List of implemented security measures

- Password strength check (Frontend + API)
- Making use of React's default dynamic content escaping to prevent XSS and don't use dangerouslySetInnerHTML
- As strict as possible Content-Security-Policy
  - Check CSP with tools like https://observatory.mozilla.org/ and https://csp-evaluator.withgoogle.com/

```
form-action 'self';
frame-ancestors 'none';
base-uri 'self';
default-src 'none';
manifest-src 'self';
script-src 'self' https://unpkg.com/pdfjs-dist@2.8.335/build/pdf.worker.min.js blob:;
connect-src https://*.tulip.florist/ blob:;
img-src 'self' blob: data:;
style-src 'self' blob: data:;
worker-src 'self' blob:;
font-src 'self' blob:;
child-src 'self' blob:;
```

- HTTP security headers
  - Check headers with tools like https://observatory.mozilla.org/
  - `Strict-Transport-Security: max-age=31536000` → ensure the use of HTTP**S**
  - `X-Frame-Options: DENY` → protect against clickjacking attacks
  - `X-Content-Type-Options: nosniff` → tell browser stop automatically detecting contents of files → protect against being tricked into incorrectly interpreting files as JavaScript
  - `Referrer-Policy: no-referrer` → protect the privacy of the user by restricting the data that the browser provides when accessing another site
- HTTPS only (Frontend + Backend)

    - Properly configured HTTPS → only allow strong ciphers and protocols
    - Check configurations with tools like Qualys ssllabs.com
- Authentication + Authorisation (email + password)
    - Implemented entire auth logic/flow without pre-build auth services/libraries for educational purposes (therefore probably not as secure as with pre-build solutions)
    - JWT based
    - Stored in “secure” cookies
        - `secure: true`
        - `httpOnly: true`
        - `sameSite: strict`
        - `expires: 10min / 1day`
    - Short lived access_token & long lived refresh_token
        - access_token valid for 10 min
        - refresh_token valid for 1 day
        - refresh_token rotation
        - refresh_token re-use detection
            - if the system detects a reuse of a refresh_token, it invalidates all active refresh_tokens of that user
        - Possibility to invalidate refresh_tokens
- Validate + sanitise input
    - mongoSanitize middleware → prevent noSQL injections
    - validate emails
    - validate passwords (strength check)
- Use of `sameSite` cookies → protect against CSRF attacks
- Usage of argon2
    - hashing + salting passwords
    - configured argon2 parameters to be “slow enough”
- Properly configured CORS

    - Allow only required origins, methods and headers
- Don't disclose information via API
    - Proper error handling
    - Generic errors
- Rate / Size limiting API endpoints / HTTP requests
- Encrypt database and database connections
- Canary users + detection
    - Detect compromised emails / passwords
    - Shut down entire system if canary user activity gets detected
- Monitoring via Sentry & AWS
    - different logging levels + (email) notifications
- Scan application vulnerabilities with Synk, OWASP-ZAP, Mozilla Observatory, Google CSP evaluator
- Strong passwords and 2-factor auth (Yubikeys) for accounts like AWS, MongoDb Atlas, ...