Exam Project Description

This exam project provides an overview of various QA testing practices and demonstrates the practical, everyday use of tools, frameworks, and methodologies. It serves as a representation or a shallow dive into common challenges encountered in QA.

1. Mock Frontend (React + Vite)

A simple user interface was created using React and built with Vite. This UI simulates client-side usage of the Unix timestamp conversion API.

2. API Testing with Cypress

Cypress was used to write modular, maintainable API tests, covering:

- For the purpose of this exam positive, negative and edge cases were created
- Also provided re-usability of util functions and cypress commands

3. UI End-to-End Testing with Cypress

Cypress also covers full end-to-end testing through the React UI. Cypress-mochawesome reporter was utilized to create test reports as a preparation for CI/CD.

- E2E tests were constructed with scalability & maintainability for larger QA operations.

4. Performance Testing with k6

A minimal k6 test script was created to simulate concurrent load on the API. Due to the remote nature of tested API only minimal performance test was created for validate low to medium load.

Test results help validate API responsiveness and reliability under load.

Configuration:

- Virtual users (VUs) simulate real-world concurrent access
- Measured metrics include:
    - Latency, Throughput, Error Rate

Test Strategy and Recommendations

Scaling Test Automation

- Tests are modular and organized by type (API, UI, performance)
- Reusable utilities and data-driven tests support scalability across services

Avoiding Flaky Tests

- Use of Cypress's built-in retry mechanism
- Network requests intercepted and controlled via `cy.intercept`

Test Coverage Balance

- Focus on API tests for logic validation and edge coverage
- UI tests cover critical user flows only, to minimize test runtime and flakiness
- Performance tests ensure baseline responsiveness

API Improvement Suggestions

- Include an OpenAPI/Swagger specification for better integration and testing
- Introduce versioning for long-term maintainability

How to Run

1. Install Dependencies

```
npm install
```

install k6 depending on used OS (https://grafana.com/docs/k6/latest/set-up/install-k6/)

3. Run React.js

To open Cypress in interactive mode:

```
npm run build
```

```
npm run dev
```

3. Run Cypress Tests

To open Cypress in interactive mode:

```
npx cypress open
```

To run tests headlessly: (reports will be generated during headless run)

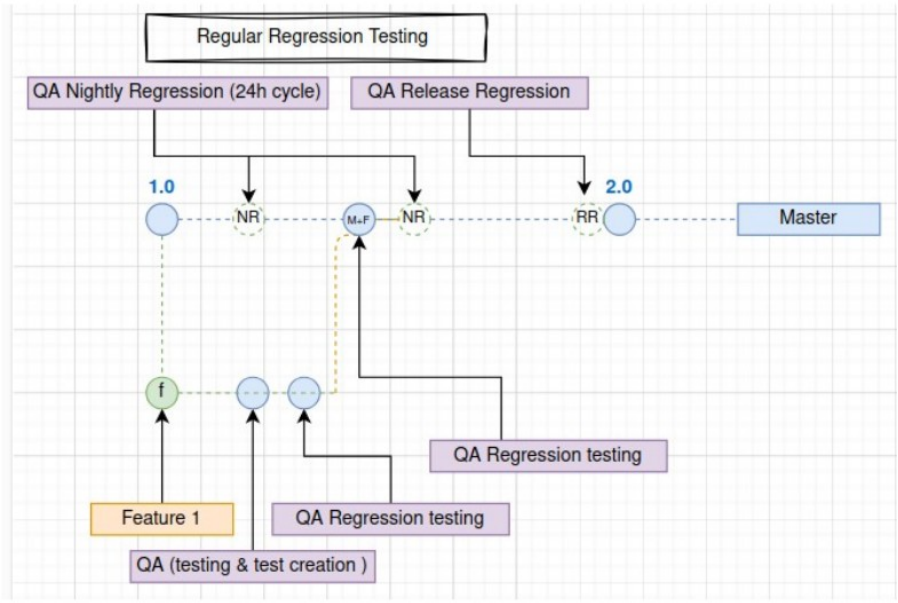
```
npx cypress run
```

4. Run Performance Tests with k6

```
k6 run .\tests\performance\timestamp-api.test.js
```

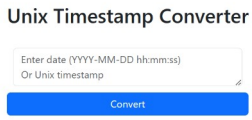
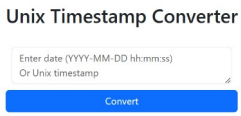
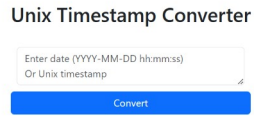
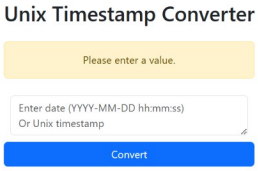
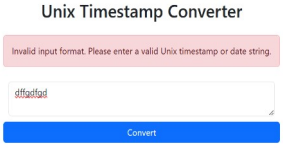
CI Approach:

The Continuous Integration (CI) approach involves creating a GitHub Actions job that will support both daily QA activities for feature testing and periodic regression testing. This workflow will be defined within the appropriate repository, where all code will undergo peer review through pull requests. Test artifacts, such as reports, will be archived after each run and optionally integrated with project management tools like JIRA or Trello as part of the ticket completion process.



Test Plan:

Used ACC model for short identification of functionality and start of QA process. **(ACC)** refers to the ability of a software component or module to exhibit specific attributes or functionalities as intended. It involves validating that each component meets defined requirements, behaves correctly under various conditions.

ID	Attribute	Components	Capabilities
AC1	Valid Date String to Unix		User can input a valid date string and see the correct Unix timestamp returned
AC2	Valid Unix to Date String		User can input a valid Unix timestamp and see the correct 12-hour datetime string returned.
AC3	Invalid Input		Invalid input (non-date, non-timestamp) results in an appropriate error message.
AC4	Empty Input		Empty input triggers alert without making a backend API call.
AC5	Backend Interception		API call to backend is made correctly on valid inputs.
AC6	Alert Messaging		Alerts correctly show success/failure messaging based on input.

Test Coverage Matrix:

Shows in practice what functionality is covered with witch test.

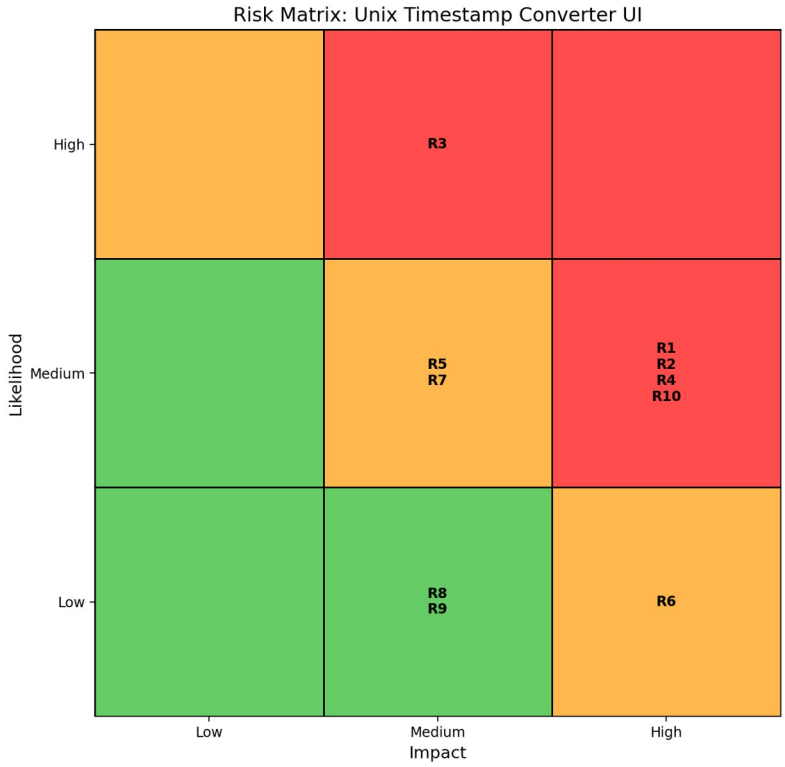
Test Case ID	Description	Covered ACs	Method
TC1	Convert valid datetime string to Unix timestamp	AC1, AC5, AC6	`it('converts valid datetime string to Unix timestamp')`
TC2	Convert valid Unix timestamp to 12h datetime string	AC2, AC5, AC6	`it('converts valid Unix timestamp to datetime string')`
TC3	Handle invalid input (non-date, non-timestamp)	AC3, AC5, AC6	`it('shows false for invalid input')`
TC4	Handle empty input, show	AC4	`it('shows alert message for`

	alert, no API call		`none input')`

Risk identification matrix:

Following table and visual representation of identified risk in tested functionality.

ID	Risk Description	Impact	Likelihood	Risk Level	Mitigation / Test Case
R1	Valid datetime input does not convert to correct Unix timestamp	High	Medium	High	Covered in **TC1** — validates timestamp conversion from datetime
R2	Valid Unix timestamp does not convert to correct datetime	High	Medium	High	Covered in **TC2** — validates datetime conversion from timestamp
R3	Invalid inputs do not show proper error messages	Medium	High	High	Covered in **TC3** — verifies alert for invalid format
R4	Empty input sends API call or breaks app	High	Medium	High	Covered in **TC4** — verifies no API call on empty input
R5	API call fails silently (no alert shown)	Medium	Medium	Medium	Partially covered in **TC1–TC3**, but no direct simulation of API failure
R6	API endpoint changes or is unreachable	High	Low	Medium	Not explicitly covered — suggest mocking failure response
R7	Incorrect success/failure alerts shown	Medium	Medium	Medium	Covered in all test cases — alerts checked after action
R8	Timezone mismatch in displayed date/time	Medium	Low	Low	Out of scope — not currently tested
R9	Inconsistent behavior across browsers	Medium	Low	Low	Out of scope — cross-browser testing not included
R10	Input format changes break conversion	High	Medium	High	Indirectly covered — assumes format compatibility via utility functions


Risk Matrix: Unix Timestamp Converter UI
High
Medium
Low
Likelihood
R3
R5
R7
R1
R2
R4
R10
R8
R9
R6
Low
Medium
High
Impact