

Namen: \_\_\_\_\_

Note: \_\_\_\_\_

# Projekt Auswertung

## Einleitung:

---

Sie zeigen anhand Ihrer Umsetzung, dass Sie das Projekt gemäss den Anforderungen umgesetzt haben.

Leitlinien sind die Anforderungen, Akzeptanzkriterien und das Bewertungsraster aus 30.1\_ProjektM450&M324-Projekt.pdf

Sie zeigen damit Ihre Essenz der Lösung.

## Ziel

---

- ⇒ Sie können Ihr Projekt auswerten und aufzeigen, wie Sie die Anforderungen umgesetzt haben.
- ⇒ Sie können sich damit selbst beurteilen.

## Inhalte

---

Einleitung: .....	1
Ziel .....	1
Inhalte .....	1
Umsetzung und Ergebnis der User Story 1 .....	2
Umsetzung und Ergebnis der User Story 2 .....	5
Umsetzung und Ergebnis der User Story 3 .....	8
Umsetzung Automatisierung (324).....	9
Umsetzung Testen (450) .....	14

## Vorgehen:

---

Die folgenden Kapitel basieren auf den Anforderungen, Akzeptanzkriterien und dem Bewertungsraster aus 30.1\_ProjektM450&M324-Projekt.pdf.

Lesen Sie Kapitel und Auftrag sorgfältig durch.

Lösen Sie den Auftrag. Belassen Sie die Struktur dieses Dokumentes. Halten Sie nur die Essenz fest.

Dieses Dokument und allfällig weitere Dokumente sowie git-Repo als eine ZIP-Datei abgeben.  
Pro Gruppe ein Dokument.

## **Umsetzung und Ergebnis der User Story 1**

---

### **User Story 1:**

1. Der Benutzer kann beim Erstellen einer Aufgabe eine Priorität auswählen (hoch, mittel, niedrig).
2. Die Aufgabenliste zeigt die Priorität jeder Aufgabe an.
3. Der Benutzer kann die Priorität einer bestehenden Aufgabe ändern.
4. Die Aufgaben können nach Priorität sortiert werden, sodass "hohe" Priorität ganz oben erscheinen.

Zeigen Sie mit einzelnen Screenshots (zBsp Frontend) zu den einzelnen Akzeptanzkriterien Ihre Umsetzung.

\* Task Title

HIGHHHHHHHH

\* Priority

high

medium

low

\* Category

General

\* Due Date

30-01-2025

Add Task

1.

Titel	Category	Priority	Todo u...	Edit	Done
Aufräumen	General	high	1. Sep 2021	Edit	Done
Einkaufen	General	medium	2. Sep 2021	Edit	Done
Znacht ko...	General	low	3. Sep 2021	Edit	Done
HIGHHHHHH	General	high	30. Jan 2025	Edit	Done

2.

\* Task Title



HIGHHHHH

\* Priority

high

medium

low

\* Category

General

\* Due Date

30-01-2025



Update Task

3.

HIGHHHHH

General

low

30. Jan 2025

Edit

Done

4.

Titel	Category	Priority	Todo u...	Edit	Done
Znacht ko...	General	low	3. Sep 2021	Edit	Done
HIGHHHHH	General	low	30. Jan 2025	Edit	Done
Einkaufen	General	medium	2. Sep 2021	Edit	Done
Aufräumen	General	high	1. Sep 2021	Edit	Done

Titel	Category	Priority	Todo u...	Edit	Done
Aufräumen	General	high	1. Sep 2021	Edit	Done
Einkaufen	General	medium	2. Sep 2021	Edit	Done
Znacht ko...	General	low	3. Sep 2021	Edit	Done
HIGHHHHH	General	low	30. Jan 2025	Edit	Done

(2nd is for top = "high")

## Umsetzung und Ergebnis der User Story 2

### User Story 2:

1. Der Benutzer kann beim Erstellen einer Aufgabe eine Kategorie auswählen oder eine neue erstellen.
2. Jede Aufgabe zeigt die zugewiesene Kategorie an.
3. Der Benutzer kann bestehende Aufgaben einer anderen Kategorie zuordnen.
4. Der Benutzer kann in der Ansicht nach Kategorien filtern, um nur Aufgaben einer bestimmten Kategorie zu sehen.

Zeigen Sie mit einzelnen Screenshots (zBsp Frontend) zu den einzelnen Akzeptanzkriterien Ihre Umsetzung.

\* Task Title



Task Title

\* Priority

high

medium

low

\* Category

Category

General

Work

Personal

Shopping

Others

1.

Titel	Category	Priority	Todo u...	Edit	Done
Aufräumen	General	high	1. Sep 2021	Edit	Done
Einkaufen	General	medium	2. Sep 2021	Edit	Done
Znacht ko...	General	low	3. Sep 2021	Edit	Done
TOdo Now	General	medium	20. Jan 2025	Edit	Done

2.

3.

all General Work Personal Shopping Others after clicking on edit i make my own category						
Titel	Category	Priority	Todo u...	Edit	Done	
Aufräumen	General	high	1. Sep 2021	Edit	Done	
Einkaufen	after clicki...	medium	2. Sep 2021	Edit	Done	

4.

all General Work Personal Shopping Others after clicking on edit i make my own category						
Titel	Category	Priority	Todo u...	Edit	Done	
Einkaufen	after clicki...	medium	2. Sep 2021	Edit	Done	

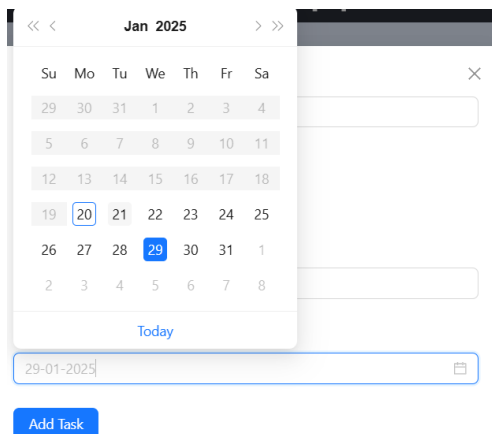
## Umsetzung und Ergebnis der User Story 3

### User Story 3:

1. Der Benutzer kann beim Erstellen einer Aufgabe ein Fälligkeitsdatum auswählen.
2. Jede Aufgabe zeigt ihr Fälligkeitsdatum an.
3. Aufgaben werden farblich markiert, wenn das Fälligkeitsdatum innerhalb der nächsten 24 Stunden liegt.
4. Der Benutzer kann Aufgaben nach Fälligkeitsdatum sortieren, sodass die Aufgaben mit nahen Deadlines oben stehen.

### Auftrag:

Zeigen Sie mit einzelnen Screenshots (zBsp Frontend) zu den einzelnen Akzeptanzkriterien Ihre Umsetzung.



1.

Titel	Category	Priority	Todo until	Edit	Done
Aufräumen	General	high	1. Sep 2021	Edit	Done
Einkaufen	General	medium	2. Sep 2021	Edit	Done
Znacht kochen	General	low	3. Sep 2021	Edit	Done

2.

3.

Titel	Category	Priority	Todo u...	Edit	Done
Aufräumen	General	high	1. Sep 2021	Edit	Done
Einkaufen	General	medium	2. Sep 2021	Edit	Done
Znacht ko...	General	low	3. Sep 2021	Edit	Done
TOdo Now	General	medium	20. Jan 2025	Edit	Done



4.

Titel	Category	Priority	Todo until	Edit	Done
Aufräumen	General	high	1. Sep 2021	Edit	Done
Einkaufen	General	medium	2. Sep 2021	Edit	Done

Titel	Category	Priority	Todo until	Edit	Done
Einkaufen	General	medium	2. Sep 2021	Edit	Done
Aufräumen	General	high	1. Sep 2021	Edit	Done

## Umsetzung Automatisierung (324)

### CI/CD-Pipeline automatisiert Build

1. Jeder Schritt in der GitHub Actions-Konfiguration (YAML-Datei) muss kommentiert sein.
2. Struktur der Jobs und Abhängigkeiten zwischen den Prozessen
3. Sensible Daten wie API-Keys sollen sicher mit GitHub Secrets verwaltet werden.
4. Fehlgeschlagene Builds sollen die Pipeline stoppen und eine Fehlermeldung zurückgeben.
5. Nach erfolgreichem Testen soll die Pipeline die App automatisch in eine Staging-Umgebung deployen.
6. Die Funktionsweise der Pipeline wird kurz im Projektbericht dokumentiert.

### Testintegration

7. Automatisierte Tests sind in die Pipeline integriert und dokumentieren Ergebnisse.
8. Alle Unit-Tests und Integrationstests werden automatisch ausgeführt.
9. Die Tests müssen beim Durchlaufen der Pipeline mindestens 80 % der definierten Testfälle bestehen.

### Code Qualität / Linter

10. Linter meldet Probleme
11. fehlerhafter Code wird nicht in den Hauptbranch gemerged.

### Versionskontrolle / Git

12. Systematische Git-Nutzung mit sinnvollen Commits, Branches
13. Vermeidung ungetesteter Merges.

### Auftrag:

Zeigen Sie mit einzelnen Screenshots zu den einzelnen Anforderungen Ihre Umsetzung. Vermerken Sie, wenn Sie eine Position nicht umgesetzt haben.

```
build:
  runs-on: ubuntu-latest # Verwenden des neuesten Ubuntu-Images
  needs:
    - e2e-tests # Abhängigkeit von e2e-tests
    - unit-tests # Abhängigkeit von unit-tests

  steps:
    - name: Checkout code
      uses: actions/checkout@v2 # Code aus dem Repository auschecken

    - name: Set up Node.js
      uses: actions/setup-node@v2 # Node.js einrichten
      with:
```

1.

```
build:
  runs-on: ubuntu-latest # Verwenden des neuesten Ubuntu-Images
  needs:
    - e2e-tests # Abhängigkeit von e2e-tests
    - unit-tests # Abhängigkeit von unit-tests
```

2.

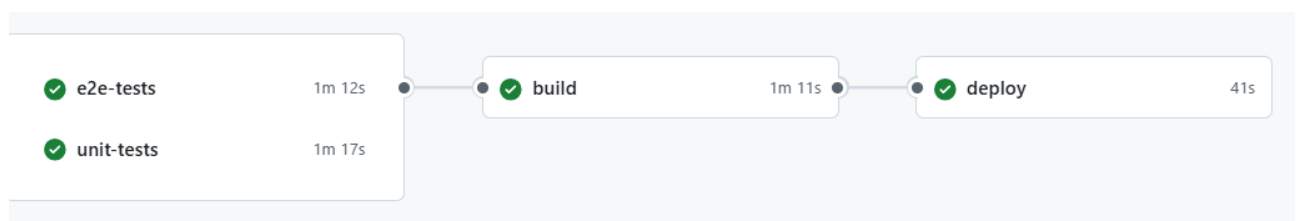
3. Keine keys

### ✖ fix all - final push v2

Deploy to GitHub Pages #39: Commit [608a3ad](#) pushed by SeverinFux

(needs)

4.



5.

```
testkonzept.txt  deploy.yml  README.md x
```

### 1. End-to-End Tests ( e2e-tests )

- **Runs on:** ubuntu-latest
- **Steps:**
  - Checkout code from the repository.
  - Set up Node.js version 16.
  - Install project dependencies.
  - Start the application in the background.
  - Wait for the server to be ready.
  - Run Cypress end-to-end tests.

### 2. Unit Tests ( unit-tests )

- **Runs on:** ubuntu-latest
- **Steps:**
  - Checkout code from the repository.
  - Set up Node.js version 16.
  - Install project dependencies.
  - Start the application in the background.
  - Wait for the server to be ready.
  - Run Jest unit tests.

### 3. Build ( build )

- **Runs on:** ubuntu-latest
- **Needs:** e2e-tests , unit-tests
- **Steps:**
  - Checkout code from the repository.
  - Set up Node.js version 16.
  - Install project dependencies.
  - Build the project.

### 4. Deploy ( deploy )

- **Runs on:** ubuntu-latest
- **Needs:** build
- **Steps:**
  - Checkout code from the repository.
  - Set up Node.js version 16.
  - Install project dependencies.
  - Deploy the project to GitHub Pages.
  - Merge the current branch into the target branch.

6.

```

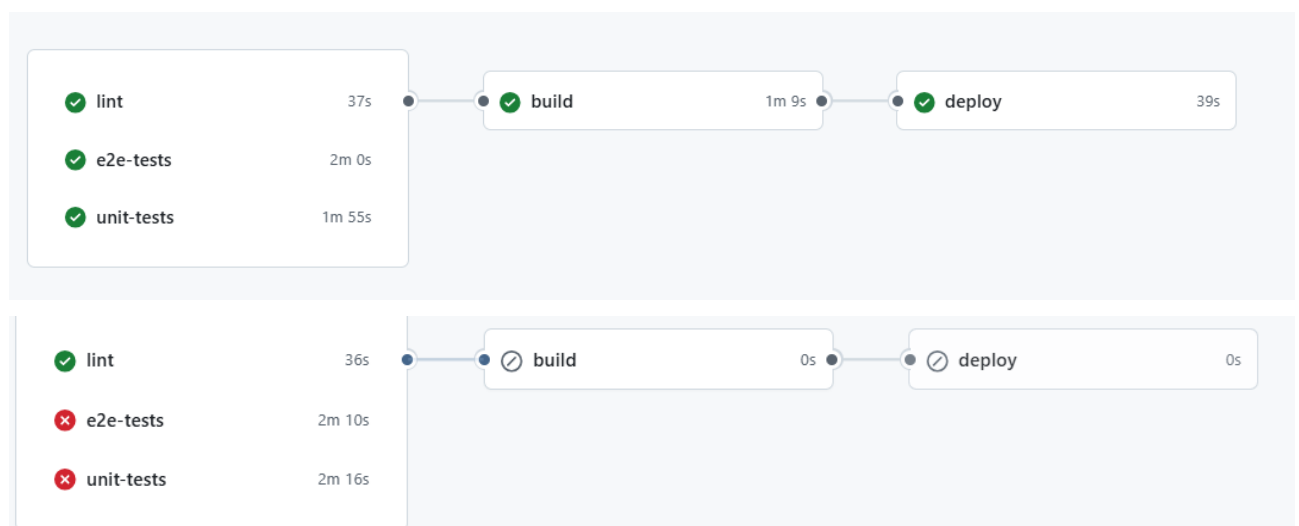
e2e-tests
succeeded 4 minutes ago in 1m 12s

Run Cypress tests

30
31
32 Running:  todo.cy.ts (1 of 1)
33
34
35 TODO App Tests
36   ✓ should render the TODO-App header (1914ms)
37   ✓ add a new TODO item (1737ms)
38   ✓ mark a TODO item as completed (414ms)
39   ✓ sort TODOs by priorities (306ms)
40   ✓ sort TODOs by Category and edit to get custom Category (1838ms)
41   ✓ sort TODOs by Date (606ms)
42
43
44 6 passing (7s)
45
46
47 (Results)
48
49
50 | Tests:      6
51 | Passing:    6
52 | Failing:    0
53 | Pending:    0
54 | Skipped:    0
55 | Screenshots: 0
56 | Video:      false
57 | Duration:   6 seconds
58 | Spec Ran:   todo.cy.ts
59
60

```

7.



9.

(ON FAIL CANCEL)

10.

```
> ✓ Set up Node.js
> ✓ Install dependencies
v ✓ Run ESLint

1 ▶ Run npm run lint
4
5 > todo-app@0.1.0 lint
6 > eslint .
7
```

11.

main-requests
4 branch rules • targeting 2 branches
...

Branch targeting criteria

Add target

- Default
- develop

Applies to 2 targets including `develop`, and `main`.

12.

⚠ Push rejected

Push main to origin/main was rejected by the remote

[View received commit](#) [More](#)

13. Rejected by ruleset 😊

## Umsetzung Testen (450)

---

### Testkonzept:

1. Vollständiges Testkonzept mit klarer Beschreibung von Testarten, Zielen und Fällen.
2. Erstellen eines Testkonzepts mit Definition der Testarten (z.B. Unit-Tests, Integrationstests, E2E-Tests).-
3. Dokumentation der Testfälle mit Beschreibung von Vorbedingungen, Eingaben, erwarteten Ergebnissen und Nachbedingungen.

### Testabdeckung:

4. Mindestens 80 % der User-Story-Anforderungen sind durch Tests abgedeckt.

### Testumsetzung:

5. Automatisierte Tests sind korrekt implementiert und führen nachvollziehbare Ergebnisse auf.
6. Implementierung und Ausführung automatisierter Tests mit einem Testframework (z.B. Jest).
7. Nutzung von Mocking-Tools oder Stubs, um Abhängigkeiten in Integrationstests zu simulieren.

### Testergebnis

8. Testergebnisse und Fehler sind dokumentiert (Protokoll, Analyse).
9. Nachvollziehbare Dokumentation der Testergebnisse, z.B. in Testprotokollen.

### Auftrag:

Geben Sie Ihr **Testkonzept** ab. Verweisen Sie hier auf das konkrete Dokument.

Testkonzept.md

Geben Sie Ihre **Testprotokolle** ab. Verweisen Sie hier auf das konkrete Dokument. (neben den Tests z.B unter E2ETEST.md)

Zeigen Sie **Testabdeckung** und **Testumsetzung** mit Screenshots und kurzen Erklärungen.

1.

# Testkonzept für TODO-App

## Überblick

Dieses Dokument beschreibt die Teststrategie für die TODO-App, einschliesslich automatisierter und manueller Testansätze, um die Qualität und Zuverlässigkeit der Anwendung sicherzustellen.

## Automatisierte Tests

### Unit-Tests

Unit-Tests werden mit Jest geschrieben, um die Funktionalität einzelner Komponenten und Utility-Funktionen zu überprüfen.

- Ort: `Util.test.ts`
- Umfang:
  - `sortByString` : Testet das alphabetische Sortieren von Strings.
  - `sortByNumber` : Testet das Sortieren von Zahlen in aufsteigender Reihenfolge, inklusive der Behandlung negativer und identischer Zahlen.
  - `sortByDate` : Testet das Sortieren von Daten in aufsteigender Reihenfolge, inklusive der Behandlung identischer Daten und verschiedener Zeiten.

### End-to-End-Tests

End-to-End-Tests werden mit Cypress geschrieben, um Benutzerinteraktionen zu simulieren und das Verhalten der Anwendung aus der Perspektive der Benutzer zu überprüfen.

- Ort: `cypress/e2e/todo.cy.ts`
- Umfang:
  - Darstellung des TODO-App-Headers.
  - Hinzufügen eines neuen TODO-Items.
  - Markieren eines TODO-Items als abgeschlossen.
  - Sortieren von TODOs nach Prioritäten, Kategorien und Daten.
  - Bearbeiten von TODO-Items zur Erstellung benutzerdefinierter Kategorien.

### Linting

ESLint wird verwendet, um die Codequalität und die Einhaltung von Coding-Standards sicherzustellen.

## Linting

ESLint wird verwendet, um die Codequalität und die Einhaltung von Coding-Standards sicherzustellen.

- Ort: `.github/workflows/deploy.yml`
- Umfang:
  - Ausführen von ESLint zur Überprüfung von Stil- und Syntaxfehlern.

## Manuelle Integrationstests

Manuelle Integrationstests werden durchgeführt, um die Integration der verschiedenen Komponenten und die Gesamtfunktionalität der Anwendung zu überprüfen.

### Testszenarios

1. **Benutzerauthentifizierung:**
  - Überprüfen, ob sich Benutzer erfolgreich registrieren, einloggen und ausloggen können.
  - Sicherstellen, dass Benutzersitzungen korrekt aufrechterhalten werden.
2. **Verwaltung von TODO-Items:**
  - Überprüfen, ob Benutzer TODO-Items hinzufügen, bearbeiten und löschen können.
  - Sicherstellen, dass TODO-Items korrekt in der Liste angezeigt werden.
3. **Sortieren und Filtern:**
  - Überprüfen, ob TODO-Items nach Priorität, Kategorie und Datum sortiert werden können.
  - Sicherstellen, dass Filteroptionen wie erwartet funktionieren.
4. **Benachrichtigungen:**
  - Überprüfen, ob Benutzer Benachrichtigungen für bevorstehende und überfällige Aufgaben erhalten.
5. **Responsives Design:**
  - Sicherstellen, dass die Anwendung responsiv ist und auf verschiedenen Geräten und Bildschirmgrößen gut funktioniert.

## Fazit

Dieses Testkonzept gewährleistet eine umfassende Abdeckung der TODO-App durch eine Kombination aus automatisierten Unit- und End-to-End-Tests sowie manuellen Integrationstests. Dieser Ansatz hilft, Probleme frühzeitig zu identifizieren und zu beheben, und sorgt für eine qualitativ hochwertige und zuverlässige Anwendung.

[Markdown Split Editor](#)
[Markdown Editor](#)

## 2. (mit 1. )

Testfall	Beschreibung	Vorbedingungen	Eingaben	Erwartetes Ergebnis	Nachbedingungen
Darstellung des Headers	Überprüft, ob der Header der Anwendung korrekt angezeigt wird.	Anwendung gestartet.	Keine	Header zeigt den Text "TODO-App".	Keine
Neue Aufgabe hinzufügen	Testet das Hinzufügen einer Aufgabe mit Priorität, Kategorie und Fälligkeitsdatum.	Anwendung gestartet, Eingabefelder sichtbar.	Titel, Priorität, Kategorie, Fälligkeitsdatum	Aufgabe erscheint in der Liste mit den korrekten Attributen.	Aufgabe korrekt in der Liste sichtbar.
Sortieren nach Priorität	Testet die Sortierung der Aufgaben nach Priorität (hoch, mittel, niedrig).	Aufgaben in der Liste mit verschiedenen Prioritäten.	Klick auf "Sortieren nach Priorität"	Aufgaben werden in der Reihenfolge der Priorität angezeigt.	Aufgabenliste sortiert nach Priorität.
Aufgabe als erledigt markieren	Testet, ob eine Aufgabe korrekt als erledigt markiert und aus der Liste entfernt wird.	Aufgabe in der Liste vorhanden.	Klick auf "Erledigt"	Aufgabe wird aus der Liste entfernt.	Aufgabe ist nicht mehr in der Liste.
Sortieren nach Datum	Testet, ob Aufgaben korrekt nach Fälligkeitsdatum sortiert werden.	Aufgaben mit verschiedenen Fälligkeitsdaten.	Klick auf "Sortieren nach Datum"	Aufgaben werden in aufsteigender oder absteigender Reihenfolge der Daten angezeigt.	Aufgabenliste nach Datum sortiert.



Testdokumentation für die <code>Util</code> -Klasse					
Testfall-ID	Methode	Beschreibung	Testschritte	Erwartetes Ergebnis	Hinweise
TC001	<code>sortByString</code>	Überprüft, ob Strings alphabetisch sortiert werden.	1. Erstellen eines Arrays <code>['banana', 'apple', 'cherry']</code> .	Das sortierte Array ist <code>['apple', 'banana', 'cherry']</code> .	
			2. Aufrufen von <code>sort(Util.sortByString)</code> auf dem Array.		
TC002	<code>sortByNumber</code>	Überprüft, ob Zahlen in aufsteigender Reihenfolge sortiert werden.	1. Erstellen eines Arrays <code>[5, 2, 9, 1]</code> .	Das sortierte Array ist <code>[1, 2, 5, 9]</code> .	
			2. Aufrufen von <code>sort(Util.sortByNumber)</code> auf dem Array.		
TC003	<code>sortByNumber</code>	Überprüft die Sortierung mit negativen Zahlen.	1. Erstellen eines Arrays <code>[-10, 0, 5, -2]</code> .	Das sortierte Array ist <code>[-10, -2, 0, 5]</code> .	
			2. Aufrufen von <code>sort(Util.sortByNumber)</code> auf dem Array.		
TC004	<code>sortByDate</code>	Überprüft, ob Daten in aufsteigender Reihenfolge sortiert werden.	1. Erstellen eines Arrays mit Daten <code>[2023-01-03, 2023-01-01, 2023-01-02]</code> .	Das sortierte Array ist <code>[2023-01-01, 2023-01-02, 2023-01-03]</code> .	
			2. Aufrufen von <code>sort(Util.sortByDate)</code> auf dem Array.		
TC005	<code>sortByDate</code>	Überprüft die Behandlung identischer Daten.	1. Erstellen eines Arrays mit identischen Daten <code>[2023-01-01, 2023-01-01]</code> .	Das sortierte Array bleibt <code>[2023-01-01, 2023-01-01]</code> .	

### 3. Haben wir

```
Running: todo.cy.ts (1 of 1)

TODO App Tests
  ✓ should render the TODO-App header (3428ms)
  ✓ add a new TODO item (2356ms)
  ✓ mark a TODO item as completed (555ms)
  ✓ sort TODOs by priorities (369ms)
  ✓ sort TODOs by Category and edit to get custom Category (1987ms)
  ✓ sort TODOs by Date (1034ms)

6 passing (10s)

(Results)

| Tests:      6
| Passing:    6
| Failing:    0
| Pending:    0
| Skipped:    0
| Screenshots: 0
| Video:      false
| Duration:   9 seconds
| Spec Ran:   todo.cy.ts

tput: No value for $TERM and no -T specified
=====

(Run Finished)
```

### 4.

```

it("sort TODOs by priorities", () => {
  cy.get(':nth-child(3) > .ant-segmented-group > :nth-child(2) > .ant-segmented-item-label')
  cy.get('table').should('contain.text', 'Aufräumen');
  cy.get(':nth-child(3) > .ant-segmented-group > :nth-child(3) > .ant-segmented-item-label')
  cy.get('table').should('contain.text', 'Einkaufen');
  cy.get(':nth-child(3) > .ant-segmented-group > :nth-child(4) > .ant-segmented-item-label')
  cy.get('table').should('contain.text', 'Znacht kochen');
  cy.get(':nth-child(3) > .ant-segmented-group > .ant-segmented-item-selected > .ant-segmented-it
  cy.get('table').should('contain.text', 'Aufräumen');
  cy.get('table').should('contain.text', 'Einkaufen');
  cy.get('table').should('contain.text', 'Znacht kochen');
});

```

5.

```

describe("TODO App Tests", () : void => {  ⚡ severin-fux
  beforeEach(() : void => {
    cy.visit("/");
  });

  it("should render the TODO-App header", () : void => {
    cy.get('⚡ .App-header').should('contain.text', 'TODO-App');
  });

  it("add a new TODO item", () : void => {
    cy.get('⚡ button').contains('+').click();
    cy.get('⚡ input[placeholder="Task Title"]').type('New Task');
    cy.get('⚡ #priority > .ant-segmented-group > :nth-child(3) > .ant-segmented-item-label').click(
    cy.get('⚡ input[id="categoryInput"]').type('{selectall}{del}Per{downarrow}{enter}') // use auto
    cy.get('⚡ input[placeholder="Select date"]').type('31-12-2025');

    cy.get('⚡ button').contains('Add Task').click(); //click out of Date selector
    cy.get('⚡ button').contains('Add Task').click();

    //check the 5th row
    cy.get('⚡ :nth-child(4) > [title="New Task"]').should('be.visible');
    cy.get('⚡ :nth-child(4) > [title="Personal"]').should('be.visible');
    cy.get('⚡ :nth-child(4) > [title="low"]').should('be.visible');
  });
}

```

```

3 describe('Util class', () :void => {  ± severin-fux +1
4   describe('sortByString', () :void => {
5     test('sorts strings alphabetically', () :void => {
6       const result :string[] = ['banana', 'apple', 'cherry'].sort(Util.sortByString);
7       expect(result).to.equal(['apple', 'banana', 'cherry']);
8     });
9   });
10
11   describe('sortByNumber', () :void => {
12     test('sorts numbers in ascending order', () :void => {
13       const result :number[] = [5, 2, 9, 1].sort(Util.sortByNumber);
14       expect(result).to.equal([1, 2, 5, 9]);
15     });
16
17     test('works with negative numbers', () :void => {
18       const result :number[] = [-10, 0, 5, -2].sort(Util.sortByNumber);
19       expect(result).to.equal([-10, -2, 0, 5]);
20     });
21   });
22
23   describe('sortByDate', () :void => {
24     test('sorts dates in ascending order', () :void => {
25       const date1 = new Date('2023-01-01');
26       const date2 = new Date('2023-01-02');
27       const date3 = new Date('2023-01-03');
28       const result :Date[] = [date3, date1, date2].sort(Util.sortByDate);
29       expect(result).to.equal([date1, date2, date3]);
30     });
31
32     test('handles identical dates', () :void => {
33       const date1 = new Date('2023-01-01');
34       const date2 = new Date('2023-01-01');
35       const result :Date[] = [date2, date1].sort(Util.sortByDate);
36       expect(result).to.equal([date1, date2]);
37     });
38   });
39 });

```

```

test('handles identical dates', () :void => {
  const date1 = new Date('2023-01-01');
  const date2 = new Date('2023-01-01');
  const result :Date[] = [date2, date1].sort(Util.sortByDate);
  expect(result).to.equal([date1, date2]);
});

```

- 6.
7. 9. (ist das selbe mit den aus 2.)

## Manuelle Integrationstests Übersicht

Test-ID	User Story	Testfall	Eingabe	Erwartetes Ergebnis	status
IT-01	Aufgaben priorisieren	Änderung der Priorität einer Aufgabe	Aufgabe "Einkaufen", neue Priorität "mittel"	Die Priorität wird erfolgreich auf "mittel" geändert.	pass
IT-02	Aufgaben in Kategorien einteilen	Aufgabe in eine andere Kategorie verschieben	Aufgabe "Joggen", neue Kategorie "Arbeit"	Die Aufgabe wird korrekt in die Kategorie "Arbeit" verschoben.	pass
IT-03	Aufgaben in Kategorien einteilen	Filtern nach einer Kategorie	Filter "Privat"	Nur Aufgaben der Kategorie "Privat" werden angezeigt.	pass
IT-04	Fälligkeitsdatum setzen	Farbige Markierung bei naher Deadline	Fälligkeitsdatum ist morgen	Die Aufgabe wird visuell hervorgehoben (z. B. rot markiert).	pass
IT-05	Fälligkeitsdatum setzen	Änderung des Fälligkeitsdatums	Aufgabe "Projekt", neues Datum 05.12.	Das Fälligkeitsdatum wird erfolgreich auf den 05.12. geändert.	pass
IT-06	Aufgaben priorisieren	Priorisierte Aufgaben korrekt sortieren	Aufgaben mit Priorität "hoch", "mittel", "niedrig"	Die Aufgaben werden nach Priorität sortiert angezeigt.	pass
				Nur Aufgaben der Kategorie	