werden.

Architektur



Projektstruktur

```
.
├── src
│   ├── app.controller.spec.ts  # Unit-Tests für den App-Controller
│   ├── app.controller.ts       # Der Hauptcontroller der Anwendung, definiert Endpunkte
│   ├── app.module.ts           # Das Hauptmodul der Anwendung, importiert und organisiert andere Module
│   ├── app.service.ts          # Der Hauptdienst der Anwendung, enthält Geschäftslogik
│   └── main.ts                 # Der Einstiegspunkt der Anwendung, bootstrapt das NestJS-Modul
├── test
│   ├── app.e2e.spec.ts         # End-to-End (E2E) Tests für die Anwendung
│   └── jest.e2e.json           # Konfigurationsdatei für Jest E2E Tests
├── .eslintrc.js                # Konfigurationsdatei für ESLint, das JavaScript/TypeScript-Linting-Tool
├── .gitignore                  # Dateien und Ordner, die von Git ignoriert werden sollen
├── .prettierrc                 # Konfigurationsdatei für Prettier, das Code-Formatierungs-Tool
├── nestcli.json                # Konfigurationsdatei für die NestJS CLI
├── package-lock.json           # Lock-Datei für Node-Pakete, gewährleistet konsistente Installationen
├── package.json                # Enthält Projektinformationen und Abhängigkeiten
├── Readme.md                   # README-Datei mit Projektinformationen und Anweisungen
├── tsconfig.build.json         # TypeScript-Konfigurationsdatei für den Build-Prozess
└── tsconfig.json               # Hauptkonfigurationsdatei für TypeScript
```

Architektur - Provider

events.service.ts

```ts
import { Injectable } from '@nestjs/common';

@Injectable()
export class EventsService {
  getHello(): string {
    return 'Hello World!';
  }
}
```

Architektur - Controller

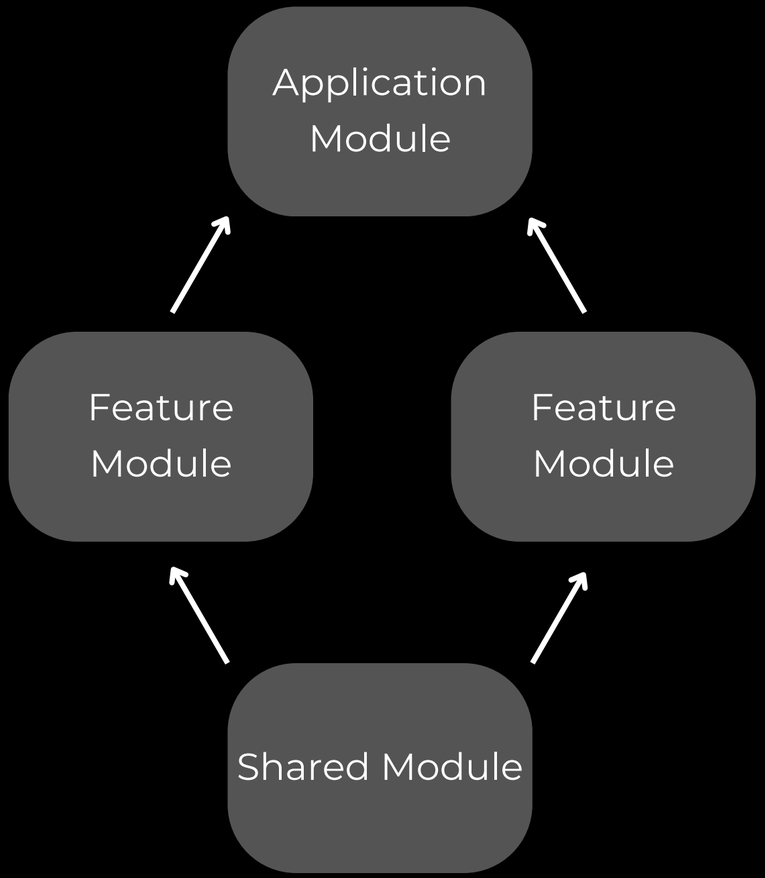
events.controller.ts

```ts
import { Controller, Get } from '@nestjs/common';
import { EventsService } from './events.service';

@Controller('events')')
export class EventsController {
  constructor(private eventsService: EventsService) {}

  @Get()
  getHello(): string {
    return this.eventsService.getHello();
  }
}
```

Architektur - Module

app.module.ts

```ts
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { EventsModule } from './feature/feature.module';

@Module({
 imports: [EventsModule],
 controllers: [AppController],
 providers: [AppService],
 exports: [AppService],
})
export class AppModule {}
```



Architektur - Main

main.ts

```ts
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';

async function bootstrap() {
 const app = await NestFactory.create(AppModule);
 await app.listen(3000);
}

bootstrap();
```

NestJS CLI

```
$ npm i -G @nestjs/cli
$ nest new my-application
```

Neues NestJS Projekt erstellen

```
$ nest generate module events
$ nest generate controller events
$ nest generate service events
```

Erzeugen von Dateien, entsprechend der Architekurschematik

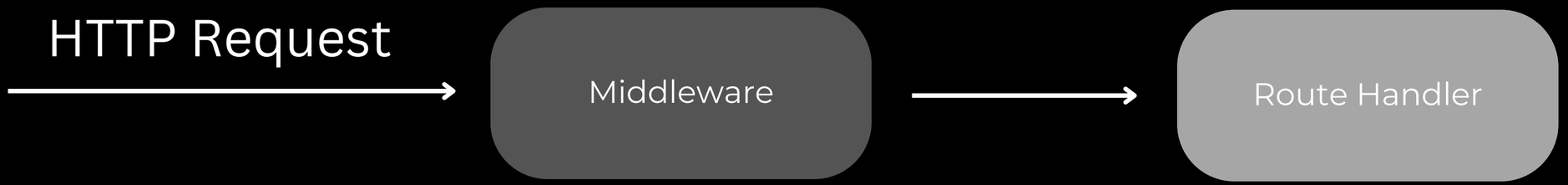
```
$ nest build
$ nest start --watch
```

Kompilieren oder direkt starten

Aufgabe 6

10 Minuten

Middleware



- Kann beliebigen Code ausführen
- Ermöglich Änderungen am req und res body durchführen
- Req/Res Zyklus fortführen oder beenden

Middleware

logger.middleware.ts

```ts
import { Injectable, NestMiddleware } from '@nestjs/common';
import { Request, Response, NextFunction } from 'express';

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

Middleware

events.module.ts

```
import { MiddlewareConsumer, Module, NestModule } from '@nestjs/common';
import { EventsController } from './events.controller';
import { EventsService } from './events.service';
import { LoggerMiddleware } from 'src/logger/logger.middleware';

@Module({
 controllers: [EventsController],
 providers: [EventsService]
})
export class EventsModule implements NestModule {
 configure(consumer: MiddlewareConsumer) {
  consumer
   .apply(LoggerMiddleware)
   .exclude('login')
   .forRoutes({path: 'events', method: RequestMethod.GET});
 }
}
```

Routing

events.controller.ts

```ts
@Controller('events')
export class EventsController {
  constructor(private readonly eventsService: EventsService) {}

  @Get()
  findAll(): any {
    return this.eventsService.findAll();
  }
  @Get(':id')
  findOne(@Param id: number): any {
    return this.eventsService.findOne(id);
  }
  @Post()
  create(@Body event: any): any {}
  @Delete(':id')
  delete(@Param id: number): any {}
}
```

Pipes

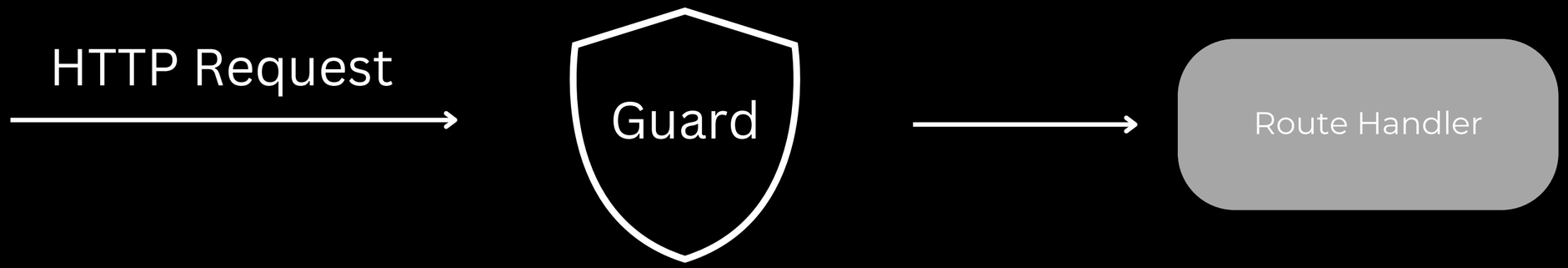
validation.pipe.ts

```ts
import { PipeTransform, Injectable, ArgumentMetadata } from '@nestjs/common';

@Injectable()
export class ValidationPipe implements PipeTransform {
     transform(value: any, metadata: ArgumentMetadata) {
          return value;
     }
}
```

events.controller.ts

```ts
@Get()
findOne(@Query('id', ParseIntPipe) id: number) {
     return this.catsService.findOne(id);
}
```

Guards



auth.guard.ts

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

}
```

```ts
@Controller('cats')
@UseGuards(AuthGuard)
export class CatsController {}
```

Aufgabe 7

15 Minuten

MongoDB

```
$ npm i @nestjs/mongoose mongoose
```

app.module.ts

```ts
import { Module } from '@nestjs/common';
import { MongooseModule } from '@nestjs/mongoose';
@Module({
  imports: [MongooseModule.forRoot('mongodb://localhost/nest')],
})
export class AppModule {}
```

MongoDB - Model Injection

cat.schema.ts

```ts
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import { HydratedDocument } from 'mongoose';

export type CatDocument = HydratedDocument<Cat>;
@Schema()
export class Cat {
  @Prop()
  name: string;
  @Prop({ required: true })
  age: number;
  @Prop([String])
  breed: string;
}

export const CatSchema = SchemaFactory.createForClass(Cat);
```

MongoDB - Model Injection

cat.module.ts

```
import { Module } from '@nestjs/common';
import { MongooseModule } from '@nestjs/mongoose';
import { CatsController } from './cats.controller';
import { CatsService } from './cats.service';
import { Cat, CatSchema } from './schemas/cat.schema';

@Module({
  imports: [MongooseModule.forFeature([{ name: Cat.name, schema: CatSchema }])],
  controllers: [CatsController],
  providers: [CatsService],})

export class CatsModule {}
```

MongoDB - Model Injection

cat.service.ts

```ts
import { Model } from 'mongoose';
import { Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { Cat } from './schemas/cat.schema';
import { CreateCatDto } from './dto/create-cat.dto';

@Injectable()
export class CatsService {
    constructor(@InjectModel(Cat.name) private catModel: Model<Cat>) {}

    async create(createCatDto: CreateCatDto): Promise<Cat> {
        const createdCat = new this.catModel(createCatDto);
        return createdCat.save();
    }
}
```

Aufgabe 8

25 Minuten