To-Do List (app.component.ts)

```
@Component({
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  imports: [CommonModule, ItemComponent],
})
```

Hier werden die Sachen wie etwa HTML code und CSS geladen, das ist von Angular.

```
export class AppComponent implements OnInit {
  componentTitle = "My To Do List";
  componenquestion = "What would you like to do today?"
  filter: "all" | "active" | "done" = "all";

  allItems: Item[] = [];
  title: any;
```

Hier wird eine neue Klasse mit dem Namen **AppComponent** erstellt und durch das Schlüsselwort **export** für den Export in andere Dateien bereitgestellt. Später werde ich auf die vielen Funktionen eingehen, die darin enthalten sind, aber zunächst auf die grundlegenden Elemente oben im Code. Hier oben werden Dinge wie die zwei Titel erstellt, die dank Angular in das HTML eingefügt werden. Auch die Filter, welche das Anordnen der Todos regeln, werden hier definiert. Zuunterst wird ein spezielles Array erstellt, das **allItems** (also alle Items) enthält und für die Funktionen verwendet wird. Das bedeutet, dass dein neues To-do als Item hinzugefügt und in **allItems** gespeichert wird.

```
ngOnInit() {
  this.loadItems();
}
```

Das ist die Oberste Funktion in der Classe was sie aber ausführt wird zuunterst definiert. Diese Funktion lädt alle Items, wenn man die Seite neu lädt.

```
addItem(description: string) {
  if (!description) return;

  this.allItems.unshift({
    description,
    done: false,
    comments: []
  });
  this.saveItems();
}
```

Das ist eine der wichtigsten Funktionen, die in TypeScript geschrieben wurde. Die **addItem**-Funktion wird aufgerufen und erhält den Namen des To-dos, z. B. "WC putzen". Dann wird dank TypeScript überprüft, ob es sich um einen String handelt – wenn ja, wird der Prozess fortgesetzt, andernfalls nicht. Anschließend wird das Item mit **unshift** an den Anfang des zuvor erstellten Arrays hinzugefügt. Das Item erhält eine **description**, die ihm oben zugewiesen wurde. Da das Item beim Erstellen noch nicht fertiggestellt ist, wird **done** standardmäßig auf **false** gesetzt. Unten wird ein Array für die Kommentare erstellt, welches erst nach dem Hinzufügen des To-dos verwendet werden kann. Am Schluss wird die **saveItems**-Methode mit **this** aufgerufen, da sie in derselben Klasse liegt.

```
remove(item: Item) {
  this.allItems.splice(this.allItems.indexOf(item), 1);
  this.saveItems();
}
```

Hier sieht man eine einfache Löschfunktion, der das zu löschende Item über den Index mitgegeben wird. Der Funktion wird der Index des Items übergeben, sodass sie weiß, welches Element aus dem **allItems**-Array entfernt werden muss. Am Ende speichert sie die aktualisierten Items erneut.

```
addComment(item: Item, comment: string) {
  if (!comment) return;
  item.comments.push(comment);
  this.saveItems();
}
```

Nachdem diese Funktion aufgerufen wird, wird ein Kommentar zum jeweiligen To-do hinzugefügt. Wie bei der Löschfunktion benötigt auch diese den Index des Items, um den Kommentar korrekt hinzuzufügen. Zusätzlich muss der Kommentar als String (also in Textform) übergeben werden. Es wird, wie bei der Hinzufüg-Funktion, überprüft, ob eine Eingabe vorliegt. Wenn der Benutzer etwas eingibt, wird der Kommentar in das zuvor erstellte Array eingefügt, wobei der gesamte Array ersetzt wird. Dies funktioniert gut, da es sich um einen Kommentar pro Item handelt. Beim Haupt-Array mit den To-dos würde diese Methode nicht funktionieren, da sonst alle anderen Einträge gelöscht würden. Zum Schluss wird erneut die **saveItems**-Funktion aufgerufen, um die Änderungen zu speichern.

```
get items() {
  if (this.filter === "all") {
    return this.allItems;
  }
  return this.allItems.filter((item) =>
    this.filter === "done" ? item.done : !item.done
  );
}
```

Hier sehen wir die **get items()**-Methode, die verwendet wird, um eine gefilterte Liste von Items zurückzugeben. Wenn der Filter auf **"all"** gesetzt ist, gibt die Methode einfach das gesamte **allItems**-Array zurück. Wenn der Filter jedoch auf **"done"** oder **"not done"** gesetzt ist, filtert die Methode die Items entsprechend. Bei einem Filterwert von **"done"** werden nur die Items zurückgegeben, bei denen das **done**-Flag auf **true** steht. Bei einem Filterwert von **"not done"** werden nur die Items zurückgegeben, bei denen das **done**-Flag auf **false** steht. Auf diese Weise ermöglicht die Methode eine dynamische Auswahl der anzuzeigenden Items basierend auf dem aktuellen Filterkriterium.

```
saveItems() {
  localStorage.setItem('todo-items', JSON.stringify(this.allItems));
}
```

Hier sehen wir eine sehr nützliche Funktion, die es ermöglicht, die Items im Browser so zu speichern, dass sie auch nach einem Seitenreload oder Browserneustart noch vorhanden sind. Dazu wird das Array mit den Items in einen String umgewandelt und im Local Storage des Browsers unter dem Schlüssel **"todo-items"** gespeichert. Diese Funktion wurde bereits weiter oben erwähnt, beispielsweise bei der Kommentar-Hinzufügen-Funktion, wo sie verwendet wird, um die To-dos zusammen mit den Kommentaren im Browser zu speichern.

```
  loadItems() {
    const storedItems = localStorage.getItem('todo-items');
    if (storedItems) {
      this.allItems = JSON.parse(storedItems);
    }
  }
}
```

Dies ist die finale Funktion, die die To-dos aus dem Browser lädt und anzeigt. Sie wird nicht durch einen Button oder eine andere Benutzeraktion ausgeführt, sondern durch die **ngOnInit()**-Methode, die beim Laden der Seite automatisch aufgerufen wird.