**Todo Application - Test Plan & Strategy Document**

1. **Summary**

This document outlines the comprehensive testing strategy for the Todo Application, covering both UI automation and API automation testing. The application consists of a React frontend and Node.js backend API, providing user authentication and CRUD operations for todo items.

Testing Scope: Full-stack todo application with authentication, item management, and data persistence  
Test Environment: Local development environment (localhost)

## 2. What is Being Tested

### **2.1 Application Under Test**

* **Frontend**: React-based Todo Application (<http://localhost:3000>)
* **Backend**: Node.js REST API (<http://localhost:5000/api>/health)

### **2.2 Core Functionalities**

### **User Authentication:**

* Login with valid/invalid credentials
* Session management and token handling
* Access control and authorization

**Todo Item Management:**

* Create new todo items (title, description)
* Read/retrieve existing todo items
* Update todo items (edit title, description, completion status)
* Delete todo items

**3. Test Coverage Areas**

**3.1 UI Automation Coverage (Selenium WebDriver)**

|  |  |  |
| --- | --- | --- |
| **Test Area** | **Coverage** | **Status** |
| Login & View Todo List | Logs in with correct credentials and View list | Pass |
| Create Todo Item | Adds a new items, validation | Pass |
| Update Todo Item | Edits an existing item | Pass |
| Delete Todo Item | Deletes a selected item | Pass |
| Invalid Login Attempt | Verifies rejection of incorrect login | Pass |

**3.2 API Automation Coverage (Python Requests)**

AUTH\_USERNAME=**testuser**  and AUTH\_PASSWORD=**testpass**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Endpoint | Method | | Test Scenario | Coverage |
| /api/auth/login | POST | | User authentication | 100% |
| /api/todos | GET | | Fetch data | 100% |
| /api/todos | POST | Create with valid/invalid data | | 100% |
| /api/todos/:id | PUT | Update existing item | | 100% |
| /api/todos/:id | DELETE | Delete valid/invalid IDs | | 100% |

## 4. Tools and Technologies

## 4.1 UI Automation Stack

**Primary Tool**: Selenium WebDriver with Python

* **Rationale**: Industry standard, robust browser automation, extensive community support
* **Browser**: Chrome (ChromeDriver managed via webdriver-manager)
* **Framework**: Python unittest for test structure and assertions
* **Reporting**: Built-in unittest reporting with custom logging

**Key Libraries**: selenium # Browser automation webdriver-manager # Automatic driver management unittest # Test framework (built-in)

### **4.2 API Automation Stack**

**Primary Tool**: Python Requests Library

* **Rationale**: Simple, powerful HTTP library with excellent JSON support
* **Authentication**: JWT token management
* **Framework**: Python unittest for consistency with UI tests
* **Data Handling**: JSON request/response processing

**Key Libraries**: requests # HTTP client library json # JSON data handling (built-in) unittest # Test framework (built-in) datetime # Timestamp handling (built-in)

### **4.3 Supporting Tools**

* **IDE**: VS Code or Visual Studio for development
* **Version Control**: Git for test code management

## 5. Test Environment Setup

### **5.1 Prerequisites**

* **Application**:Todo app running on localhost:3000 -frontend and localhost:5000 backend
* **Python** **&** **Chrome Browser**: Latest version should be installed
* **Test Credentials** : TEST\_USERNAME=**testuser**  and TEST\_PASSWORD=**testpass**

## 6. How to Run the Tests

### **6.1 UI Automation Execution**

* Navigate to UI test directory or open the script in Visual Studio
* Install dependencies : pip install -r requirements.txt
* Run complete UI test script : python todo\_seelenium\_automation.py
* Log file with successfully UI automation is added in the docs folder in github

**6.2 API Automation Execution**

* Navigate to API test directory or open the script in Visual Studio
* Install dependencies : pip install requests
* Run API test script : python todo\_API\_automation.py
* Log file with successfully API automation is added in the docs folder in github

## 7. Assumptions and Limitations

### **7.1 Assumptions**

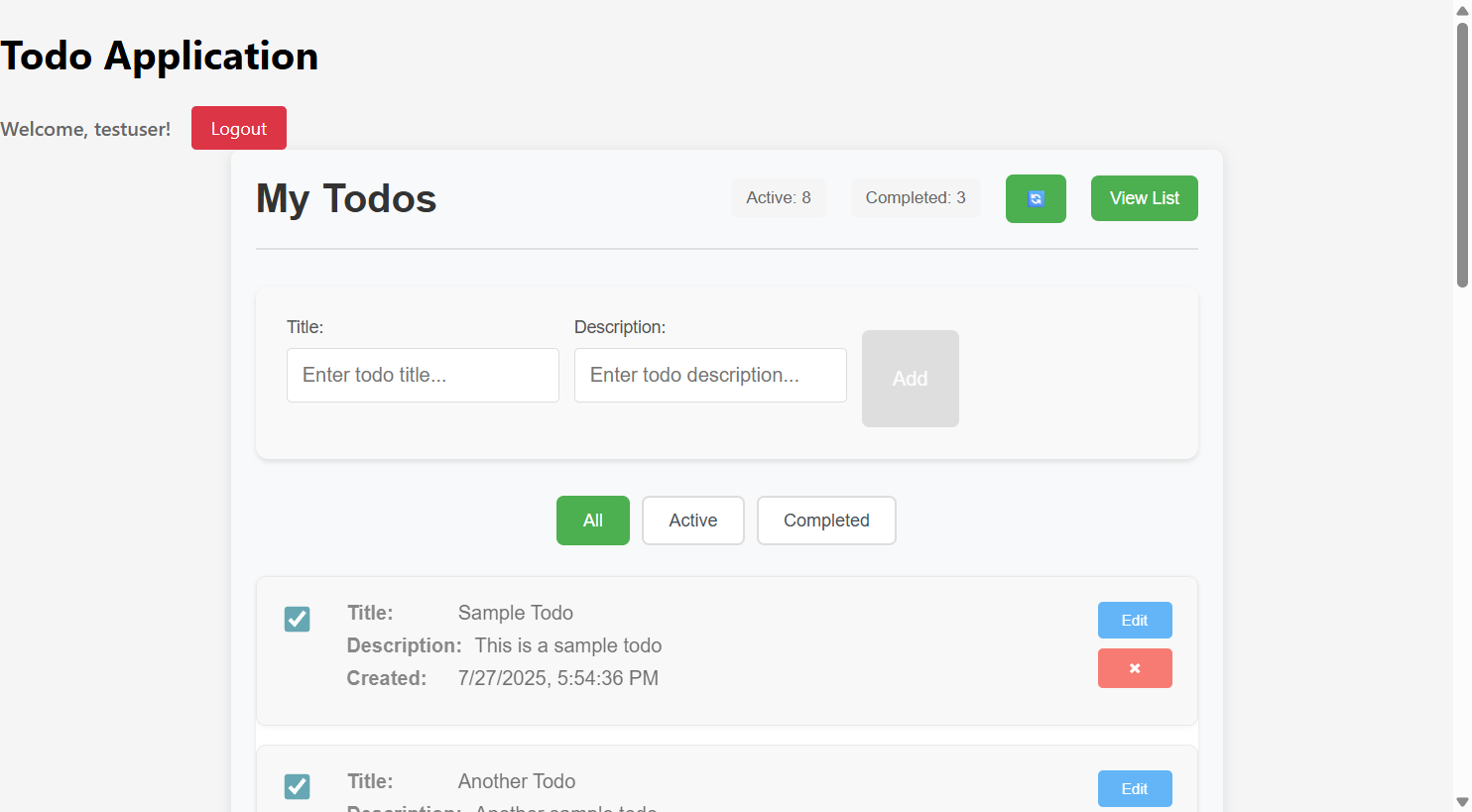
* **Application Availability**: Frontend and backend services are running and accessible
* **Test Environment**: Stable local development environment
* **Browser Compatibility**: Primary testing on Chrome
* **User Permissions**: Test user has full CRUD permissions for todo items

### **7.2 Limitations**

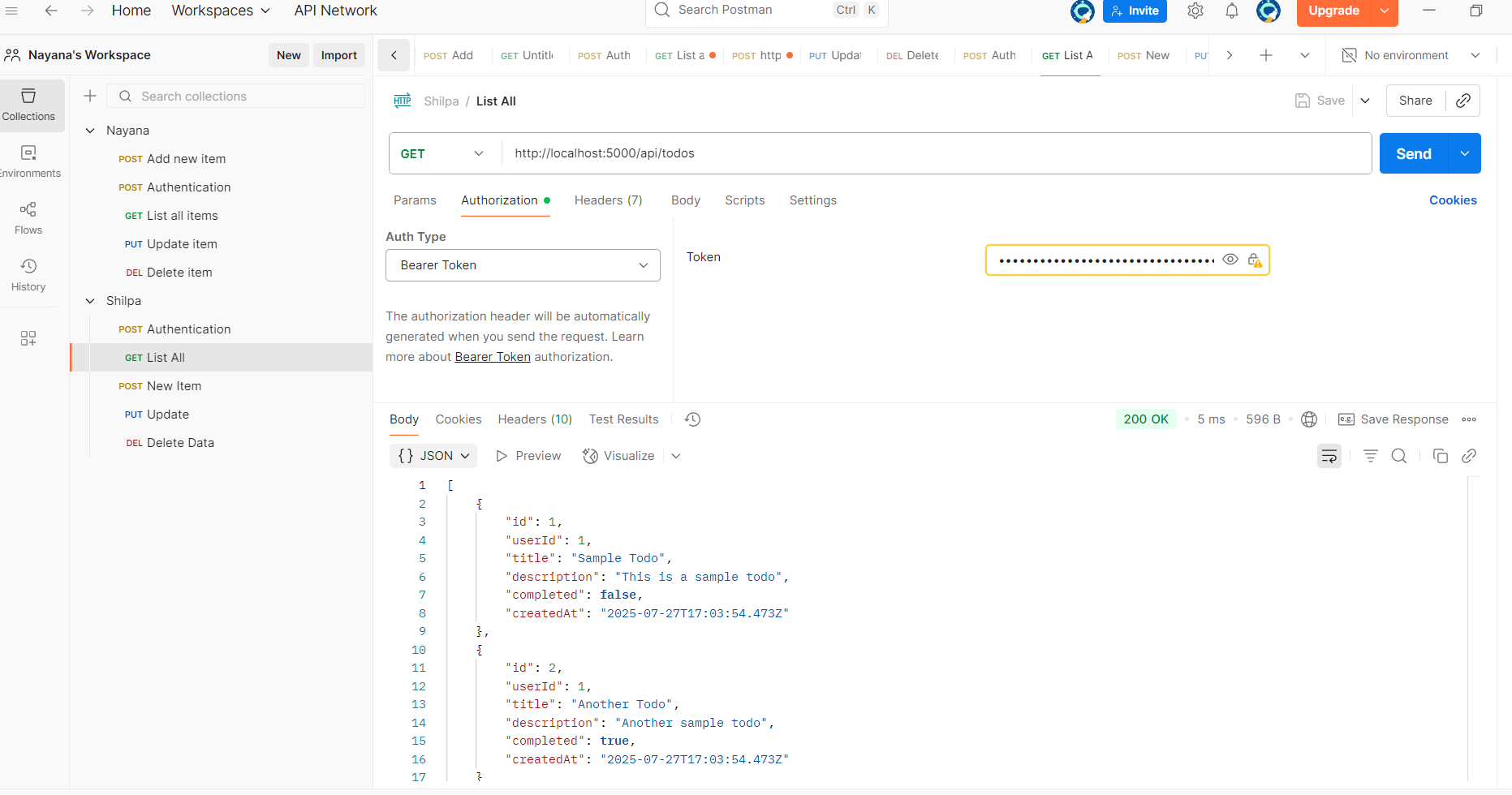
* **Performance Testing**: Basic response time validation only (no load testing)
* **Mobile Testing**: No mobile device or responsive testing included
* **Database Testing**: No direct database validation or performance testing

**8. Screenshots**

* **Screenshot of Logged in Application after successful UI Automation :**

****

* **Screenshot of API testing done manually using POSTMAN :**

****