1. Pendahuluan

1.1. Latar Belakang

Di era modern saat ini, kebutuhan akan manajemen tugas harian di lingkungan perkotaan semakin meningkat. Aktivitas yang padat dan kompleks menuntut individu maupun tim untuk dapat mengelola waktu dan prioritas secara efektif. Penggunaan alat bantu digital untuk manajemen tugas menjadi salah satu solusi yang efektif untuk meningkatkan produktivitas, mengurangi risiko lupa, dan memastikan pekerjaan terselesaikan tepat waktu.

Seiring berkembangnya teknologi web, aplikasi berbasis browser dengan kemampuan interaktif menjadi pilihan populer karena dapat diakses di berbagai perangkat tanpa perlu instalasi yang kompleks. Salah satu teknologi yang banyak digunakan adalah **React**, sebuah library JavaScript yang memungkinkan pengembangan antarmuka pengguna yang responsif dan dinamis.

1.2. Studi Kasus

Studi kasus yang dipilih dalam penelitian ini adalah **Urban Task Manager**, sebuah aplikasi pengelolaan tugas harian berbasis React. Aplikasi ini dirancang dengan berbagai fitur yang mendukung manajemen tugas secara efisien, antara lain:

- **Menambah tugas baru** dengan judul dan kategori tertentu.
- **Mengedit tugas** untuk memperbarui informasi yang sudah dibuat.
- **Menghapus tugas** yang sudah selesai atau tidak relevan.
- **Mengubah status tugas** (selesai/belum selesai) untuk memantau progres.
- **Filter kategori dan pencarian** untuk mempermudah pengguna menemukan tugas tertentu.
- **Penyimpanan state global** menggunakan **Redux Toolkit**, sehingga data dapat dikelola secara konsisten di seluruh komponen aplikasi.

Pengembangan aplikasi ini bertujuan untuk memberikan solusi praktis dalam pengelolaan tugas harian, sekaligus menjadi studi kasus yang dapat digunakan untuk mengevaluasi penerapan teknologi React dan Redux dalam konteks manajemen tugas perkotaan.

2. Analisis Arsitektur

2.1. Pendekatan Arsitektur

Aplikasi **Urban Task Manager** dikembangkan menggunakan **Component-Based Architecture** yang ditawarkan oleh React. Dalam pendekatan ini, antarmuka pengguna dibagi menjadi komponen-komponen kecil dan independen yang masing-masing memiliki tanggung jawab spesifik. Hal ini sejalan dengan prinsip **MVVM-like separation of concern**, di mana logika bisnis, state, dan tampilan dipisahkan untuk meningkatkan keterbacaan dan maintainability kode.

- **Component-Based Architecture** memungkinkan setiap bagian UI (misalnya tombol tambah tugas, daftar tugas, filter kategori) dibuat sebagai komponen terpisah yang dapat digunakan kembali di berbagai bagian aplikasi.
- **MVVM-like separation of concern** memisahkan state (model) dari tampilan (view) dan interaksi pengguna (view model), sehingga mempermudah pengelolaan dan pengujian tiap komponen.

2.2. Alasan Pemilihan Arsitektur

Beberapa alasan pemilihan arsitektur ini antara lain:

1. **Reuse (Penggunaan Ulang Komponen):** Komponen yang modular dapat digunakan kembali di berbagai halaman atau fitur, mengurangi duplikasi kode dan mempermudah pengembangan fitur baru.
2. **Testability (Kemudahan Pengujian):** Dengan komponen yang terisolasi, setiap unit dapat diuji secara terpisah menggunakan unit test atau integration test.
3. **Manajemen State yang Efisien:** Penggunaan **Redux Toolkit** memungkinkan pengelolaan state global secara terstruktur dan konsisten, sehingga setiap perubahan state dapat diakses oleh komponen yang membutuhkannya tanpa konflik.

Dengan arsitektur ini, aplikasi tidak hanya menjadi lebih mudah dikembangkan dan dipelihara, tetapi juga lebih skalabel untuk penambahan fitur di masa depan, sesuai dengan kebutuhan manajemen tugas harian yang dinamis di lingkungan perkotaan.

3. Struktur Folder (Usulan)

Untuk mempermudah pengembangan, pemeliharaan, dan skalabilitas aplikasi **Urban Task Manager**, struktur folder yang disarankan adalah sebagai berikut:

urban-task-manager/

```
├── .vscode/                # Konfigurasi VSCode, debugger, extensions
├── node_modules/           # Semua package npm/yarn yang terinstal
├── public/                 # File statis seperti index.html, favicon, gambar
│
├── src/
│   ├── app/
│   │   └── store.ts        # Konfigurasi Redux store global
│   │
│   ├── components/         # Komponen UI reusable
│   │   └── FilterBar.tsx
│   │
│   ├── features/           # Slice atau fitur aplikasi
│   │   ├── kategories/     # Slice dan logic untuk kategori
│   │   └── tasks/          # Slice dan komponen terkait task
│   │       ├── taskSlice.ts
│   │       └── TaskList.tsx
│   │
│   ├── pages/              # Halaman aplikasi (menggabungkan beberapa komponen)
│   │   └── HomePage.tsx
│   │
│   ├── routes/             # Routing aplikasi (React Router)
│   │   └── AppRoutes.tsx
│   │
│   ├── hooks/              # Custom hooks reusable
│   │   └── useFetchTasks.ts
│   │
│   ├── utils/              # Fungsi utilitas (pure functions)
│   │   └── dateFormatter.ts
│   │
│   ├── types/              # TypeScript types / interfaces
│   │   └── task.ts
│   │
│   ├── App.tsx             # Root component, layout dan routing
│   ├── App.css             # Styling global App
│   ├── main.tsx            # Entry point aplikasi (ReactDOM.render)
│   └── index.css           # Styling global tambahan
```

3.1. Penjelasan Struktur

1. **.vscode/** → Konfigurasi editor, debugging, settings workspace.
2. **node_modules/** → Semua package npm/yarn yang terinstal.
3. **public/** → File statis, seperti `index.html`, favicon, gambar, dll.
4. **src/app/store.ts** → Redux store global. Menggabungkan semua slice dari `features/`.
5. **src/components/** → Komponen UI reusable.
   Contoh: `FilterBar.tsx` bisa digunakan di banyak halaman.
6. **src/features/** → Slice / fitur spesifik aplikasi.
   a. `tasks/` → Semua logic dan UI terkait task, termasuk `taskSlice.ts` dan `TaskList.tsx`.
   b. `kategories/` → Slice untuk kategori, bisa menampung logic dan komponen terkait kategori.

7. **src/pages/** → Halaman lengkap, biasanya menggabungkan beberapa komponen.
   Contoh: `HomePage.tsx` menampilkan daftar task + FilterBar.
8. **src/routes/** → Menangani routing (React Router).
   Contoh: `AppRoutes.tsx` berisi `<Route>` untuk setiap halaman.
9. **src/hooks/** → Custom hooks untuk logika reusable.
   Contoh: `useFetchTasks.ts` untuk mengambil data task dari API atau state.
10. **src/utils/** → Fungsi utilitas, pure functions.
    Contoh: `dateFormatter.ts` untuk format tanggal di seluruh aplikasi.
11. **src/types/** → TypeScript interface / type definitions.
    Contoh: `task.ts` berisi type `Task` agar type-safe di seluruh project.
12. **App.tsx** → Root component. Biasanya memuat layout global dan routing.
13. **main.tsx** → Entry point aplikasi, melakukan render `<App />` ke DOM.
14. **index.css / App.css** → Styling global.

Struktur ini memudahkan pengembangan tim, mempercepat debugging, dan menjaga kode tetap rapi dan terorganisir seiring bertambahnya fitur.

4. Diagram Alur Navigasi dan State Flow

4.1. Rute Utama (Navigation Flow)

Aplikasi **Urban Task Manager** memiliki alur navigasi sederhana yang memudahkan pengguna untuk mengelola tugas harian:

Rute	Deskripsi
`/`	Halaman **Home** menampilkan daftar tugas dengan opsi filter, pencarian, dan status tugas.
`/create`	Halaman **Tambah Tugas**, memungkinkan pengguna menambahkan tugas baru dengan kategori dan detail tertentu.
`/tasks/:id`	Halaman **Detail Tugas**, menampilkan informasi lengkap tugas tertentu, termasuk opsi edit, hapus, atau ubah status.

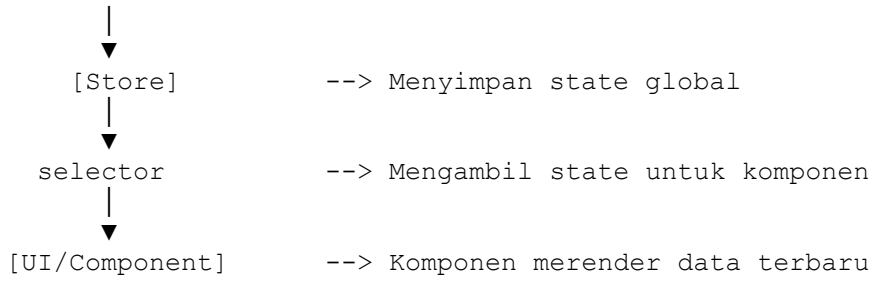
Alur Navigasi:

```
/ -> Home -> klik tombol "Tambah" -> /create -> submit -> kembali ke /
(Home)
/ -> Home -> klik salah satu tugas -> /tasks/:id -> edit/hapus/ubah status
-> kembali ke / (Home)
```

4.2. State Flow (Pengelolaan Data dengan Redux Toolkit)

Pengelolaan state mengikuti alur berikut:

```
[Component/UI]
      |
      ▼
 dispatch(action)  --> Mengirim aksi untuk mengubah state
      |
      ▼
   [Reducer]       --> Memproses aksi dan memperbarui state
```



Penjelasan alur:

1. Komponen UI memicu **action** melalui `dispatch` (misal: tambah tugas, hapus tugas, ubah status).
2. **Reducer** menerima action dan memperbarui **state** sesuai logika bisnis.
3. **Store** menyimpan state global, memastikan data konsisten di seluruh aplikasi.
4. Komponen menggunakan **selector** untuk membaca state dari store dan menampilkan data terbaru ke pengguna.

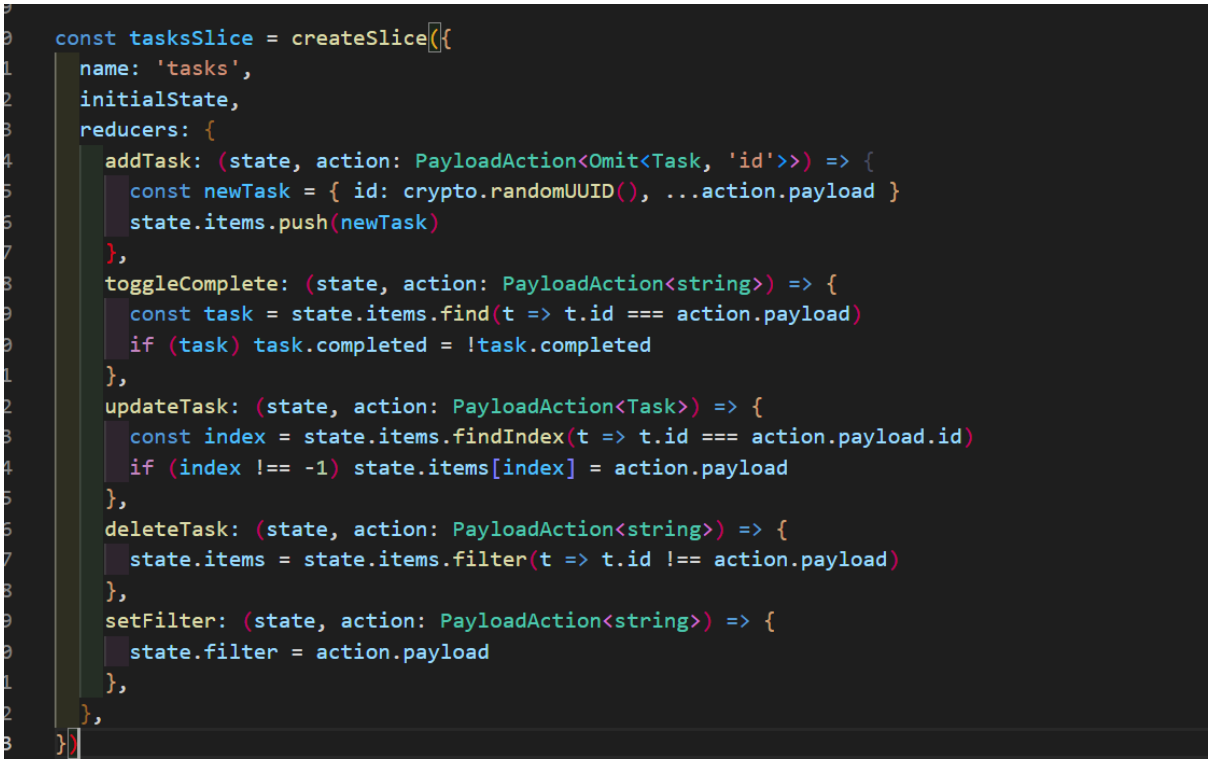
5. Implementasi Teknis (Ringkasan)

5.1. Teknologi yang Digunakan

Aplikasi **Urban Task Manager** dibangun menggunakan teknologi modern berbasis JavaScript, antara lain:

1. **React**
   Digunakan sebagai library utama untuk membangun antarmuka pengguna yang interaktif dan responsif dengan pendekatan **component-based architecture**.
2. **Redux Toolkit**
   Digunakan untuk pengelolaan **state global** aplikasi. Redux Toolkit mempermudah pembuatan reducer, action, dan store secara lebih ringkas dan terstruktur.
3. **React Router DOM**
   Digunakan untuk menangani **routing** aplikasi, memungkinkan navigasi antar halaman tanpa reload penuh.
4. **redux-persist (opsional)**
   Digunakan untuk menyimpan state Redux di **localStorage**, sehingga data tugas tetap tersimpan meskipun aplikasi direfresh.
5. **React Hook Form**
   Digunakan untuk menangani form input, validasi, dan submission secara efisien dan mudah diintegrasikan dengan state aplikasi.

Contoh Implementasi taskSlice:



```
const tasksSlice = createSlice({
  name: 'tasks',
  initialState,
  reducers: {
    addTask: (state, action: PayloadAction<Omit<Task, 'id'>>) => {
      const newTask = { id: crypto.randomUUID(), ...action.payload }
      state.items.push(newTask)
    },
    toggleComplete: (state, action: PayloadAction<string>) => {
      const task = state.items.find(t => t.id === action.payload)
      if (task) task.completed = !task.completed
    },
    updateTask: (state, action: PayloadAction<Task>) => {
      const index = state.items.findIndex(t => t.id === action.payload.id)
      if (index !== -1) state.items[index] = action.payload
    },
    deleteTask: (state, action: PayloadAction<string>) => {
      state.items = state.items.filter(t => t.id !== action.payload)
    },
    setFilter: (state, action: PayloadAction<string>) => {
      state.filter = action.payload
    },
  },
})
```

1. Halaman Utama: Daftar Tugas Harian

- **Tampilan**: Pengguna melihat tampilan daftar tugas harian dengan kategori tab (Semua, Pekerjaan, Pribadi, Belajar, Lainnya).
- **Input Tugas Baru**:
    - Ada kolom untuk menambahkan tugas dengan placeholder "Tambahkan tugas..." dan dropdown untuk memilih kategori (default: Pekerjaan).
    - Terdapat tombol “Tambah” untuk menambahkan tugas ke dalam daftar.
- **Tugas yang Terdaftar**: Pengguna dapat melihat tugas yang sudah terdaftar. Setiap tugas memiliki:
    - Nama tugas (misalnya "hallo (Pekerjaan)").
    - Tombol Edit untuk mengubah nama atau kategori tugas.
    - Tombol Hapus untuk menghapus tugas dari daftar.

2. Menambahkan Tugas Baru

- **Langkah 1**: Pengguna mengklik kolom input "Tambahkan tugas..." untuk mengetikkan nama tugas.
- **Langkah 2**: Pengguna memilih kategori tugas dari dropdown (misalnya, memilih “Pekerjaan”).
- **Langkah 3**: Pengguna menekan tombol “Tambah” untuk menambahkan tugas tersebut ke dalam daftar.

3. Melihat Daftar Tugas

- Pengguna dapat melihat tugas-tugas yang telah ditambahkan sebelumnya. Setiap tugas ditampilkan dengan nama dan kategori.
- **Tab Kategori**:
    - Pengguna dapat memfilter tugas berdasarkan kategori dengan mengklik salah satu tab: “Semua”, “Pekerjaan”, “Pribadi”, “Belajar”, atau “Lainnya”.
    - Jika tab “Pekerjaan” dipilih, hanya tugas-tugas yang masuk dalam kategori pekerjaan yang akan ditampilkan.

4. Mengedit Tugas

- Pengguna dapat mengklik tombol “Edit” di samping tugas yang terdaftar untuk mengubah nama tugas atau kategori.
- Setelah diedit, perubahan akan diperbarui dalam daftar.

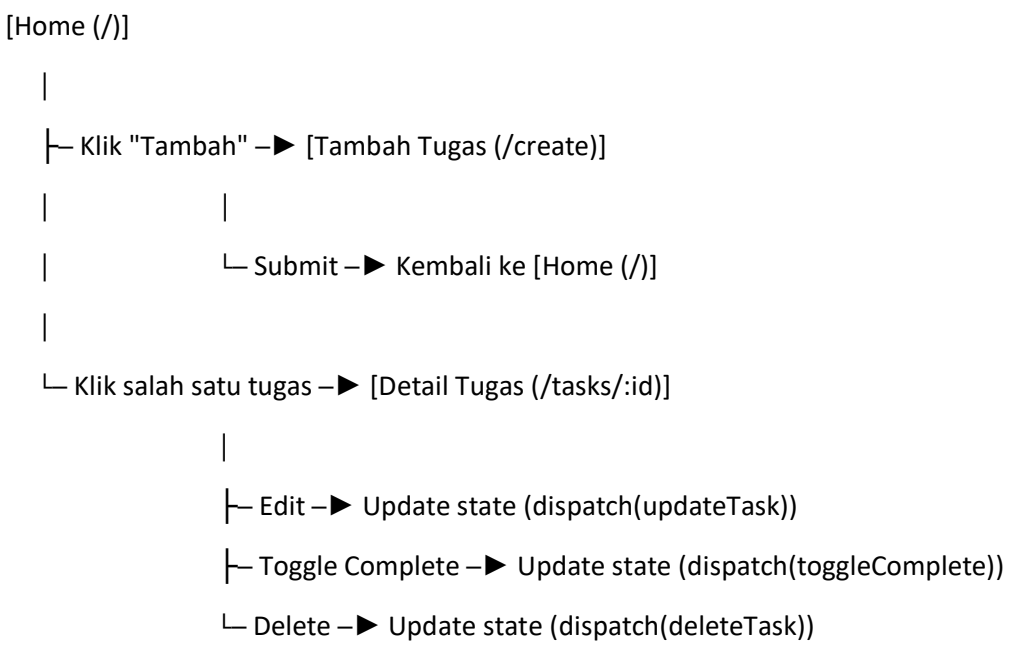
5. Menghapus Tugas

- Pengguna dapat mengklik tombol “Hapus” untuk menghapus tugas yang tidak diperlukan lagi dari daftar tugas harian.

6. Navigasi dan Filter

- Pengguna dapat berpindah antar tab kategori untuk memfilter daftar tugas sesuai dengan kategori yang diinginkan.

- Daftar tugas dapat terus diperbarui baik dengan menambah, mengedit, atau menghapus tugas.



6. Refleksi & Tantangan

Tantangan:

1. **Persist State**:
   Menjaga konsistensi state aplikasi agar tetap terjaga setelah reload atau navigasi halaman adalah tantangan utama. Dengan menggunakan state management seperti Redux Toolkit (RTK), state dapat dengan mudah diatur di seluruh aplikasi, namun jika tidak ada backend, kita perlu mempertimbangkan solusi seperti localStorage atau sessionStorage untuk menyimpan state aplikasi secara persisten di sisi klien. Tanpa ini, aplikasi akan kehilangan data setiap kali halaman dimuat ulang.
   - **Solusi**:
     Menggunakan middleware seperti `redux-persist` untuk menyimpan state tertentu ke dalam localStorage atau sessionStorage. Ini memungkinkan state untuk tetap ada meski aplikasi di-reload.
2. **Konsistensi Filter/Search**:
   Menyaring atau mencari tugas berdasarkan kategori atau status membutuhkan konsistensi dalam pengelolaan data. Tanpa backend yang menyediakan query atau pencarian yang lebih canggih, kita bergantung pada frontend untuk mengelola dan menyaring data sesuai input pengguna. Hal ini bisa menjadi menantang jika volume data yang harus dikelola besar.
   - **Solusi**:
     Menggunakan selector di Redux untuk memfilter data berdasarkan kondisi yang diterapkan pengguna. Selain itu, bisa menggunakan debouncing untuk pencarian agar tidak terjadi pengolahan data secara berlebihan saat pengguna mengetikkan query pencarian.

3. **Protected Routes Tanpa Backend**:
   Untuk aplikasi yang membutuhkan autentikasi dan otorisasi, biasanya kita melibatkan backend untuk mengelola sesi dan token. Tanpa backend, kita harus menciptakan mekanisme untuk memvalidasi apakah pengguna dapat mengakses route tertentu, misalnya dengan menyimpan status login di `localStorage` atau `sessionStorage`.
   - **Solusi**:
     Implementasi otentikasi di sisi klien (misalnya menggunakan token JWT yang disimpan di `localStorage`) dapat memberikan kontrol akses untuk protected routes. Meskipun bukan solusi yang sepenuhnya aman (karena bergantung pada sisi klien), ini bisa berfungsi untuk aplikasi dengan kebutuhan proteksi dasar.

Pembelajaran:

1. **RTK Mempercepat Pengembangan**:
   Redux Toolkit (RTK) mempermudah pengelolaan state global dengan menyediakan metode dan hooks yang intuitif, seperti `createSlice`, `createAsyncThunk`, dan `useSelector` serta `useDispatch`. RTK mengurangi boilerplate code yang diperlukan dalam pengelolaan state dan mempermudah pengembangan fitur aplikasi. Ini mempercepat pengembangan dengan menyediakan API yang lebih bersih dan efisien.
   - **Keuntungan**:
     RTK sangat mengurangi kerumitan dalam menulis reducer dan actions secara manual. Dengan pengelolaan state yang lebih terstruktur, pengembang dapat lebih fokus pada logika aplikasi daripada menghabiskan waktu dengan masalah terkait state management.
2. **Pemisahan Komponen Memudahkan Testing**:
   Memisahkan aplikasi menjadi komponen-komponen kecil dan modular mempermudah dalam hal testing. Komponen yang lebih kecil memiliki tanggung jawab yang jelas, sehingga lebih mudah diuji unit test-nya. Hal ini juga mempermudah dalam hal pemeliharaan dan refactoring kode karena pengubahan satu bagian aplikasi tidak terlalu mempengaruhi bagian lain yang tidak terkait.
   - **Keuntungan**:
     Dengan komponen yang modular, kita dapat menulis tes unit untuk setiap bagian aplikasi, memastikan bahwa fungsi dan komponen bekerja seperti yang diinginkan. Testing menjadi lebih mudah, dan kualitas kode meningkat.

Kesimpulan:

Meskipun tanpa backend, tantangan seperti persist state, konsistensi filter/search, dan pengelolaan protected routes dapat diatasi dengan solusi berbasis klien (seperti penggunaan `redux-persist`, localStorage, dan sessionStorage). Di sisi lain, penggunaan **Redux Toolkit (RTK)** dan **pemisahan komponen** memberikan pengalaman pengembangan yang lebih efisien dan memungkinkan pengujian yang lebih baik.

Secara keseluruhan, implementasi RTK sangat bermanfaat dalam mempercepat pengembangan, sementara desain aplikasi yang modular meningkatkan kemampuan untuk mengelola dan menguji aplikasi dengan lebih baik.

7. DAFTAR LAMPIRAN

- Dokumentasi React
- Dokumentasi Redux Toolkit
- Dokumentasi React Router DOM

LINK REPOSTORY

https://github.com/aliennn04/uts_react.git

LINK DEPLOY

uts-react-yven.vercel.app