

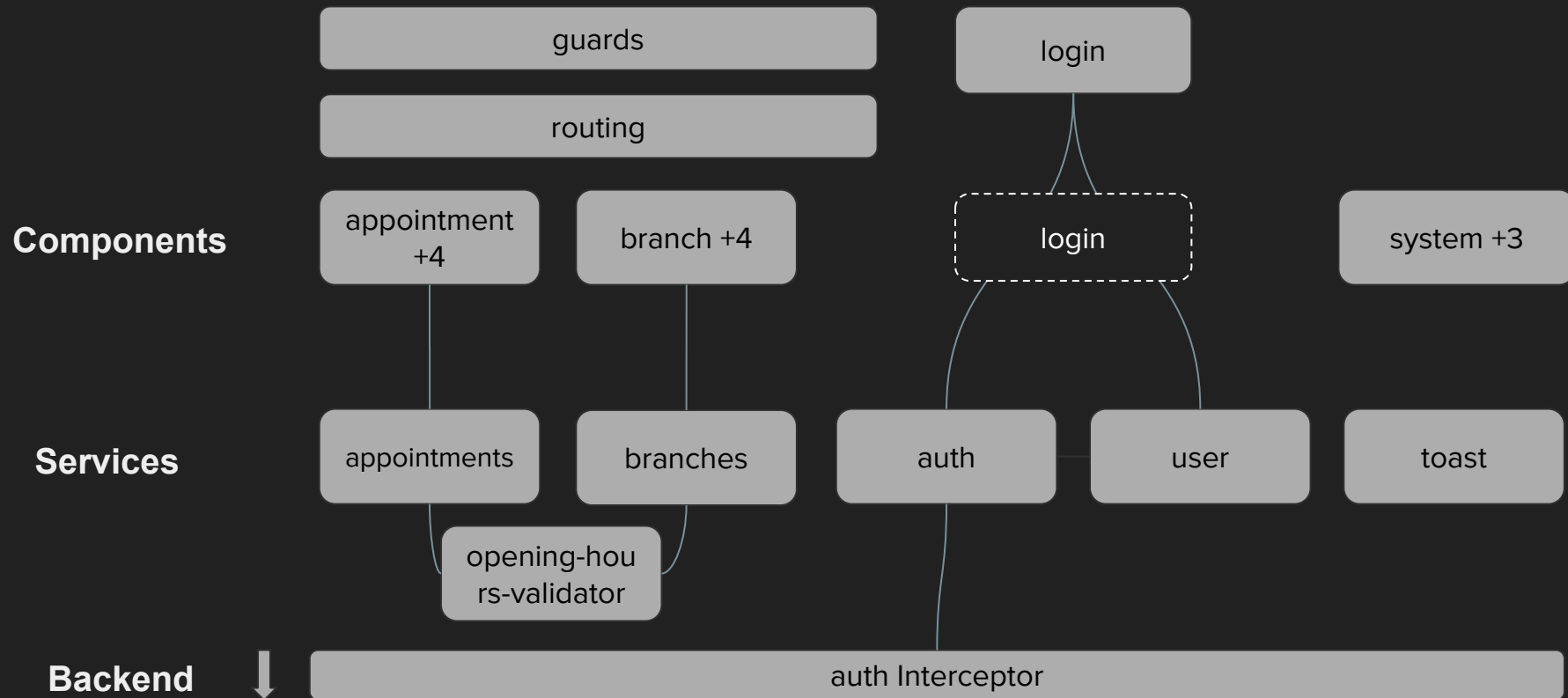
# Rapid Prototyping

Frieder Ludwig & Paul Machelett

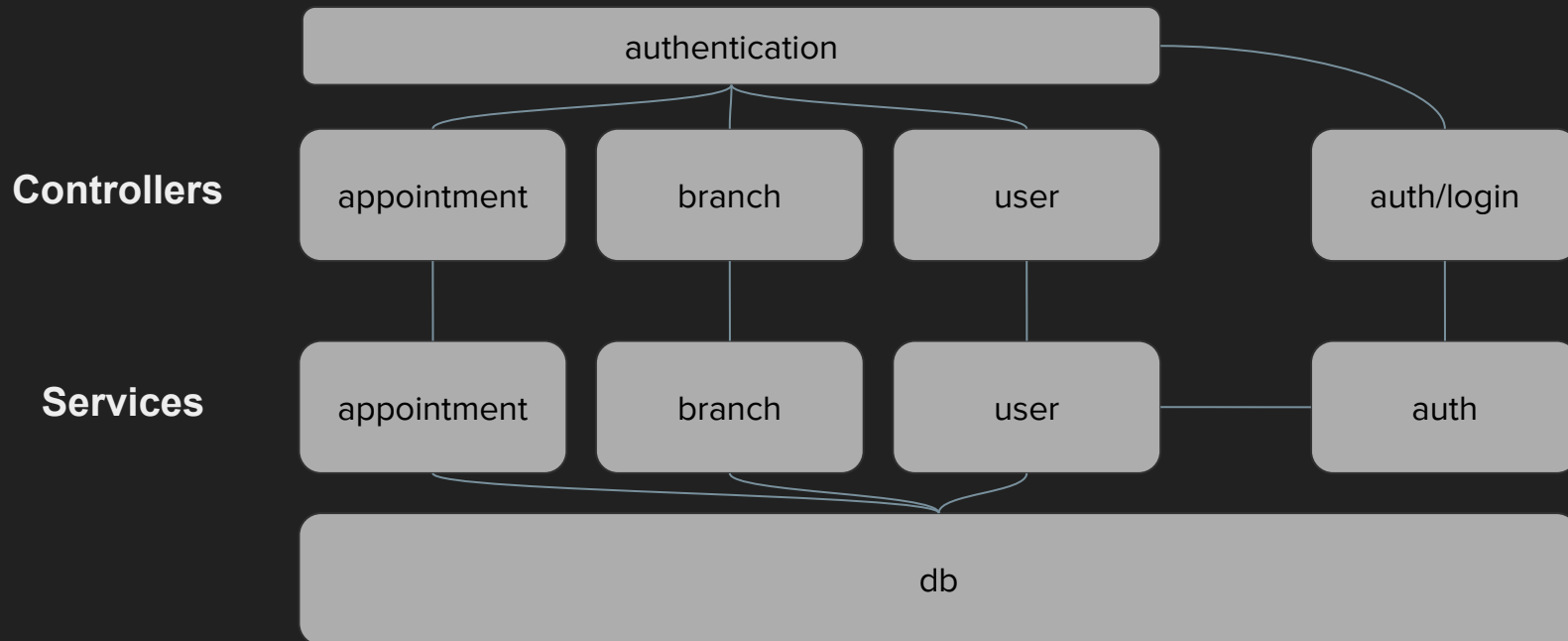
# Übersicht der Inhalte

1. Frontend & Backend Architektur
2. Authentifizierung
3. Rollen und Berechtigungen
4. Datenbank
5. Zusammenarbeit im Team
6. Reflexion

# Frontend



# Backend



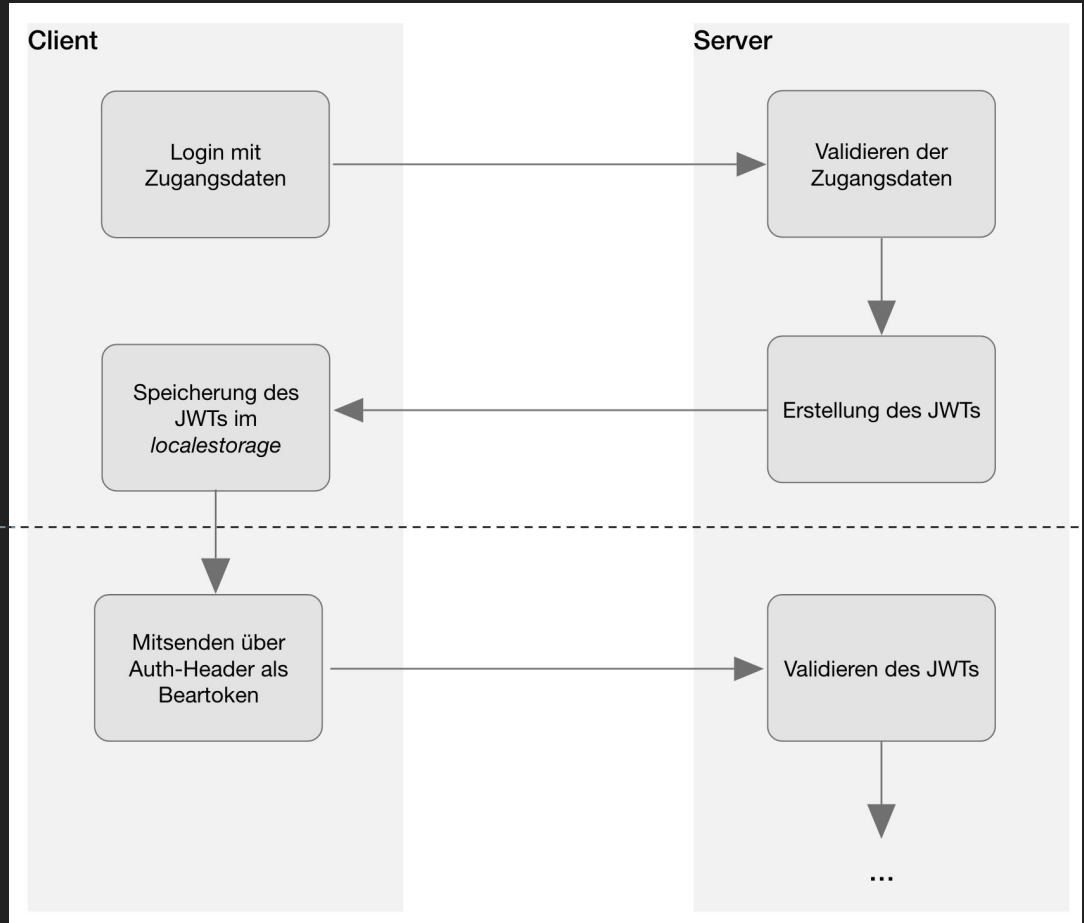
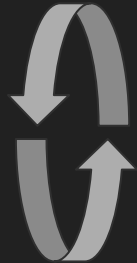
# Authentifizierung

- Erfolg über JSON Web Token (JWT)
- **JWT**: Kompakter, signierter Token für Authentifizierung in Web-Apps
- **Bestandteile**: Header (Metadaten), Payload (Benutzerdaten), Signature (Integrität)
- **Vorteil**: Zustandslose Authentifizierung ohne Serverspeicherung

# Authentifizierung

LogIn

Weitere Requests



# Authentifizierung Frontend/UI

- Ausgewählte Routes werden über einen ***AuthGuard*** geschützt
- User authentifiziert sich über Login-Formular
- Bei erfolgreicher Anmeldung wird JWT im *localStorage* gespeichert

# AuthGuard Implementierung Frontend

auth.guard.ts

```
@Injectable({
  providedIn: 'root',
})
export class AuthGuard implements CanActivate {
  constructor(
    private authService: AuthService,
    private router: Router
  ) {}

  canActivate(): boolean {
    if (this.authService.isAuthenticated()) {
      return true;
    } else {
      this.router.navigate(['/login']);
      return false;
    }
  }
}
```

app.routes.ts

```
export const appRoutes: Route[] = [
  {
    path: 'appointments',
    component: AppointmentListComponent,
    canActivate: [AuthGuard],
  },
  {
    path: 'appointments/create',
    component: AppointmentCreateRouteComponent,
    canActivate: [AuthGuard],
  },
  {
    path: 'appointments/:id',
    component: AppointmentDetailRouteComponent,
    canActivate: [AuthGuard],
  },
  ...
  { path: 'login', component: LoginComponent },
  { path: '**', redirectTo: '/appointments', pathMatch: 'full' },
];
```



# Authentifizierung Backend/API

- Routen werden via *AuthGuard* geschützt
- JWT wird über *Passport Strategy* validiert/erstellt
- Zugangsdaten (E-Mail, Passwort) werden in der DB gespeichert

# AuthGuard Implementierung Backend

appointment.controller.ts

```
@Controller('appointment')
export class AppointmentController {
  constructor(...) {}
  @UseGuards(JwtAuthGuard)
  @Get()
  GetData() {
    return this.appointmentService.getAll();
  }
  ...
}
```

# Rollen und Berechtigungen

- **User-Objekt:** Enthält ein role-Attribut (user oder admin).
- **Admin-Berechtigungen:**  
*Branches* erstellen/bearbeiten, alle *Appointments* bearbeiten
- **User-Berechtigungen:**  
*Branches* ansehen, eigene *Appointments* bearbeiten
- **Zugriffsprüfung:** Eine gemeinsame Validierungsfunktion prüft die Berechtigungen und wird in einem Shared-Modul bereitgestellt, um sie sowohl im Frontend als auch Backend zu nutzen

# Berechtigung prüfen – Implementierung

shared.ts

```
export function validateUserPermissionsForAppointment(  
  appointment: Appointment,  
  user: User  
) {  
  return user.role === UserRoles.ADMIN || user.id === appointment.createdByUser;  
}  
  
export function validateUserPermissionsForBranch(user: User) {  
  return user.role === UserRoles.ADMIN;  
}
```

# Datenbank als Persistenzschicht

- Als DBMS wird **PostgreSQL** (Postgres) verwendet
- CRUD Befehle werden über **TypeORM** ausgeführt, dies erlaubt mit Daten wie mit normalen Objekten im Code zu arbeiten ➡ So muss kein zusätzliches SQL geschrieben werden
- **Verwendete Entities:** User, Appointment, Branch

# Datenbankverbindung

```
database.providers.ts

export const DatabaseProvider = {
  provide: 'DATA_SOURCE',
  useFactory: async () => {
    const dataSource = new DataSource({
      type: 'postgres',
      host: process.env.DB_HOST,
      port: Number(process.env.DB_PORT),
      username: process.env.DB_USERNAME,
      password: process.env.DB_PASSWORD,
      database: process.env.DB_DATABASE,
      entities: [AppointmentEntity, UserEntity,
BranchEntity],
      synchronize: true,
    });

    return dataSource.initialize();
  },
};
```

.initialize() gibt Promise zurück, daher ist ein async Provider notwendig

Datenbank Parameter einstellen

Verbindung mit DB erzeugen

# Datenbank

```
branch.entity.ts

@Entity("branch")
export class BranchEntity {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ unique: true })
  city: string;

  @Column()
  openingHoursStart: string;

  @Column()
  openingHoursEnd: string;

  @OneToMany(() => AppointmentEntity, (appointment) =>
    appointment.branch)
  appointments?: AppointmentEntity[];
}
```

Erzeugung einer **Entität** mit **TypeORM** welche später die User DB Tabelle abbildet.

```
branch.provider.ts

export const branchProvider = [
  {
    provide: 'BRANCH_REPOSITORY',
    useFactory: (dataSource: DataSource) =>
      dataSource.getRepository(BranchEntity),
    inject: ['DATA_SOURCE'],
  },
];
```

Erstellung eines **Repositorys** anhand der Entität welches später den Datenaustausch mit der DB User Tabelle per **Dependency Injection** möglich macht.

# Datenbank

```
branch.service.ts

@Injectable()
export class BranchService {
  constructor(
    @Inject('BRANCH_REPOSITORY')
    private branchRepository: Repository<BranchEntity>
  ) {}

  getAll(): Promise<Branch[]> {
    return this.branchRepository.find();
  }
}
```

Einbetten des Repositorys in den Service via **Dependency Injection** damit innerhalb des Services auf die DB zugegriffen werden kann via.



# Zusammenarbeit im Team

- Gemeinsame Coding Sessions (Pair Programming)
- Mindestens ein wöchentliches Treffen
- Aufgaben nach Features verteilt (je Feature FE & BE Entwicklung)

## Ablaufplan & Meilensteine

- Bis 25.10. ➞ Umsetzung der Mindestanforderungen
- Bis 08.11. ➞ Umsetzung von Login und Rollen
- Bis 14.11. ➞ Vorbereitung Präsentation und kleine Optimierungen

# Was würden wir aus heutiger Sicht anders machen?

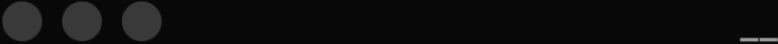
- UI bereits im voraus erstellen, um besseres Gefühl für die Anwendung zu bekommen und kein “umstyling” im Code machen zu müssen
- Systemarchitektur früher festlegen

# Präsentation der Anwendung

**Fragen**

## Link Repository

<https://github.com/friederludwig/appointment-app>



```
if (FRAGEN === true) {
```

```
    return ANTWORTEN;
```

```
} else {
```

```
    return "Danke für Ihre Aufmerksamkeit!";
```

```
}
```