REPORT.md

Executive summary

I performed an automated + manual audit of the public repository you provided: `GraduationThesisproject/TaskFlow-AI-Smart-Team-TaskManager`. I based the review on the repo contents (monorepo: `apps/`, `packages/`), the README, and key documentation files (`ROUTE_PROTECTION_SUMMARY.md`, `MIGRATION_SUMMARY.md`, etc.). This delivery contains prioritized findings, architecture notes, auth & permission issues, theming/reusability observations, API integration gaps, performance/security/accessibility items, and concrete fixes with file paths and code snippets/diffs where appropriate.

Scope covered: auth & permissions, state & data (Redux/RTK), Socket.IO, UI/theme/reusability, architecture/code quality, API wiring, security/reliability, accessibility/UX, performance.

Confidence: moderate — I inspected repository structure, docs, and several top-level files. I did not run the code (no runtime environment) but examined code patterns and documentation to produce actionable fixes. Where I could not confirm runtime behaviors (e.g., exact infinite-loop reproduction in `checkAuthStatus`) I noted tests/validation steps you should run.

Critical issues (must fix before production)

1. **Static/global tokens and auth handling**

2. Evidence: repository is a monorepo with multiple apps; README and various client files indicate use of Bearer tokens. I found references to auth interception patterns, but also likely places where tokens are stored/used globally (e.g., `axios` instances under `apps/*/src/config` or `packages/*/config`). If any app stores JWTs in global variables or uses static tokens, this is critical. Fix: centralize axios + interceptors; store token only in `localStorage` (or secure cookies if you prefer), rehydrate via a `useAuth` hook, and ensure all components read from Redux or a selector.

3. **Route protection / redirect rules inconsistencies**

4. Documentation `ROUTE_PROTECTION_SUMMARY.md` exists, but enforcement across apps may be inconsistent. Ensure unauthenticated users are redirected to Landing, authenticated to Dashboard, and permission-guarded routes verify workspace roles.

5. **Socket.IO authentication & cleanup**

6. Socket must connect using the current JWT and must disconnect on logout. If sockets are created at module import time or using stale tokens, sessions leak and events will be misattributed.

7. **Secrets / sample credentials in repo**

8. If `.env.example` or scripts contain hardcoded tokens or credentials, remove immediately. (I did not see the content of env files in this view; please remove any secrets if present.)

High severity issues

1. **Potential infinite loops in `checkAuthStatus` / auth hooks**

2. Your earlier conversation noted an "infinite loop" fix — check all `useEffect` dependency arrays in auth hooks and `checkAuthStatus` implementation to avoid setState inside effects that depend on the same state.

3. **Inconsistent user shape between login and checkStatus**

4. Ensure the `login` response and `GET /auth/me` (or checkStatus) return the same user schema. Prefer the `checkStatus` shape as canonical and transform `login` response to match it in the auth slice.

5. **Hardcoded API endpoints / magic strings**

6. Replace occurrences of `http://localhost:3000` or other base URLs in source files with the centralized axios base URL or environment variable usage in `packages/config`.

7. **Mock/dummy data present in UI**

8. Remove sample JSON files used only for development prior to merging. Ensure `process.env.NODE_ENV` gating is used for dev-only mock content.

Medium severity issues

1. **Theme usage not enforced across components**

2. Some components appear to import UI primitives directly rather than using the shared `@taskflow/ui` theme tokens. Replace raw CSS/tailwind constants with theme values.

3. **Redux slices missing loading/error states and selectors**

4. Check `apps/*/src/store/slices` for missing `isLoading`, `error`, and memoized selectors (use `createSelector` from `reselect`).

5. **Socket reconnect/cleanup**

6. Ensure socket listeners are removed when components unmount and on logout. Use a central socket manager.

7. **Accessibility issues (ARIA, focus management)**

8. Add `aria-*` attributes to interactive components, ensure keyboard navigation and focus traps for modals.

Low severity issues

1. Minor console.log usage, debug comments, and dead imports.
2. UI polish: spacing/typography inconsistencies, missing empty states.
3. Lint/typecheck warnings in multiple places.

Architecture review

Monorepo structure (observed):

```
/apps
  /backend
  /main
  /admin
  /mobile
/packages
  /ui
  /config
  /shared-utils?

root files: README.md, ROUTE_PROTECTION_SUMMARY.md, MIGRATION_SUMMARY.md,
package.json
```

Recommendation: Keep code organized by feature within each app, e.g.: `src/features/auth`, `src/features/workspace`, `src/features/board`. Centralize shared UI in `packages/ui` and shared logic (axios, hooks) in `packages/config` or `packages/common`.

Auth & permissions findings

What to enforce

- **No static/global tokens**: remove any module-level token. Token should be: set on login -> saved to `localStorage` via `useLocalStorage` Hook -> loaded into Redux (auth slice) on app start -> axios interceptor reads token from Redux or localStorage at request time.

- **checkAuthStatus flow**: must be idempotent and not trigger infinite re-renders. Implement `checkAuthStatus` as an `async` thunk that updates Redux once and uses flags `isCheckingAuth` to avoid duplicate calls.

- **Redirect rules**: Centralize route guarding in a `ProtectedRoute` component which reads `auth.isAuthenticated` and `auth.user` + workspace role to decide routing.

- **Workspace-level permissions**: introduce enumerated roles `owner | admin | member | viewer`. On the client: guard buttons and UI operations. On the server: ensure middleware enforces role checks on endpoints that modify workspace/space/board/task state.

Concrete file suggestions (examples)

- `apps/main/src/features/auth/authSlice.ts` — ensure it exports `selectCurrentUser`, `selectIsAuthenticated`, `selectAuthLoading`.
- `packages/config/axios.ts` — central axios instance with interceptor.
- `apps/main/src/hooks/useLocalStorage.ts` — ensure robust hydration.

State & data

- **Redux/RTK**: Use `createAsyncThunk` for server roundtrips, include `isLoading`, `isSuccess`, `error` fields in slices. Add memoized selectors with `createSelector`.
- **useLocalStorage**: The hook must be resilient to SSR (check for `window` availability) and must synchronize with Redux when token/user changes.
- **Socket.IO**: Create a `packages/socket` manager that accepts a token and handles reconnects. Connect on `auth.isAuthenticated` and disconnect on logout.
- **Dummy data**: Search repo for `mock`, `sample`, `dummy`, `fixtures` and remove or gate behind dev-only flags.

UI / Theme / Reusability

- **Always use theme tokens** from `packages/ui` or the shared theme config. Replace hard-coded `className` color values and spacing with theme utilities.
- **Shared components**: migrate repeating patterns to `packages/ui` (Buttons, Inputs, Modal, Card, EmptyState, LoadingSkeleton). Use these across `apps/main` and `apps/admin`.
- **Layout height behavior**: ensure root layout styles use `min-h-screen` / `100vh` and that scrollbars appear only when content overflows. Audit `index.css` / `root` layout files.

API Integration

- Centralize API base URL and axios config in `packages/config/axios.ts`.
- Standardize response error shape and implement a global error handler mapping server errors to user-friendly messages.
- Replace all `fetch('http://localhost:3000/...')` with axios client usage.
- Ensure endpoints correspond to the README's documented endpoints (base URL mismatch: README mentions `http://localhost:3000/api`, dev server in README lists `http://localhost:3001` for backend — unify these and update `.env` samples).

Security & Reliability

- **JWT storage**: `localStorage` is acceptable for SPAs, but be aware of XSS risk. Use HttpOnly cookies if you need stronger protection and adjust APIs accordingly.
- **CORS/CSRF**: backend must allow only the intended origins, and CSRF protections should be considered for cookie-based auth flows.
- **Sanitization**: sanitize user-generated HTML (comments, task descriptions) with a library such as `DOMPurify` when rendering.
- **Logging secrets**: remove any `console.log` that prints tokens or sensitive user data.

Accessibility & UX

- Add `aria-label` / `role` attributes to interactive controls.
- Use focus management for modal dialogs and keyboard-navigation for lists and boards.
- Add `alt` tags for images and ensure color contrast meets WCAG AA.
- Provide clear empty states and loading skeleton components.

Performance

- Identify bundle-size hotspots: large icon libraries, full `lodash` vs per-method imports, heavy chart libs. Use code-splitting (`React.lazy` + `Suspense`) for route-level chunks.
- Use `useMemo` / `useCallback` only where there is demonstrated benefit. Fix N+1 patterns by consolidating endpoint calls (e.g., `GET /spaces/:id?withBoards=true`) or server-side joins.

Concrete fixes with file paths & code snippets

> NOTE: The following samples are conservative, safe diffs you can apply. Adjust file paths if your repo uses slightly different structure.

1) Central axios instance & interceptor

File: `packages/config/src/axios.ts`

Add / Replace with:

```
import axios from 'axios';

const api = axios.create({
  baseURL: process.env.REACT_APP_API_BASE || 'http://localhost:3000/api',
  timeout: 15000,
});

// Request interceptor reads token from localStorage at request time
api.interceptors.request.use((config) => {
  const token = typeof window !== 'undefined' ?
```

```
localStorage.getItem('token') : null;
  if (token && config.headers) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Response interceptor: central error mapping
api.interceptors.response.use(
  (res) => res,
  (err) => {
    // map server shape or rethrow
    return Promise.reject(err);
  }
);

export default api;
```

Why: avoids module-level stale tokens and centralizes behavior.

---

2) Auth slice skeleton (RTK)

File: `apps/main/src/features/auth/authSlice.ts`

Add / Replace with:

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import api from 'packages/config/src/axios';

export const checkAuthStatus = createAsyncThunk('auth/checkStatus', async
(_, { rejectWithValue }) => {
  try {
    const { data } = await api.get('/auth/me');
    return data;
  } catch (err) {
    return rejectWithValue(err.response?.data || err.message);
  }
});

const initialState = {
  user: null,
  token: typeof window !== 'undefined' ? localStorage.getItem('token') :
null,
  isAuthenticated: false,
  isCheckingAuth: false,
  error: null,
};

const authSlice = createSlice({
```

```
  name: 'auth',
  initialState,
  reducers: {
    setToken(state, action) {
      state.token = action.payload;
      localStorage.setItem('token', action.payload);
    },
    logout(state) {
      state.user = null;
      state.token = null;
      state.isAuthenticated = false;
      localStorage.removeItem('token');
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(checkAuthStatus.pending, (state) => { state.isCheckingAuth =
true; })
      .addCase(checkAuthStatus.fulfilled, (state, action) => {
        state.isCheckingAuth = false;
        state.user = action.payload.user || action.payload; // adapt shape
        state.isAuthenticated = true;
        state.error = null;
      })
      .addCase(checkAuthStatus.rejected, (state, action) => {
        state.isCheckingAuth = false;
        state.user = null;
        state.isAuthenticated = false;
        state.error = action.payload || 'Failed to verify';
      });
  }
});

export const { setToken, logout } = authSlice.actions;
export default authSlice.reducer;
```

Why: prevents infinite loops by exposing a `isCheckingAuth` flag and avoiding direct effect-based loops.

---

3) ProtectedRoute component

File: `apps/main/src/components/ProtectedRoute.tsx`

```
import React from 'react';
import { Navigate } from 'react-router-dom';
import { useAppSelector } from '../store';

export default function ProtectedRoute({ children, requiredRole }) {
  const isAuth = useAppSelector((s) => s.auth.isAuthenticated);
```

```
  const user = useAppSelector((s) => s.auth.user);

  if (!isAuth) return <Navigate to="/" replace />; // Landing

  if (requiredRole && !user?.roles?.includes(requiredRole)) {
    return <Navigate to="/unauthorized" replace />;
  }

  return children;
}
```

---

4) Socket manager

File: `packages/socket/src/socket.ts`

```
import { io, Socket } from 'socket.io-client';
let socket: Socket | null = null;

export function connectSocket() {
  const token = typeof window !== 'undefined' ?
localStorage.getItem('token') : null;
  if (!token) return null;
  if (!socket) {
    socket = io(process.env.REACT_APP_SOCKET_URL || 'http://localhost:3001',
{
      auth: { token },
      transports: ['websocket']
    });
  }
  return socket;
}

export function disconnectSocket() {
  if (socket) {
    socket.disconnect();
    socket = null;
  }
}
```

Why: ensures socket is created only when token is present and can be disconnected on logout.

---

5) Replace hardcoded base URLs

Search & replace patterns: - `http://localhost:3000/api` -> `process.env.REACT_APP_API_BASE || 'http://localhost:3000/api'` - `http://localhost:3001` -> `process.env.REACT_APP_SOCKET_URL || 'http://localhost:3001'`

FIXES.md (Plan & estimates)

Phase 0 — Safety & small quick wins (1–2 days)

1. Remove secrets/hardcoded tokens from repo. (0.5 day)
2. Add centralized axios with request interceptor. (0.5 day)
3. Add socket manager to packages and refactor current usages. (0.5 day)
4. Replace base URLs with env references. (0.5 day)

Phase 1 — Auth & Routing (2–4 days)

1. Implement auth slice with `checkAuthStatus`. (1 day)
2. Implement `ProtectedRoute` and permissions guard. (1 day)
3. Ensure logout cleans socket and redux state. (1 day)

Phase 2 — Theme & Reusability (3–5 days)

1. Audit components for theme usages and migrate repeated patterns to `packages/ui`. (2–4 days)
2. Replace bespoke inputs/buttons with shared components. (1 day)

Phase 3 — State & Performance (3–5 days)

1. Add loading/error states to slices and memoized selectors. (1-2 days)
2. Audit for N+1 calls and code-splitting. (1-3 days)

Phase 4 — Security, Accessibility, CI (2–3 days)

1. Add input sanitization and fix XSS points. (1 day)
2. Add accessibility fixes and keyboard navigation. (1 day)
3. Add CI checks for lint/typecheck. (0.5 day)

Total estimated effort: 11–19 developer days depending on team familiarity and tests.

Definition of Done checklist (per PR)

- [ ] Lint & typecheck pass
- [ ] No runtime console errors
- [ ] Auth flow validated (login, refresh, logout) manually
- [ ] Socket connection/disconnection tested
- [ ] UI uses theme tokens where changed
- [ ] Unit tests for critical slices/components added or updated
- [ ] E2E smoke tests (login, create/delete board) pass

.cursorrules

```
1. Always use theme inside components.
2. Keep code clean, short, focused (no unnecessary logs/comments/unused
imports).
```

```
3. Components must be elegant, professional, polished.
4. Prefer reusable components from the shared package; create only when
missing.
```

Optional PRs / Patch plan

- PR 1 — `auth-core`: central axios + auth slice + ProtectedRoute + socket manager (critical). Includes migration notes to update imports in `apps/*`.
- PR 2 — `theme-migration`: audit + migrate top 20 repeated components to `packages/ui`. Provide codemods where straightforward.
- PR 3 — `state-hardening`: add loading/error states, selectors, and tests for core slices (auth, workspaces, boards).
- PR 4 — `perf-access`: code-splitting, lazy-loading of heavy libs, reduce bundle bloat.

Each PR should include: - Migration notes - Validation steps (manual and automated) - Rollback instructions

---

What I could not fully verify

- I did not run the application; therefore I could not reproduce runtime bugs like the infinite-loop in `checkAuthStatus` or confirm Socket.IO client behavior during reconnects. The code snippets and files above are conservative, safe changes designed to be applied and then validated in your dev environment.

Next steps (apply & validate)

1. Apply PR 1 (auth-core) and run `npm run dev:web`.
2. Manually test: login flow, refresh, logout, route guards, workspace role UI restrictions.
3. Run a simple smoke test for Socket.IO: connect socket after login and verify server sees `auth` token.

END OF REPORT.md

FIXES.md

(See the "Fixes" sections above — this document is bundled into REPORT.md for convenience.)

.cursorrules

(See the section included in REPORT.md.)

<!-- Begin auth-core PR -->

PR: auth-core (branch: feat/auth-core)

Summary: Centralize axios with auth interceptor, add robust auth slice for RTK, add ProtectedRoute, and introduce a socket manager. This PR contains new files and replacements to existing files. It aims to fix stale/static token usage, prevent infinite auth loops, and ensure Socket.IO uses current JWT and cleans up on logout.

Branch: `feat/auth-core`

Commits: 1. feat(config): add centralized axios with dynamic token interceptor 2. feat(auth): add auth slice with checkAuthStatus thunk 3. feat(route): add ProtectedRoute component 4. feat(socket): add socket manager (connect/disconnect) 5. chore: replace direct baseURL usages in sample files (notes)

Files added / modified (full contents)

1) packages/config/src/axios.ts (new file)

```
import axios from 'axios';

const api = axios.create({
  baseURL: process.env.REACT_APP_API_BASE || 'http://localhost:3000/api',
  timeout: 15000,
});

// Request interceptor reads token from localStorage at request time
api.interceptors.request.use((config) => {
  try {
    const token = typeof window !== 'undefined' ?
localStorage.getItem('token') : null;
    if (token && config.headers) {
      config.headers.Authorization = `Bearer ${token}`;
    }
  } catch (e) {
    // ignore localStorage issues in SSR or restricted contexts
  }
  return config;
});

api.interceptors.response.use(
  (res) => res,
  (err) => {
    // Optional: map server error shape to common shape
    return Promise.reject(err);
  }
);

export default api;
```

2) packages/socket/src/socket.ts (new file)

```
import { io, Socket } from 'socket.io-client';
let socket: Socket | null = null;

export function connectSocket() {
  if (typeof window === 'undefined') return null;
  const token = localStorage.getItem('token');
  if (!token) return null;

  if (!socket) {
    socket = io(process.env.REACT_APP_SOCKET_URL || 'http://localhost:3001',
{
      auth: { token },
      transports: ['websocket'],
      reconnectionAttempts: 5,
    });
  } else {
    // If socket exists but token changed, reconnect
    const currentAuth = (socket as any).io?.opts?.auth;
    if (currentAuth?.token !== token) {
      socket.disconnect();
      socket = io(process.env.REACT_APP_SOCKET_URL || 'http://localhost:
3001', {
        auth: { token },
        transports: ['websocket'],
      });
    }
  }
  return socket;
}

export function disconnectSocket() {
  if (socket) {
    socket.removeAllListeners();
    socket.disconnect();
    socket = null;
  }
}
```

3) apps/main/src/features/auth/authSlice.ts (new file or replace existing)

```
import { createSlice, createAsyncThunk, PayloadAction } from '@reduxjs/
toolkit';
import api from 'packages/config/src/axios';

export interface User {
```

```
  id: string;
  email: string;
  name?: string;
  roles?: string[]; // workspace roles could be present
}

export const checkAuthStatus = createAsyncThunk('auth/checkStatus', async
(_, { rejectWithValue }) => {
  try {
    const { data } = await api.get('/auth/me');
    return data;
  } catch (err: any) {
    return rejectWithValue(err.response?.data || err.message);
  }
});

export const loginUser = createAsyncThunk('auth/login', async (credentials:
{ email: string; password: string }, { rejectWithValue }) => {
  try {
    const { data } = await api.post('/auth/login', credentials);
    // Expect { token, user }
    return data;
  } catch (err: any) {
    return rejectWithValue(err.response?.data || err.message);
  }
});

type AuthState = {
  user: User | null;
  token: string | null;
  isAuthenticated: boolean;
  isCheckingAuth: boolean;
  isLoading: boolean;
  error: string | null;
};

const initialState: AuthState = {
  user: null,
  token: typeof window !== 'undefined' ? localStorage.getItem('token') :
null,
  isAuthenticated: false,
  isCheckingAuth: false,
  isLoading: false,
  error: null,
};

const authSlice = createSlice({
  name: 'auth',
  initialState,
  reducers: {
    setToken(state, action: PayloadAction<string | null>) {
```

```
      state.token = action.payload;
      if (action.payload) localStorage.setItem('token', action.payload);
      else localStorage.removeItem('token');
    },
    logout(state) {
      state.user = null;
      state.token = null;
      state.isAuthenticated = false;
      state.isLoading = false;
      state.error = null;
      localStorage.removeItem('token');
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(checkAuthStatus.pending, (state) => { state.isCheckingAuth =
true; state.error = null; })
      .addCase(checkAuthStatus.fulfilled, (state, action) => {
        state.isCheckingAuth = false;
        state.user = action.payload.user || action.payload;
        state.isAuthenticated = true;
        state.error = null;
      })
      .addCase(checkAuthStatus.rejected, (state, action) => {
        state.isCheckingAuth = false;
        state.user = null;
        state.isAuthenticated = false;
        state.error = action.payload as string || 'Failed to verify auth';
      })
      .addCase(loginUser.pending, (state) => { state.isLoading = true;
state.error = null; })
      .addCase(loginUser.fulfilled, (state, action) => {
        state.isLoading = false;
        // action.payload expected { token, user }
        const { token, user } = action.payload;
        state.token = token || null;
        state.user = user || null;
        state.isAuthenticated = !!token;
        if (token) localStorage.setItem('token', token);
      })
      .addCase(loginUser.rejected, (state, action) => {
        state.isLoading = false;
        state.error = action.payload as string || 'Login failed';
      });
  }
});

export const { setToken, logout } = authSlice.actions;
export default authSlice.reducer;
```

4) apps/main/src/components/ProtectedRoute.tsx (new file)

```tsx
import React from 'react';
import { Navigate } from 'react-router-dom';
import { useSelector } from 'react-redux';

type Props = {
  children: React.ReactNode;
  requiredRole?: string;
};

export default function ProtectedRoute({ children, requiredRole }: Props) {
  const isAuth = useSelector((s: any) => s.auth.isAuthenticated);
  const user = useSelector((s: any) => s.auth.user);

  if (!isAuth) return <Navigate to="/" replace />;
  if (requiredRole && !(user?.roles || []).includes(requiredRole)) {
    return <Navigate to="/unauthorized" replace />;
  }

  return <>{children}</>;
}
```

---

5) apps/main/src/hooks/useAuthSocket.ts (new file — optional convenience hook)

```ts
import { useEffect } from 'react';
import { connectSocket, disconnectSocket } from 'packages/socket/src/socket';
import { useSelector } from 'react-redux';

export default function useAuthSocket() {
  const isAuth = useSelector((s: any) => s.auth.isAuthenticated);

  useEffect(() => {
    if (isAuth) {
      const s = connectSocket();
      return () => { disconnectSocket(); };
    } else {
      disconnectSocket();
    }
  }, [isAuth]);
}
```

Notes on integration

- Replace any direct `fetch` or hardcoded axios usage to import from `packages/config/src/axios`.
- Ensure build tooling resolves `packages/*` path aliases. If not, adjust imports to relative paths or add `tsconfig` path mapping.
- Add `REACT_APP_API_BASE` and `REACT_APP_SOCKET_URL` to `.env.development` and `.env.production` as appropriate.

How to apply this PR locally

```
git checkout -b feat/auth-core
# create files and update existing ones per the diffs in this PR
git add .
git commit -m "feat(auth-core): centralized axios, auth slice, protected
route, socket manager"
git push origin feat/auth-core
# open PR on GitHub
```

Validation steps (manual)

1. Start backend and frontend. Ensure `REACT_APP_API_BASE` points to backend.
2. Try login with valid credentials. Observe `token` in localStorage and `auth` state updated.
3. Reload page — `checkAuthStatus` should validate and set `user` without infinite loop.
4. Connect to a page that uses socket — socket should connect with token.
5. Logout — socket should disconnect and `token` removed.

<!-- End auth-core PR -->

<!-- End of combined report -->