
COLLABORATIVE NOTES

ONLINE ECHTZEIT TEXT EDITOR
PROJEKT -MIT ANGULAR 17 UND
NODE.JS

VON:

AHMAD ABEER AHSAN

MOEEZ MUHAMMAD AHSAN



VERWENDETE TECHNOLOGIEN

- Angular 17
 - Angular Material UI Bibliothek
- Node Server
- MongoDB
- WebSockets
- CRDT Struktur
- LocalStorage

WAS WIR WOLLEN

- Textdatei schreiben
- Textdatei lokal speichern
- Benutzer erstellen und Datei online speichern
- Benutzer lokal speichern
- Dateien gemeinsam mit anderen Benutzern in Echtzeit bearbeiten

ANGULAR

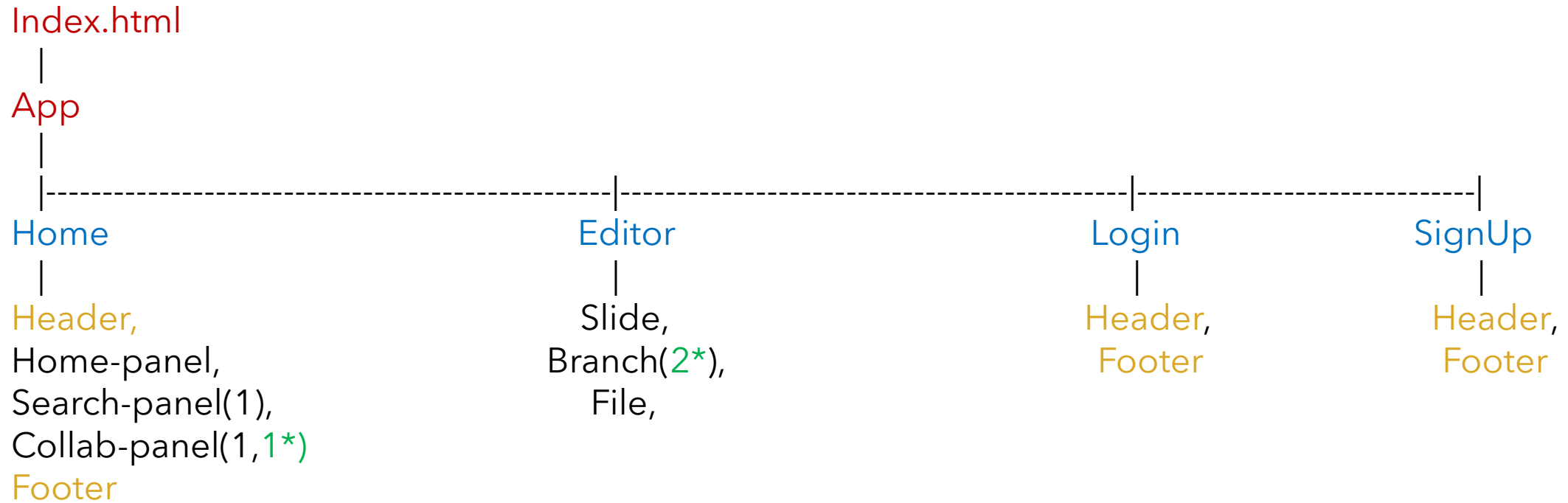
Angular ist ein von Google entwickeltes Open-Source-Framework zur Erstellung von Webanwendungen. Es basiert auf TypeScript und ermöglicht die Entwicklung skalierbarer, leistungsfähiger und wartbarer Single-Page-Anwendungen (SPAs).

- verwendet „TypeScript“
- Dank seiner komponentenbasierten Architektur und Dependency Injection ist die Code-Wiederverwendbarkeit, -Skalierbarkeit und -Testbarkeit hoch.
- Angular ist weniger flexibel als andere Frameworks, da es strikte Struktur- und Designvorgaben hat. Zudem hat es eine steilere Lernkurve, da Entwickler zunächst TypeScript und die umfangreichen Angular-Konzepte verstehen müssen.

INHALT

- Angular Komponenten Struktur
- Applikation Entwurf
- Angular Material UI
- Theme Service
- LocalStorage Implementation
- Angular Integriertes Routing
- Angular Forms
- Observables aus der rxjs-Bibliothek
- CRDT - Datenstruktur für unseren Text
- Node Server/ Websockets

WIE SIEHT EINE ANGULAR-KOMPONENTE STRUKTURIERT AUS?(1)



- wiederverwendbare Komponenten
- iterierbare Komponenten*

WIE SIEHT EINE ANGULAR-KOMPONENTE STRUKTURIERT AUS?(2)

▼ header
header.com...
<> header.com...
TS header.com...
TS header.com...

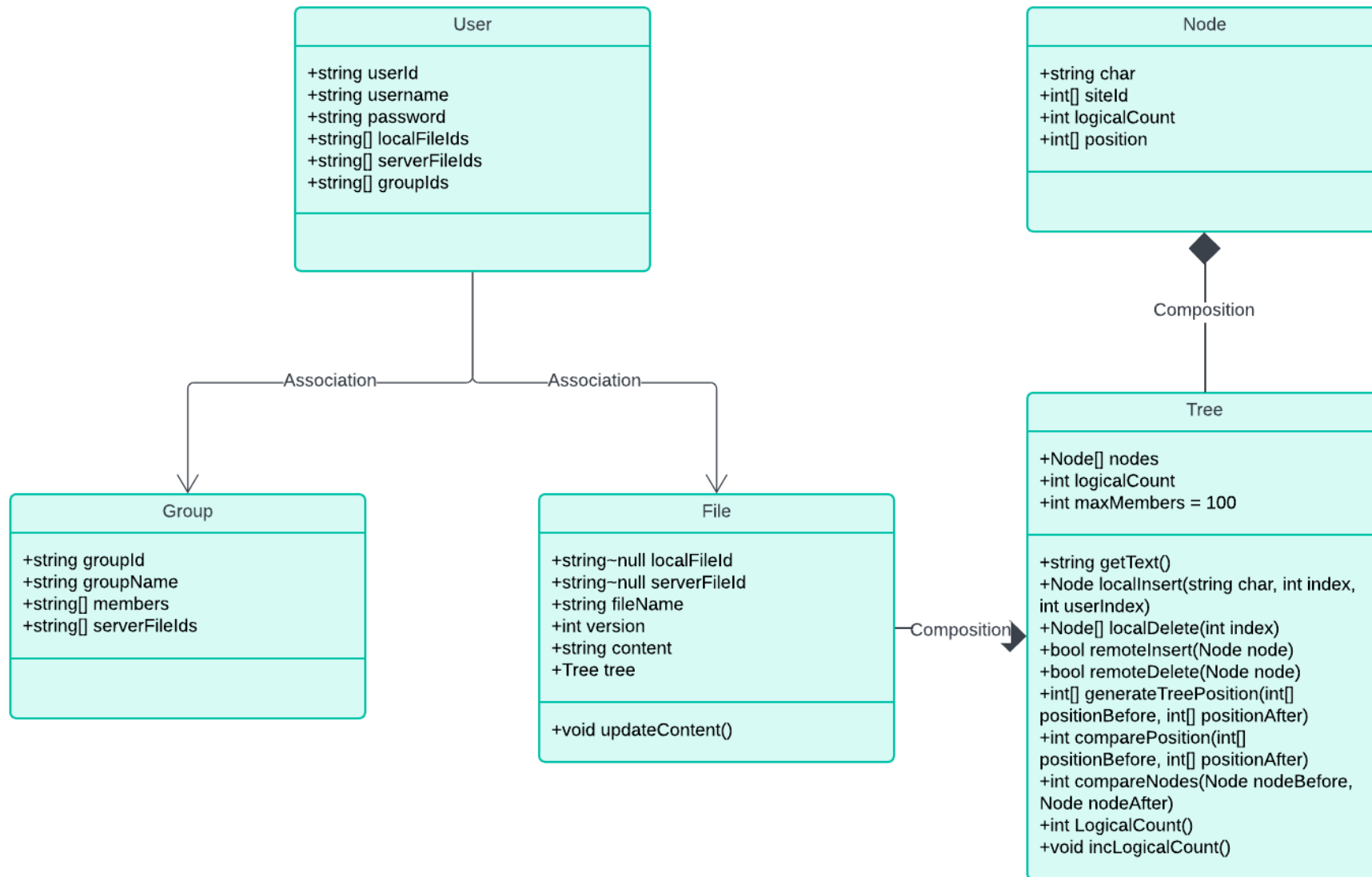
▼ services
TS files.service.... M
TS files.service.ts M
TS theme.servi... M
TS theme.servi... M
TS users.servic... M
TS users.servic... M

src > app > home > <> home.component.html > app-footer

```
2
3 <app-home-panel></app-home-panel>
4
5 <app-search-panel></app-search-panel>
6
7 <app-collab-panel></app-collab-panel>
8
9
10 <app-footer></app-footer>
11
12
```

src > app > footer > TS footer.component.ts > FooterComponent

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-footer',
5   standalone: true,
6   imports: [],
7   templateUrl: './footer.component.html',
8   styleUrls: ['./footer.component.css']
9 })
10 export class FooterComponent {
11
12   //logik hier
13
14 }
15
```



Applikation Entwurf/ UML-Diagramm

ANGULAR MATERIAL UI COMPONENT LIBRARY

Was wir benutzt haben

- Buttons
- Dialogs
- Headers/Menu Bars
- Menus
- Lists
- SVG-Icons
- Forms
- Bottom Sheet für mobile Geräte
- Eine Liste und Anleitung für alle ui Komponenten
Bibliothek finden sie hier:
<https://material.angular.io/components/categories>


```

ngOnInit(): void {
  if (!localStorage.getItem('lightMode')) {
    // Initialize theme based on system preference
    this.themeService.setThemeBasedOnSystemPreference();
    console.log('component match');
  }
}

```

```

setThemeBasedOnSystemPreference(): void {
  const prefersDark = window.matchMedia(
    '(prefers-color-scheme: dark)'
  ).matches;
  this.currentMode.lightmode = !prefersDark;
  this.currentMode.darkmode = prefersDark;
  console.log("service match")
  console.log('prefersDark: ' + prefersDark);
  localStorage.setItem(
    this.lightModeKey,
    this.currentMode.lightmode.toString()
  );
}







```

THEME SERVICE

- Verwendung eines Angular Service zur Steuerung des Farbwechsels in der Anwendung.
- Speichern des ausgewählten Themas lokal im Browser, um die Präferenzen des Benutzers zu speichern
- Lesen des Systemthemenmodus, um das Farbthema entsprechend anzupassen `@media (prefers-color-scheme: dark) {}`

LOCAL STORAGE IMPLEMENTATION

- Benutzer können Dateien lokal erstellen, speichern und bearbeiten, ohne ein Konto anzulegen.
- Benutzer muss ein Konto einrichten, um Dateien auf dem Server zu speichern/hochladen.
- Jede Datei hat eine eindeutige server_id und eine entsprechende local_id. Jede Datei hat außerdem eine Versionsnummer, die angibt, welche Datei die neuere ist.
- wenn der Benutzer sich einloggt, werden seine Informationen lokal gespeichert und entfernt, wenn er auf „Logout“ klickt

▶  Cache-Speicher	🔍 Einträge durchsuchen	
▶  Cookies	Key	Value
▶  Indexed DB	lightMode	false
▼  Local Storage	personalFiles	[{"localFileId":null,"serverFileId":"669cda828f230c0634dd2f0e","fileName":"hghg","version":0,"con
 http://localhost:4200	user	{"userId":"669cb0c31824c6ccf4892c5c","username":"mono","password":"Mnimma23900@","local
▶  ...	username	mono

```

{
  path: 'home',
  component: HomeComponent,
  title: 'Home page',
},
{
  path: '',
  redirectTo: '/home',
  pathMatch: 'full',
},
{
  path: 'editor',
  component: EditorComponent,
  title: 'Text Editor',
},

```

src > app > <> app.component.html >  div

Go to component

```

1 <div [class]="themeService.getCurrentModeClass()">
2
3 <router-outlet ></router-outlet>
4
5 </div>

```

```

this.route.params.subscribe((params) => {
  if(params['id']) {
    this.id = params['id'];
  }
}

```

ROUTING

- Einfaches Routing mit dem in Angular eingebauten Router zu implementieren
- Wir können unseren Routen Parameter hinzufügen und die Parameter beim Laden lesen

ANGULAR FORMS

- Angular hat eine eigene Funktionalität für Eingabeformulare, die wir nutzen können. Es kommt mit Funktionen wie:
 - **Reaktive** und Template-gestützte Formulare: Angular bietet zwei Ansätze für die Formularerstellung, die sowohl Flexibilität als auch Kontrolle ermöglichen.
 - **Datenbindung**: Bidirektionale Datenbindung sorgt dafür, dass Änderungen im Formularfeld und im Modell synchronisiert werden.
 - **Formularvalidierung**: Umfangreiche Validierungsoptionen, sowohl eingebaut als auch benutzerdefiniert, stellen sicher, dass Formulareingaben den Anforderungen entsprechen.
 - **Formulargruppen und -steuerelemente**: Ermöglichen die strukturierte Verwaltung von Formularen mit verschachtelten Gruppen und Steuerelementen.
 - Asynchrone Validierung: Unterstützung für serverseitige und zeitaufwändige Validierungen, die auf Benutzerinteraktionen reagieren.

```

<form [formGroup]="createForm" (ngSubmit)="createAccount()" class="accountForm">
  <h2>Create an Account</h2>
  <div>
    <mat-form-field >
      <input formControlName="username" placeholder="Username" required matInput />
      <mat-error *ngIf="username.hasError('userNotAvailable') && (username.dirty || username.touched)">Username is already
      <mat-error *ngIf="username.hasError('required') && (username.dirty || username.touched)">Username is required</mat-error>
    </mat-form-field>
  </div>
  <div>
    <mat-form-field>
      <input type="password" formControlName="password1" placeholder="Password" required matInput />
      <mat-error>Password should be Strong</mat-error>
    </mat-form-field>
  </div>
  <div>
    <mat-form-field>
      <input type="password" formControlName="password2" placeholder="Retype Password" required matInput />
      <mat-error *ngIf="createForm.hasError('passwordsMismatch')">Passwords do not match</mat-error>
    </mat-form-field>
  </div>
  <br>
  <div>
    <button style="background-color: var(--akzent);color: snow" mat-raised-button color="primary"
      type="submit" [disabled]="!createForm.valid">Create Account</button>
  </div>
  <a routerLink="/home">Home</a>
  <a routerLink="/login">login</a>
</form>

```

```

createForm = new FormGroup(
{
  username: new FormControl('', {
    nullable: true,
    validators: [Validators.required],
  }),
  password1: new FormControl('', {
    nullable: true,
    validators: [Validators.required, passwordValidator()],
  }),
  password2: new FormControl('', {
    nullable: true,
    validators: [Validators.required, passwordValidator()],
  }),
},
{ validators: passwordsMatch('password1', 'password2') }
);

```

Form group zur Steuerung des Formulars

```

import { AbstractControl, ValidationErrors, ValidatorFn } from '@angular/forms';

export function passwordsMatch(password: string, confirmPassword: string): ValidatorFn {
  return (control: AbstractControl): ValidationErrors | null => {
    const passwordControl = control.get(password);
    const confirmPasswordControl = control.get(confirmPassword);

    if (!passwordControl || !confirmPasswordControl) {
      return null; // controls are not yet defined
    }

    const passwordValue = passwordControl.value;
    const confirmPasswordValue = confirmPasswordControl.value;

    return passwordValue === confirmPasswordValue ? null : { passwordsMismatch: true };
  };
}

```

Benutzerdefinierten Validator erstellen

OBSERVABLES

Observables aus der RxJS-Bibliothek sind Datenströme, die asynchrone Ereignisse verwalten. In Angular helfen sie, indem sie eine effiziente Möglichkeit bieten, mit Datenquellen wie HTTP-Anfragen, Benutzerereignissen oder WebSocket-Nachrichten umzugehen. Observables ermöglichen es, auf einfache Weise auf Datenänderungen zu reagieren und komplexe asynchrone Operationen zu verketteten. Sie unterstützen auch Funktionen wie Fehlerbehandlung und das erneute Abonnieren, was die Verwaltung von Echtzeit-Datenflüssen vereinfacht. Durch die Integration von Observables in Angular können Entwickler reaktive und wartbare Anwendungen erstellen.

```
getFileFromServer(server_file_id: string | null): Observable<File> {  
  return this.http  
    .get<File>(`${this.api}/files/${server_file_id}`)  
    .pipe(catchError(this.errorHandler));  
}
```

```
public socket$: WebSocketSubject<any>;
```

```
onMessage(): Observable<any> {  
  return this.socket$.asObservable().pipe();  
}
```

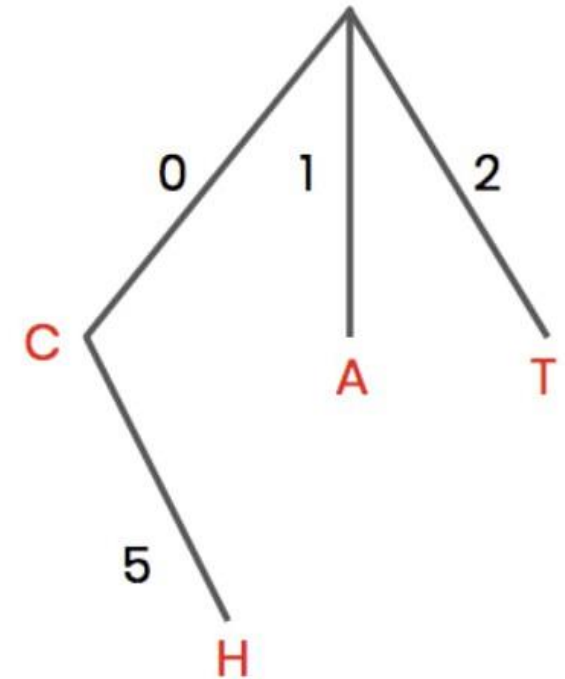
```
send(message: any) {  
  console.log('message: ', message);  
  this.socket$.next(message);  
}
```

```
createFile(file: File): Observable<File> {  
  const body = {  
    localFileId: file.localFileId,  
    serverFileId: file.serverFileId,  
    fileName: file.fileName,  
    version: file.version,  
    content: file.content,  
    tree: file.tree  
  };  
  return this.http  
    .put<File>(`${this.api}/files`, body)  
    .pipe(catchError(this.errorHandler));  
}
```

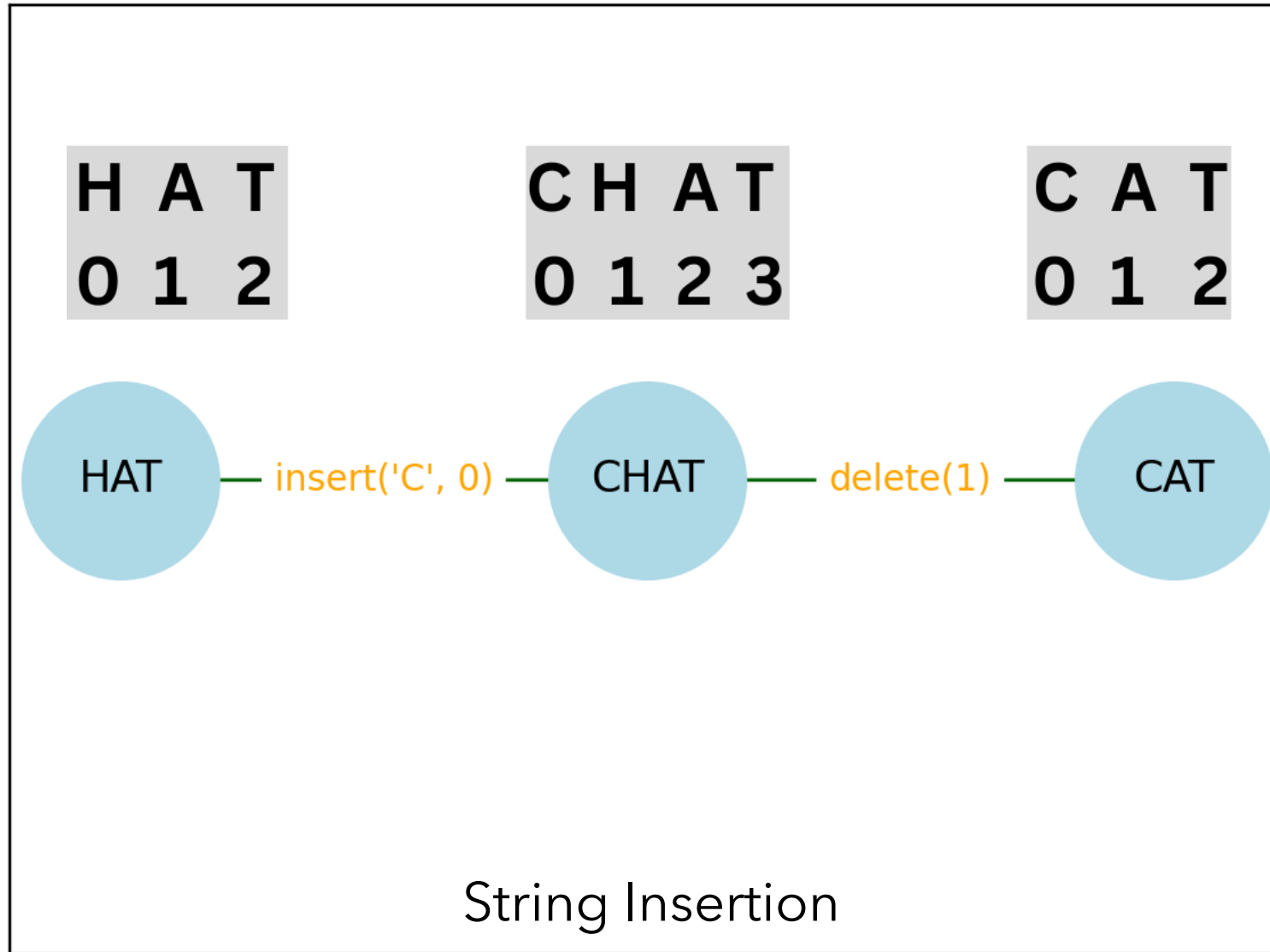
```
updateFileInDB(file: File): Observable<any> {  
  const body = {  
    localFileId: file.localFileId,  
    serverFileId: file.serverFileId,  
    fileName: file.fileName,  
    version: file.version,  
    content: file.content,  
    tree: file.tree  
  };  
  return this.http  
    .post<any>(`${this.api}/files`, body)  
    .pipe(catchError(this.errorHandler));  
}
```

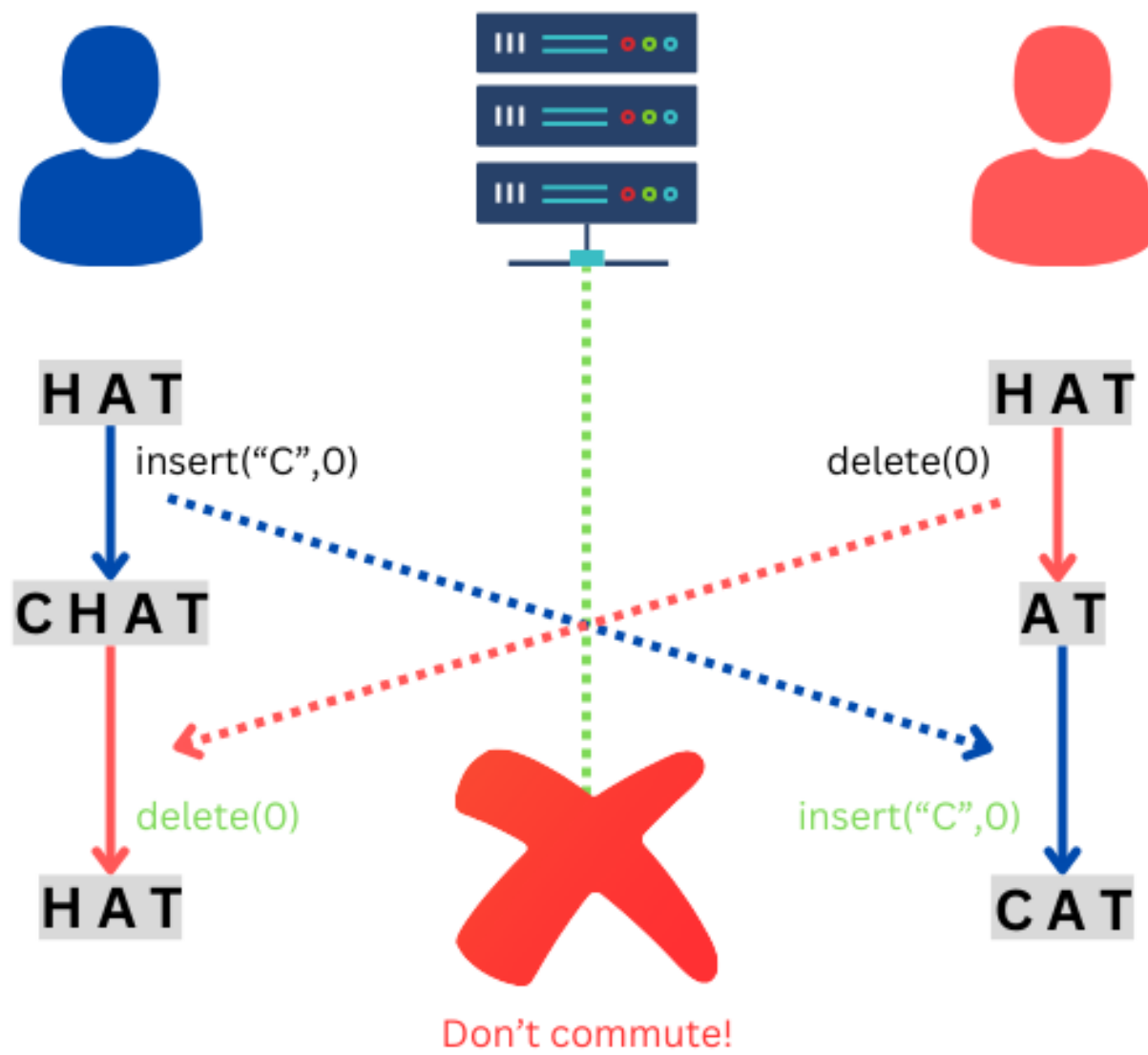

CRDT

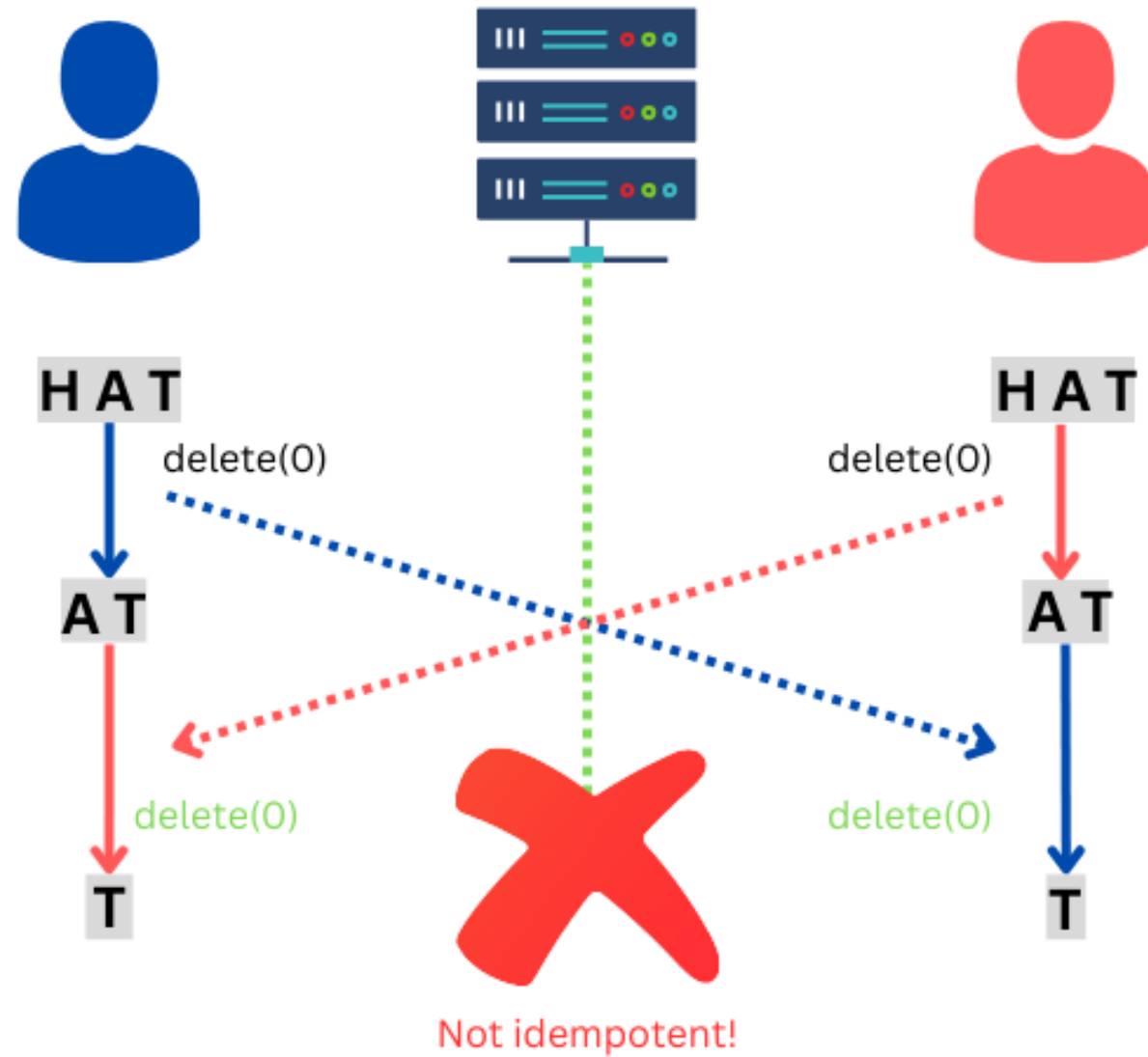
- CRDTs (Conflict-free Replicated Data Types) sind spezielle Datenstrukturen für verteilte Systeme. Sie ermöglichen gleichzeitige Bearbeitungen ohne zentrale Koordination. CRDTs lösen Konflikte automatisch und gewährleisten Konsistenz zwischen Replikaten. Sie werden oft in kollaborativen Anwendungen und verteilten Datenbanken eingesetzt.
- Wir haben vier Funktionen von crdt: lokales Einfügen(localInsert), lokales Löschen(localDelete), entferntes Einfügen(remoteInsert), entferntes Löschen(remoteDelete)

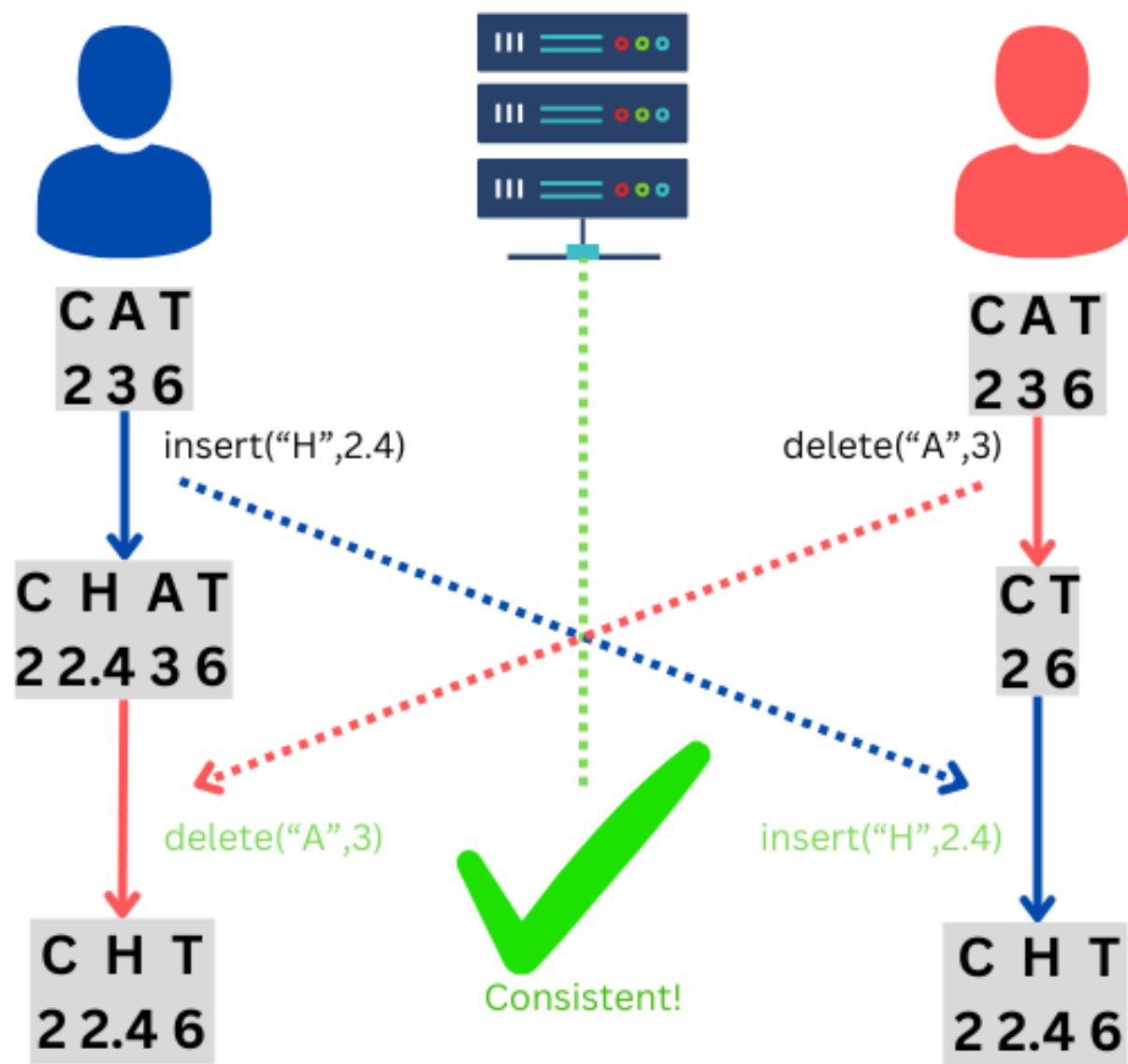


String Manipulation Operations









-
- CRDT Data Structure auch als „LSEQ: an Adaptive Structure for Sequences in Distributed Collaborative Editing“ genannt:
<https://hal.science/hal-00921633/document>



NODE SERVER UND WEBSOCKETS

- Node.js: JavaScript-Laufzeitumgebung für serverseitige Ausführung.
- Express.js: Web-Anwendungsframework für Node.js, vereinfacht die Servererstellung.
- WebSocket (WS) in Node.js: Eine Technologie für bidirektionale Echtzeitkommunikation zwischen Client und Server. Sie ermöglicht eine dauerhafte Verbindung, über die beide Seiten jederzeit Daten senden können, ohne wiederholte HTTP-Anfragen.

An abstract network diagram on a black background. It features several nodes, represented by small white circles with black outlines, some of which have concentric circles. These nodes are interconnected by a web of thin, curved lines in red, green, and orange. The lines create a sense of depth and movement, with some lines appearing to converge towards the bottom right corner. The overall composition is dynamic and complex.

ENDE

GITHUB: [HTTPS://GITHUB.COM/MONOMARKOR/COLLABARATIVENOTES](https://github.com/monomarkor/collabarativenotes)