Advance Guide: How to Use TestSprite MCP

This guide will help you start, run, and fix automated tests using TestSprite MCP.

https://www.testsprite.com/solutions/mcp

1. Get Ready

Before you start:

- **Install TestSprite MCP Server** on your computer.
- Make sure you have a project. This means a website (frontend), an app (backend), or both.
- Check your project is running:
  - For a website, run:
    `npm run dev`
    (Usually runs on 3000, 5173, or 8080.)
  - For an backend, run:
    `node index.js`
    (Usually runs on 4000, 8000, or 3001.)
- Open your IDE (such as Cursor or Windsurf).
- Make folders clear and organized (frontend, backend, docs).
- Prepare test accounts for login.
  (Don't use real usernames or passwords—make up test ones).[1]

2. Start Testing

- In your IDE, open the chat window.
- Type:

```
Can you test this project with TestSprite?
```

- If you want to test only part of your project, drag its folder into the chat.
- Press Enter.

Now, the AI assistant will start helping you!

3. What Happens Next—Step by Step

Here's what TestSprite MCP does automatically:

1. **Sets Up Testing**
   Opens a tool for you to set test options.
2. **Checks Your Project**
   Reads your project's files and understands its features.
3. **Handles PRD Document**
   You upload your PRD (Product Requirements Document). The tool creates a clean, easy-to-use version for tests.
4. **Makes Test Plan**
   It writes test cases for the website and app, making sure everything gets checked.
5. **Runs Tests**
   It writes code for tests, runs it in a safe online sandbox, and gives you reports.
6. **Fixes Bugs Automatically**
   After testing, you can ask the AI:

   ```
   Please fix the codebase based on TestSprite testing results.
   ```

The assistant finds the broken parts, fixes them, tests again, and keeps working until it's all good.[1]

4. Setting Things Up

The setup tool asks for:

- **Test Account Details**

  - Username: test@example.com
  - Password: your-test-password
- **App Links**
  - Website: http://localhost:5173
  - App: http://localhost:4000
- **PRD**
  Upload your PRD file. A simple one is OK!

5. See Your Results

Look in the `testsprite_tests` folder. There you'll find:

- PRD files
- Test settings
- Project summary
- Final, easy-to-read test reports (markdown and HTML)
- Small files for each test (like: TC001_Login_Success_with_Valid_Credentials.py)
- Results file with list of passed and failed tests

Reports show:

- Total tests
- How many passed or failed
- Coverage (how much of your code was tested)
- Why a test failed (and what to fix)

6. Best Tips

- Make sure your website and app run and can be accessed.
- Use easy-to-understand folder names.

- Always have fake test accounts.
- Read the PRD and test plan—make sure they match your project.
- Fix your code using AI assistant after tests.

Example Commands

- Start tests:

```
Can you test this project with TestSprite?
```

- Fix code after failed tests:

```
Please fix the codebase based on TestSprite testing results.
```

TestSprite MCP makes it fast and easy to test your code, get detailed reports, and fix problems—all with simple commands and guidance.[1]

⁂

1. For more info FOLLOW: https://docs.testsprite.com/mcp/quickstart

PROMPT: Youtube Watch Together

Build a WebRTC-based 'Watch Together' web app where two users can watch the same YouTube video in sync and chat in real time.

Requirements:

1. Frontend:
   - A simple page with:
     - An input box to paste a YouTube URL.

    - A shared YouTube video player (use YouTube IFrame API).
    - A chat panel (messages displayed with timestamps and usernames).
    - A 'Connect' button to start WebRTC connection.

2. WebRTC Logic:
    - Use WebRTC Peer-to-Peer connection with a signaling server (WebSocket/Express+Socket.io).
    - Sync YouTube player states (play, pause, seek, current time) between both peers.
    - Ensure low-latency syncing (use player.getCurrentTime() and send updates on actions).
3. Chat:
    - Implement data channel over WebRTC for sending messages.
    - Messages should appear instantly on both sides.
4. Signaling Server (Node.js + Socket.io):
    - Handles exchange of SDP offers/answers and ICE candidates.
    - Manages room creation (2 users per room).
5. Tech stack:
    - Frontend: HTML, CSS, JavaScript (YouTube IFrame API + WebRTC).
    - Backend: Node.js with Express + Socket.io for signaling.
6. Behavior:
    - When one user plays/pauses/seeks the video, the other user's player updates instantly.
    - Both users can chat via the WebRTC data channel.

    - The system should handle network drops gracefully (reconnect if possible).

Output complete working code for:

- server.js (signaling server).
- index.html + script.js (frontend with YouTube sync + WebRTC chat).

Keep the UI minimal but functional.