Engineering Report

Alexandria Engineering Syndicate & Identity Solutions

PART 1: BACKEND

1) Authorization Bypass [CRITICAL]

File: `src/api/v1/controllers/BookingController.ts` line 79

Code: `if (!booking.userId.toString() === (req.user as any)._id.toString())`

Problem: JavaScript evaluates `!` before `===`. The `!` is applied to the string `booking.userId.toString()` (making it `false`), and then `false === "someId"` is always `false`. The authorization block never executes. Any authenticated user can read any other user's booking by ID.

Fix: Change to `if (booking.userId.toString() !== (req.user as any)._id.toString())`

---

2) Error Message Leak to Client [CRITICAL]

File: `src/app.ts` line 114

Code: `ResponseUtils.send(res, 500, error.message)`

Problem: The raw Node.js `error.message` is returned to the HTTP client, exposing internal stack traces, file paths, database schema hints, and implementation details to attackers.

Fix: Return a generic string (`"Internal server error"`) and log the full error server-side only.

---

3) Packages Updates [CRITICAL]

Package: `request@^2.88.2` is deprecated and `passport-remember-me@^0.0.1` is unmaintained since 2014, `mongoose@^6.2.7` has a more stable update

Problem: Request is officially deprecated since February 2020 and no security releases planned. Any vulnerability discovered in it will never be fixed.

Fix: Replace all request usages with `axios` or the native `fetch` API (Node 18+). Update all the libs to the latest

---

4) Tokens and Passwords Persisted to Database [HIGH]

File: `src/utils/ResponseUtils.ts` lines 65–68
Code:

```
body: JSON.stringify(req.body),
headers: JSON.stringify(req.headers),
```

Problem: Every API response logs the full request body and all headers to MongoDB's `request_logs` collection. This means: login requests write the user's plaintext password to the database; every authenticated request writes `Authorization: Bearer <token>` to the database; Firebase `idToken`, `refreshToken`, and `accessToken` fields in request bodies are all stored permanently. The logs collection becomes a plaintext credential store

Fix: Before persisting, strip sensitive fields. At minimum remove `password`, `authorization`, `idToken`, `accessToken`, `refreshToken`, `token`, `secret` from both body and headers.

---

5) SMTP TLS Disabled [HIGH]

File: `src/helpers/MailSender.ts` line 18

Problem: SMTP connection uses `secure: false` and `rejectUnauthorized: false`. TLS is disabled. All outgoing email, including password reset links, OTP codes, and booking confirmations, can be intercepted by a man-in-the-middle on the network path between the server and the mail provider.

Fix: Enable TLS (`secure: true`) or use STARTTLS with `rejectUnauthorized: true`.

---

6) CORS Allows All Origins in Production [HIGH]

File: `src/cors.ts` lines 4 and 20

Problem: The CORS origin whitelist resolves to an empty array `[]` in production. Combined with `credentials: true`, this configuration is ambiguous and potentially broken, it may allow any origin or block all origins depending on the `cors` library version. There is no explicit allowlist of production domains.

Fix: Set an explicit array of allowed production origins.

---

7) Session Cookie Missing `Secure` and `SameSite` [HIGH]

File: `src/app.ts` line 168

Code: `cookie: { path: '/', httpOnly: true }`

Problem: The admin session cookie has `httpOnly: true` (good) but is missing `secure: true` (travels over plain HTTP) and `sameSite: 'strict'` (vulnerable to CSRF). An attacker on the same network can intercept the session cookie and a CSRF attack can forge admin actions.

Fix: `cookie: { path: '/', httpOnly: true, secure: true, sameSite: 'strict' }`

---

8) Hardcoded Weak Default Credentials [HIGH]

File: `src/config/index.ts` lines 61 and 97

Problem: Cookie secret falls back to `'a-very-very-long-secret-key'` and identity password defaults to `'12345678'` if the corresponding environment variables are unset. If a server is misconfigured or a new environment is stood up without proper env vars, these weak known defaults are used silently.

Fix: Remove all defaults. Call `process.exit(1)` at startup if required secrets are missing.

---

9) Weak JWT and Session Secrets [HIGH]

File: `.env.staging` lines 34 and 36 & `.env.production` lines 29 and 31

Values: `JWT_KEY = "anyLargeRandomValueToken"` and `ADMIN_SESSION_SECRET="Session secret"`

Problem: Both are human-readable strings, not cryptographically random values. A weak JWT secret allows an attacker to forge valid JWT tokens by brute force. A predictable session secret allows session cookie forgery.

Fix: Replace with at least 32 bytes of cryptographically random data.

---

10) File Uploads Without MIME/Size Validation [HIGH]

File: `src/api/web/DashBoard/controllers/ImageController.ts` lines 28-55

Problem: Images are stored without any whitelist check on file extension or MIME type, and without a file size limit. An attacker can upload executable files, scripts, or multi-gigabyte files. There is no check that the uploaded content is actually an image.

Fix: Validate file extension against an allowlist (`jpg`, `jpeg`, `png`, `webp`, `gif`,...), verify MIME type using a magic-bytes check (not just the extension) and enforce a maximum file size.

---

11) No Rate Limiting on File Upload Endpoint [HIGH]

File: `src/app.ts` lines 55 & 56

Problem: The `/dashboard/images` endpoint in DashboardImageController is not covered by any rate limiter. An attacker can flood it with concurrent uploads to exhaust storage, bandwidth, or server memory.

Fix: Apply a dedicated rate limiter (e.g., 10 uploads per minute per IP) to the images endpoint.

---

12) No Per-User Rate Limiting [HIGH]

Problem: The global limiter (300 req / 15 min) applies equally to all users. A single abusive user can consume the entire budget and degrade service for everyone else.

Fix: Add a per-user (by authenticated `userId` and by IP) rate limiter in addition to the global one.

---

13) No Database Migration Versioning [MEDIUM]

Problem: Schema changes are applied ad-hoc and manually to production. There is no versioned migration system. New environments cannot be reproduced reliably.

Fix: Adopt migration versioning. Version every schema change as a numbered migration file committed to source control.

---

14) Pagination Has No Upper Bound [MEDIUM]

File: `src/api/v1/validators/BaseValidator.ts` lines 14-15

Problem: `perPage` allows `{ min: 0 }` (returning zero results is pointless) and has no maximum. A request with `?perPage=999999` forces a full collection scan, exhausting memory and causing a denial of service.

Fix: `isInt({ min: 1, max: 100 })` on `perPage`.

---

15) `container.get()` Called Inside Services [MEDIUM]

File: `src/services/GuestService.ts` line 16, `src/services/AuthResponderService.ts` line 13 and different files

Problem: Services call `container.get()` directly instead of declaring dependencies through `@inject()` constructor parameters. This violates the DI contract, hides dependencies from the IoC container, and makes unit testing impossible.

Fix: Declare all dependencies as constructor parameters with `@inject()`.

---

16) Query Params Cast Directly to DTOs [MEDIUM]

File: Multiple controllers under `src/api/v1/controllers/`

Problem: Patterns like `const filter = req.query as CourseQuery` bypass all validation. Unexpected or malformed fields from query params reach database queries unchecked.

Fix: Use an explicit validation step before casting to extract only known fields.

---

17) MongoDB Connection and App Seed/Setting Failures Do Not Halt Startup [LOW]

File: `src/app.ts` lines 106-108 and lines 176-180

Problem: The MongoDB connection error is caught and logged, but the server continues to start and accepts incoming requests with no database. Seed and job startup calls not awaited or caught `Seed.run(), AppSettings.run(), EventStatusJob.run()` which makes failures invisible.

Fix: On connection failure, either retry or `exit`

---

18) Date Validator Accepts Past and Far-Future Dates [LOW]

File: `src/api/v1/validators/BookingValidator.ts` line 21

Problem: The date field is only validated for format (`YYYY-MM-DD`). A booking for the year 1900 or 2999 passes validation.

Fix: Add `.isAfter(today)` and a reasonable upper bound check.

---

19) No Request Correlation ID [LOW]

Problem: Logs do not carry a per-request correlation ID. When a bug is reported, it is impossible to trace a specific request across the log stream.

Fix: Add middleware that attaches a `X-Request-Id` to every incoming request and includes it in every log entry.

---

20) No CI/CD pipeline [MEDIUM]

Problem: No .github/workflows. Every deployment is a manual SSH session: git pull, then restart the process. Consequences: (1) A broken commit reaches production with zero automated gate. (2) There is no rollback mechanism, recovery from a bad deploy is entirely manual. (3) There is no audit trail of who deployed what and when. (4) The developer's SSH key is the only control between a commit and production.

Fix: Start with a minimal GitHub Actions workflow that runs tsc --noEmit and tests on every push to main. Block merges that fail. Add a deploy step as a second job.

---

21) No Tests [MEDIUM]

Problem: Very low automated test coverage across the entire backend.

Fix: Write unit, integration and end-to-end tests. Coverage percentage should be high.
