Healthcare Application - Testing Documentation

Table of Contents

Overview

React-based healthcare application with role-based workflows for patients and doctors, including authentication, video consultations, prescription management, and profile customization.

For website setup, see the detailed guide in the Setup README.

Testing Focus:

1. Client and server-side mutation testing (Jest + Stryker)
2. Client and server-side bypass testing (Selenium + Axios)

Testing Approach

Unit Testing

Individual component testing for isolated functionality. 11 test files (client-side) and 6 test files (server-side) focus on specific components.

Integration Testing

Multi-component tests implemented in `App.test.jsx` for the client side and `integration` folder for the server side.

Mutation Testing

Fault-based testing using Stryker on critical JSX files and all the backend services to identify test suite gaps.

Bypass Testing

Security validation through payload mutation to test input validation and error handling on both client and server.

Test Suites

Mutation Test Suites

Location: `client/src/__tests__/` - 104 total test cases

Test Suite	Type	Focus Area
`App.test.jsx`	Integration	Routing, authentication, translation
`Appointments.test.jsx`	Unit	Appointment management, filtering
`DoctorDetails.test.jsx`	Unit	Profile fetching, booking
`DoctorNotificationFab.test.jsx`	Unit	Notifications, meeting management
`DoctorSearch.test.jsx`	Unit	Search, filters, parameters
`Login.test.jsx`	Unit	Authentication, token storage
`Meeting.test.jsx`	Unit	Video call lifecycle
`Navbar.test.jsx`	Unit	Navigation, role-based UI
`ProfileChange.test.jsx`	Unit	Patient profile updates
`ProfileChangeDoctor.test.jsx`	Unit	Doctor profile, scheduling
`ProfileDropDown.test.jsx`	Unit	Dropdown, logout
`Register.test.jsx`	Unit	Registration, validation

Location: `server/tests` and `server/integration` - 85 total test cases

Test Suite	Type	Focus Area
`AuthService.test.js`	Unit	Authentication
`AuthService.int.test.js`	Integration	Authentication
`BookingService.test.js`	Unit	Appointments, Prescriptions
`BookingService.int.test.js`	Integration	Appointments, Prescriptions
`DoctorService.test.js`	Unit	Doctor Info, Profile Update
`DoctorService.int.test.js`	Integration	Doctor Info, Profile Update
`MeetService.test.js`	Unit	View, Create, Delete Meeting
`MeetService.int.test.js`	Integration	View, Create, Delete Meeting
`HeartBeatService.test.js`	Unit	Monitor, Join Meeting
`HeartBeatService.int.test.js`	Integration	Monitor, Join Meeting
`UserService.test.js`	Unit	View, Update User Details
`UserService.int.test.js`	Integration	View, Update User Details

Bypass Test Suites

Location: `tests/`

Client-Side: `BlankRegister.test.js`, `BlankLogin.test.js`, `InvalidEmailRegisted.test.js` and `UpdateDoctor.test.js`

- Removes HTML `required` attributes and changes the `type` field via JavaScript
- Submits blank registration form, blank login form, registration form filled with incorrect email format and updates doctor profile with invalid slot information.
- Validates server-side rejection

Server-Side: `BookingTesting.test.js`

- Authenticates users via Selenium
- Extracts tokens from localStorage
- Generates 66 mutated booking payloads
- Tests API robustness with invalid data

Mutation Variations:

- Empty/whitespace strings
- Extremely long strings (500 chars)
- SQL injection patterns
- XSS payloads
- Invalid ObjectId formats
- Wrong data types (objects, numbers, null)
- Extra/nested fields

Mutation Testing

Client-Side

Setup

- **Config:** `client/stryker.config.mjs`
- **Mocks:** `client/test/__mocks__/` (CSS files)
- **Setup:** `client/jest.setup.js` (imports `@testing-library/jest-dom`)

Key Findings

- ~50% mutation kill rate
- Tested important JSX files only
- Translation code produced many surviving mutants
- Most survivors presumed equivalent mutants

Server-Side

Setup

- **Config:** `server/stryker.conf.js`
- **Setup:** `client/jest.config.js`

Key Findings

- ~71% mutation kill rate
- Tested services files only

Bypass Testing

Client-Side

Technology: Selenium WebDriver (Chrome)
Target: Registration form validation bypass
Method: JavaScript injection to remove HTML validation attributes

Server-Side

Technology: Selenium + Axios
Target: Booking API endpoint robustness
Method: Payload mutation to test state validation

Test Process:

1. Authenticate two patients via frontend (Selenium)
2. Extract authentication tokens
3. Authenticate as doctor via API
4. Retrieve valid doctor/slot IDs
5. Generate 66 mutated payloads
6. Send PUT requests to `/booking`
7. Log responses and count errors

Result: 36/66 payloads returned HTTP 500, validating proper error handling

Execution

Prerequisites

```
npm install
```

Run Tests

Client Side

```
# All Jest tests
npx jest

# Specific test file
npx jest src/__tests__/App.test.jsx

# With coverage
npx jest --coverage
```

```
# Mutation testing
npx stryker run
```

Server Side

```
cd server

# All Jest tests
npm test

# Mutation testing
npx stryker run
```

Run Bypass Tests

```
# Start servers first:
# Frontend: http://localhost:5173
# Backend: http://localhost:3000

# Client-side bypass
node tests/BlankRegister.test.js
node tests/BlankLogin.test.js
node tests/InvalidEmailRegister.test.js
node tests/UpdateDoctor.test.js

# Server-side bypass
node tests/BookingTesting.test.js
```

Results

Mutation Testing (Client-Side)

- **Total Tests:** 104
- **Mutation Score:** ~50%
- **Coverage:** Important JSX files

Mutation Testing (Server-Side)

- **Total Tests:** 85
- **Mutation Score:** ~71%
- **Coverage:** All backend services
- **Note:** Translation logic and equivalent mutants account for survivors and upon running stryker it generates a detailed report in `client/report/` (for client-side) and `server/report/` (for server-side) in html format which can be opened in any browser to see detailed results.

Bypass Testing

Client-Side:

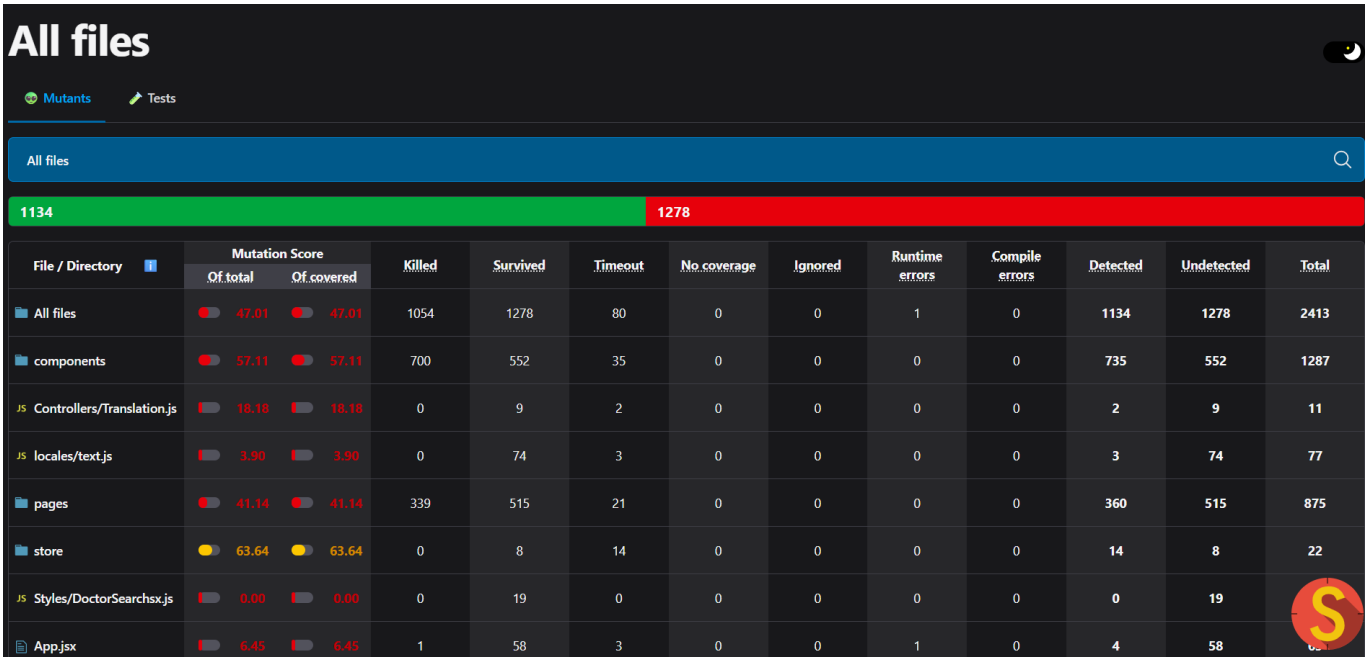
- ✓ Server correctly rejects empty registration submissions
- ✓ Server correctly rejects empty login submissions
- ✗ Server does not reject invalid email addresses
- ✓ Server rejects empty slot submissions
- ✓ Server rejects incorrect data types for slot numbers

Server-Side:

- **Total Mutated Payloads:** 66
- **HTTP 500 Responses:** 36 (54.5%)
- **Validation:** Proper rejection of invalid booking states

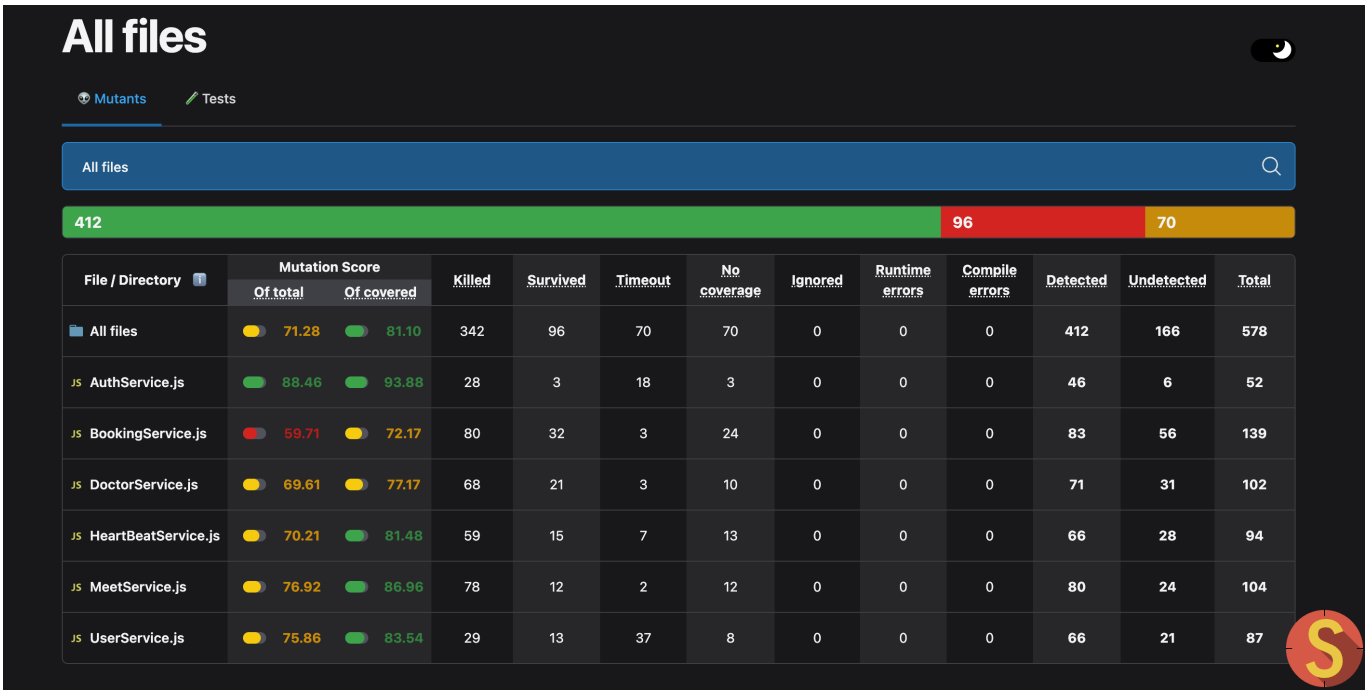
Images

Mutation Testing Coverage



File / Directory	Mutation Score Of total	Mutation Score Of covered	Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
All files	47.01	47.01	1054	1278	80	0	0	1	0	1134	1278	2413
components	57.11	57.11	700	552	35	0	0	0	0	735	552	1287
Controllers/Translation.js	18.18	18.18	0	9	2	0	0	0	0	2	9	11
locales/text.js	3.90	3.90	0	74	3	0	0	0	0	3	74	77
pages	41.14	41.14	339	515	21	0	0	0	0	360	515	875
store	63.64	63.64	0	8	14	0	0	0	0	14	8	22
Styles/DoctorSearchsx.js	0.00	0.00	0	19	0	0	0	0	0	0	19	
App.jsx	6.45	6.45	1	58	3	0	0	1	0	4	58	

Stryker dashboard showing mutation scores and killed/survived mutants for client files.

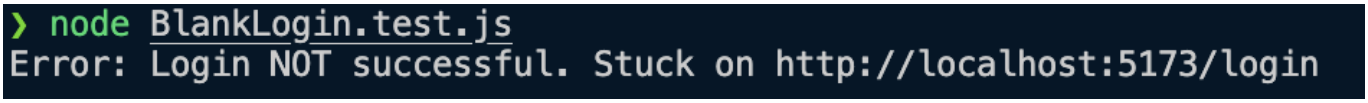


File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
All files	71.28	81.10	342	96	70	70	0	0	0	412	166	578
AuthService.js	88.46	93.88	28	3	18	3	0	0	0	46	6	52
BookingService.js	59.71	72.17	80	32	3	24	0	0	0	83	56	139
DoctorService.js	69.61	77.17	68	21	3	10	0	0	0	71	31	102
HeartBeatService.js	70.21	81.48	59	15	7	13	0	0	0	66	28	94
MeetService.js	76.92	86.96	78	12	2	12	0	0	0	80	24	104
UserService.js	75.86	83.54	29	13	37	8	0	0	0	66	21	87

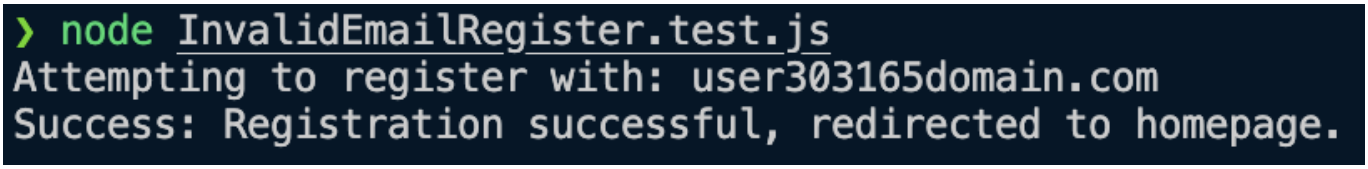
Stryker dashboard showing mutation scores and killed/survived mutants for server files.

Bypass Testing Validation

Terminal output with API error responses from bypass payload tests.



Output of BlankLogin.test.js.



Output of InvalidEmailRegister.test.js.

```
PATIENT
{
  token: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2OTI2N2YxODMxYzg1OGE5Mzc2OTk3MzciLCJpYXQiOjE3NjQxMzA1ODQsImV4cCI6MTc2NDIxNjk4NH0.qXBTUBp-K9AMV8ZAT7HCTY-MD2cjemo1J_X387PYjRw',
  msg: 'signed up successfully',
  status: 200,
  user: {
    _id: new ObjectId("69267f1831c858a937699737"),
    name: 'Test User 303165',
    email: 'user303165domain.com',
    password: '$2b$10$mO3F2XekKyMC3VPJHaMwpu/byFIONTt18JQ1rFcIlIufhPLvXquju',
    role: 'PATIENT',
    user: 'Patient',
    __v: 0
  }
}
```

Terminal output showing the user is registered.

```
❯ node UpdateDoctor.test.js
Pre-step: Registering test user (if not exists)...
Selected 'Doctor' role for registration.
No registration alert appeared (User likely created successfully).
Step 1: Logging in...
Logged in and redirected to doctor profile.

Step 2: Testing Empty Inputs (Bypassing 'required')...
Success: Alert caught for empty inputs:  Invalid slot data provided

Step 3: Testing Invalid Type for Slots...
Navigating back to doctor profile page...
Success: Alert caught for invalid type:  Invalid slot data provided
```

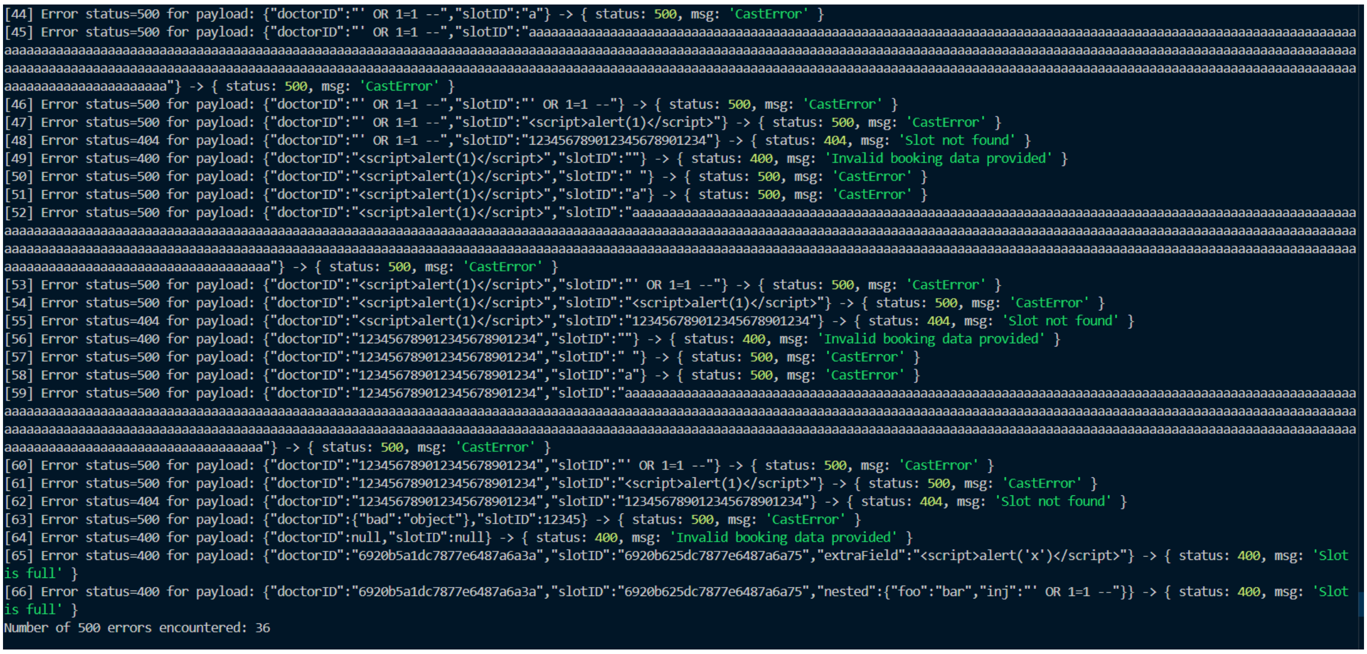
Output of UpdateDoctor.test.js.

```
Error: Invalid slot data provided
    at file:///Users/siddharth/IIITB/Semester7/SoftwareTesting/Project/Health-Care/server/services/DoctorService.js:42:27
    at Array.map (<anonymous>)
    at updateDoctor (file:///Users/siddharth/IIITB/Semester7/SoftwareTesting/Project/Health-Care/server/services/DoctorService.js:40:67)
    at process.processTicksAndRejections (node:internal/process/task_queues:105:5)
    at async file:///Users/siddharth/IIITB/Semester7/SoftwareTesting/Project/Health-Care/server/routers/DoctorRouter.js:14:22
```

Terminal output showing the error response of the server due to an invalid slot update.


```
Error: Invalid slot data provided
    at file:///Users/siddharth/IIITB/Semester7/SoftwareTesting/Project/Health-Care/server/services/DoctorService.js:42:27
    at Array.map (<anonymous>)
    at updateDoctor (file:///Users/siddharth/IIITB/Semester7/SoftwareTesting/Project/Health-Care/server/services/DoctorService.js:40:67)
    at process.processTicksAndRejections (node:internal/process/task_queues:105:5)
    at async file:///Users/siddharth/IIITB/Semester7/SoftwareTesting/Project/Health-Care/server/routers/DoctorRouter.js:14:22
```


Terminal output showing the error response of the server due to an invalid slot update.



```
[44] Error status=500 for payload: {"doctorID":"' OR 1=1 --","slotID":"a"} -> { status: 500, msg: 'CastError' }
[45] Error status=500 for payload: {"doctorID":"' OR 1=1 --","slotID":"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"} -> { status: 500, msg: 'CastError' }
[46] Error status=500 for payload: {"doctorID":"' OR 1=1 --","slotID":"' OR 1=1 --"} -> { status: 500, msg: 'CastError' }
[47] Error status=500 for payload: {"doctorID":"' OR 1=1 --","slotID":"<script>alert(1)</script>"} -> { status: 500, msg: 'CastError' }
[48] Error status=404 for payload: {"doctorID":"' OR 1=1 --","slotID":"123456789012345678901234"} -> { status: 404, msg: 'Slot not found' }
[49] Error status=400 for payload: {"doctorID":"<script>alert(1)</script>","slotID":""} -> { status: 400, msg: 'Invalid booking data provided' }
[50] Error status=500 for payload: {"doctorID":"<script>alert(1)</script>","slotID":" "} -> { status: 500, msg: 'CastError' }
[51] Error status=500 for payload: {"doctorID":"<script>alert(1)</script>","slotID":"a"} -> { status: 500, msg: 'CastError' }
[52] Error status=500 for payload: {"doctorID":"<script>alert(1)</script>","slotID":"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"} -> { status: 500, msg: 'CastError' }
[53] Error status=500 for payload: {"doctorID":"<script>alert(1)</script>","slotID":"' OR 1=1 --"} -> { status: 500, msg: 'CastError' }
[54] Error status=500 for payload: {"doctorID":"<script>alert(1)</script>","slotID":"<script>alert(1)</script>"} -> { status: 500, msg: 'CastError' }
[55] Error status=404 for payload: {"doctorID":"<script>alert(1)</script>","slotID":"123456789012345678901234"} -> { status: 404, msg: 'Slot not found' }
[56] Error status=400 for payload: {"doctorID":"123456789012345678901234","slotID":""} -> { status: 400, msg: 'Invalid booking data provided' }
[57] Error status=500 for payload: {"doctorID":"123456789012345678901234","slotID":" "} -> { status: 500, msg: 'CastError' }
[58] Error status=500 for payload: {"doctorID":"123456789012345678901234","slotID":"a"} -> { status: 500, msg: 'CastError' }
[59] Error status=500 for payload: {"doctorID":"123456789012345678901234","slotID":"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"} -> { status: 500, msg: 'CastError' }
[60] Error status=500 for payload: {"doctorID":"123456789012345678901234","slotID":"' OR 1=1 --"} -> { status: 500, msg: 'CastError' }
[61] Error status=500 for payload: {"doctorID":"123456789012345678901234","slotID":"<script>alert(1)</script>"} -> { status: 500, msg: 'CastError' }
[62] Error status=404 for payload: {"doctorID":"123456789012345678901234","slotID":"123456789012345678901234"} -> { status: 404, msg: 'Slot not found' }
[63] Error status=500 for payload: {"doctorID":{"bad":"object"},"slotID":12345} -> { status: 500, msg: 'CastError' }
[64] Error status=400 for payload: {"doctorID":null,"slotID":null} -> { status: 400, msg: 'Invalid booking data provided' }
[65] Error status=400 for payload: {"doctorID":"6920b5a1dc7877e6487a6a3a","slotID":"6920b625dc7877e6487a6a75","extraField":"<script>alert('x')</script>"} -> { status: 400, msg: 'Slot is full' }
[66] Error status=400 for payload: {"doctorID":"6920b5a1dc7877e6487a6a3a","slotID":"6920b625dc7877e6487a6a75","nested":{"foo":"bar","inj":"' OR 1=1 --"}} -> { status: 400, msg: 'Slot is full' }
Number of 500 errors encountered: 36
```

Server error trace for missing required email field during signup.

Directory Structure

```
client/
├── src/
│   └── __tests__/          # 104 mutation test cases
├── test/
│   └── __mocks__/          # CSS mocks
├── jest.setup.js
└── stryker.config.mjs

server/
├── integration             # Integration Tests
|
├── tests                   # Unit Tests
|
├── jest.config.js
└── stryker.conf.js

testImages/             # Screenshots for README

bypass-tests/
├── BlankRegister.test.js          # Client bypass
├── BlankLogin.test.js             # Client bypass
├── InvalidEmailRegister.test.js # Client bypass
├── UpdateDoctor.test.js           # Client bypass
└── BookingTesting.test.js         # Server bypass
```

Usage of AI

AI tools were actively used to enhance this project's documentation and test code quality. The comments within `.test.js` files were primarily written with the assistance of AI, improving clarity and consistency. While the design of test cases and test logic was mainly my own, GitHub Copilot was sometimes leveraged to generate and suggest code for certain test cases. Additionally, this README's structure and formatting were organized with AI guidance based on content I provided, giving credit for the polished layout and presentation to AI tools.

Contribution

Sai Venkata Sohith Gutta(IMT2022042) - Responsible for client-side mutation testing as well as client-side and server-side bypass testing.

Siddharth Reddy Maramreddy (IMT2022031) - Responsible for server-side mutation testing as well as client-side bypass testing.