Quality Assurance Report: National Fire Web App

Stack: Node.js, React, Tailwind CSS, PostgreSQL (Neon DB), Drizzle ORM

Traffic Expectation: \~100 visits/day\ **Goal:** Reliability and maintainability equivalent to a professional-grade application

---

1. Security & Authentication

1.1 Enforce HTTPS and Security Headers

- **File:** `server/index.ts`
- **Improvement:** Ensure app uses HTTPS and security headers in production.
- **How:**

```
if (process.env.NODE_ENV === 'production') {
  app.set('trust proxy', 1);
  app.enable('trust proxy');
}
```

- **Scope:** Minor config update

1.2 Secure Session Cookie

- **File:** `server/routes.ts`
- **Improvement:** Use custom cookie name and secure flags.
- **How:**

```
app.use(session({
  secret: sessionSecret!,
  name: 'sessionId',
  resave: false,
  saveUninitialized: false,
  store: sessionStore,
  cookie: {
    secure: process.env.NODE_ENV === 'production',
    httpOnly: true,
    sameSite: process.env.NODE_ENV === 'production' ? 'strict' : 'lax'
```

```
  }
}));
```

- **Scope:** Configuration change for cookie security

1.3 Rate Limiting for Login

- **File:** `server/routes.ts`
- **Improvement:** Apply rate limiter middleware to `/api/login`
- **How:**

```
app.post('/api/login', authLimiter, (req, res) => {
  // login logic
});
```

- **Scope:** Minor security enhancement

1.4 API 404 and Error Handling

- **File:** `server/index.ts`
- **Improvement:** Add 404 handler for APIs and global error handler
- **How:**

```
app.use('/api/*', (_req, res) => {
  res.status(404).json({ error: 'API endpoint not found' });
});

app.use((err, req, res, next) => {
  console.error(err);
  res.status(500).json({ error: 'Internal server error' });
});
```

- **Scope:** Small addition to improve developer experience

1.5 Input Validation & Output Sanitization

- **Files:** Backend routes + frontend rendering components
- **Improvement:** Sanitize HTML before rendering
- **How:**

```
import DOMPurify from 'dompurify';
<div dangerouslySetInnerHTML={{ __html: DOMPurify.sanitize(content) }} />;
```

- **Scope:** Medium. Prevents XSS vulnerabilities

1.6 Password Handling & Secrets

- Passwords hashed with bcrypt ✔️
- Do not log or hardcode credentials/secrets ❗
- Rotate session secrets periodically ✅
- `.gitignore` sensitive files (e.g., `cookies.txt`) ❗

---

2. API & Backend Architecture

2.1 Rate Limiting for APIs

- General rate limiter allows 1000 req/15min, fine for 100 visits/day.
- No change needed now.

2.2 Production Session Store

- PostgreSQL session store already in use ✅
- Ensure `NODE_ENV=production` in deployment ❗

2.3 CORS Policy

- Current config ok (same-origin fetches)
- If domain changes, use:

```
app.use(cors({ origin: 'https://yourdomain.com', credentials: true }));
```

2.4 Include Credentials in Fetches

- **File:** `client/src/lib/queryClient.ts`
- **Improvement:** Always include credentials in fetch
- **How:**

```
fetch(url, {
  method,
  credentials: 'include',
  headers: data ? { 'Content-Type': 'application/json' } : {},
  body: data ? JSON.stringify(data) : null
});
```

- **Scope:** Ensures session cookies are sent

2.5 Consistent Error Format

- Standardize all API errors to `{ error: string }`
- Helps with frontend error handling

2.6 Modularize Routes (Optional)

- Current monolithic `routes.ts` works for small scale
- Suggest: Split into `/routes/users.ts`, `/routes/products.ts`, etc. for maintainability

---

3. Database (Drizzle ORM + PostgreSQL)

3.1 Foreign Keys and Relations

- Ensure `.references()` used properly across all schema files ✅

3.2 Indexes

- Ensure frequently queried fields are indexed (e.g., `slug`, `name`)

3.3 Seed Data

- Auto-seeding okay during dev
- Protect against overwriting data in production ❗

---

4. Front-End Reliability

4.1 Session/Auth Sync

- Auth state is synced correctly ✅
- Invalidate React Query cache manually after state-changing mutations ❗

4.2 Static Assets

- Ensure `/public` folder is used correctly
- Vite includes hashed file names by default ✅

4.3 Tailwind CSS

- Tailwind config is correct ✅

4.4 Error Display

- Catch and display all API errors clearly
- Use toasts or modals for feedback

4.5 Base API URL (Optional)

- Add `VITE_API_BASE_URL` to `.env` for future flexibility

---

5. Performance

5.1 Gzip Compression

- **File:** `server/index.ts`
- **How:**

```
import compression from 'compression';
app.use(compression());
```

- **Scope:** Improves load times for static/API responses

5.2 Logging

- **File:** `server/index.ts`
- **How:**

```
import morgan from 'morgan';
app.use(morgan('combined'));
```

- Adds server-side access logs

Summary Table

Area	File/Location	Action	Priority
HTTPS Config	`index.ts`	Trust proxy + secure headers	High
Session Security	`routes.ts`	Custom cookie name + flags	High
Rate Limiting	`routes.ts`	Add limiter to login	Medium
Error Handling	`index.ts`	Add 404 + global error handler	Medium
HTML Sanitization	Frontend components	Use DOMPurify	Medium
CORS (future)	`index.ts`	Setup allowed origins	Low
API Error Format	All APIs	Standardize `{ error: string }`	Medium
Logging	`index.ts`	Add Morgan logger	Low
Compression	`index.ts`	Use gzip compression	Low
Query Invalidation	Client after mutations	Invalidate/refresh queries	Medium

Final Notes

This app is already well-built for small-scale production. Applying the above enhancements will bring it to a solid professional level, especially in reliability, session handling, error visibility, and user protection. All changes are lightweight and will not introduce bloat, aligning with your traffic expectations and production goal.