0. Folder setup

Open a terminal and create a fresh project:

```
mkdir cypress-cucumber-express-demo
cd cypress-cucumber-express-demo
npm init -y
```

---

1. Create a simple Express app

1.1 Install Express

```
npm install express
```

1.2 Create `server.js`

In the project root, create `server.js`:

```
// server.js
const express = require("express");
const app = express();

app.use(express.json());

// Simple health-check endpoint
app.get("/health", (req, res) => {
  res.json({ status: "ok" });
});

// Very simple login endpoint
app.post("/login", (req, res) => {
  const { username, password } = req.body;

  if (username === "admin" && password === "secret") {
    return res.json({ success: true, message: "Login successful" });
  }

  return res.status(401).json({ success: false, message: "Invalid credentials" });
});
```

```
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`🚀 Server running at http://localhost:${PORT}`);
});
```

1.3 Add start script in `package.json`

In `package.json`, inside `"scripts"`, add:

```
"scripts": {
  "start": "node server.js",
  "test": "echo \"No tests yet\""
}
```

(Keep any existing script like `"test"` if you want; just add `"start"`.)

1.4 Run and check the app

In one terminal:

```
npm start
```

You should see:

🚀 Server running at http://localhost:3000

Quick sanity check (another terminal):

```
curl http://localhost:3000/health
# {"status":"ok"}
```

App is ready. ✅

---

2. Install Cypress + Cucumber preprocessor

We'll use the official `@badeball/cypress-cucumber-preprocessor` and esbuild bundler. (LambdaTest)

npm install --save-dev cypress @badeball/cypress-cucumber-preprocessor @bahmutov/cypress-esbuild-preprocessor

Add Cypress scripts to `package.json`:

```
"scripts": {
  "start": "node server.js",
  "cypress:open": "cypress open",
  "cypress:run": "cypress run"
}
```

---

3. Initialize Cypress project structure

Run:

npx cypress open

- Choose **E2E Testing**.
- Pick any browser.
- Cypress will create a `cypress/` folder and `cypress.config.js`.

Close Cypress for now.

---

4. Configure Cypress + Cucumber plugin

4.1 Update `cypress.config.js`

Open `cypress.config.js` and replace contents with:

```
// cypress.config.js
const { defineConfig } = require("cypress");
const createBundler = require("@bahmutov/cypress-esbuild-preprocessor");
const {
  addCucumberPreprocessorPlugin,
```

```
} = require("@badeball/cypress-cucumber-preprocessor");
const {
  createEsbuildPlugin,
} = require("@badeball/cypress-cucumber-preprocessor/esbuild");

// this function will be used in e2e.setupNodeEvents
async function setupNodeEvents(on, config) {
  // register the Cucumber plugin
  await addCucumberPreprocessorPlugin(on, config);

  // tell Cypress to use esbuild for preprocessing feature + JS files
  on(
    "file:preprocessor",
    createBundler({
      plugins: [createEsbuildPlugin(config)],
    })
  );

  return config;
}

module.exports = defineConfig({
  e2e: {
    baseUrl: "http://localhost:3000", // our Express app
    specPattern: "cypress/e2e/**/*.feature", // look for .feature files
    setupNodeEvents,
  },
});
```

This is basically what the latest docs recommend (Cypress 10+ + Badeball preprocessor). (LambdaTest)

4.2 Tell the preprocessor where step definitions live

In `package.json`, add a config section:

```
"cypress-cucumber-preprocessor": {
  "stepDefinitions": [
    "cypress/e2e/[filepath].{js,ts}",
    "cypress/support/step_definitions/**/*.{js,ts}"
  ]
}
```

Important idea:
If you have `cypress/e2e/login.feature`, the first pattern says:
➡ look for step file `cypress/e2e/login.js` (same name, no extension).

---

5. Write a BDD test (feature file)

Now we do the **"Writing BDD-style tests"** bit.

Create `cypress/e2e/login.feature`:

```
Feature: Login API

  As a client of the API
  I want to log in with valid credentials
  So that I can access protected resources

  Scenario: Successful login with valid credentials
    Given the API is running
    When I login with username "admin" and password "secret"
    Then I should get a 200 response
    And the response should indicate success

  Scenario: Failed login with invalid credentials
    Given the API is running
    When I login with username "wrong" and password "user"
    Then I should get a 401 response
    And the response should indicate failure
```

This is pure **Gherkin/BDD**:

- **Feature / Scenario** → business description
- **Given** → precondition
- **When** → action
- **Then / And** → expectations

---

6. Implement step definitions (Cypress + Cucumber)

Create `cypress/e2e/login.js` (note: same base name as `login.feature`):

```
// cypress/e2e/login.js
const { Given, When, Then } = require("@badeball/cypress-cucumber-preprocessor");

let response; // will hold the response from cy.request

Given("the API is running", () => {
  cy.request("GET", "/health")
    .its("status")
    .should("eq", 200);
});

When(
  "I login with username {string} and password {string}",
  (username, password) => {
    cy.request({
      method: "POST",
      url: "/login",
      failOnStatusCode: false, // allow 4xx/5xx, we'll assert manually
      body: { username, password },
    }).then((res) => {
      response = res;
    });
  }
);

Then("I should get a {int} response", (statusCode) => {
  expect(response.status).to.eq(statusCode);
});

Then("the response should indicate success", () => {
  expect(response.body).to.have.property("success", true);
});

Then("the response should indicate failure", () => {
  expect(response.body).to.have.property("success", false);
});
```

What’s happening:

- We import **Given / When / Then** from the preprocessor. (LambdaTest)
- Steps with `{string}` and `{int}` are **parameterized** BDD steps → reusable, less duplication.
- We use `cy.request` since this is an API-only Express app.
- We store the response in a local variable `response` and assert in later steps.

At this point you have:

```
cypress-cucumber-express-demo
├─ server.js
├─ cypress.config.js
├─ cypress
│  └─ e2e
│     ├─ login.feature
│     └─ login.js
└─ package.json
```

---

7. Run the feature file with Cypress

Now we cover **"Running feature files with Cypress"**.

7.1 Start the Express server

In **Terminal 1**:

```
npm start
# Server running at http://localhost:3000
```

7.2 Run Cypress in headed mode (GUI)

In **Terminal 2**:

```
npm run cypress:open
```

- Choose **E2E Testing**.

- Select browser.
- You should see `login.feature` in the spec list.
- Click it → Cypress runs the BDD scenarios defined in the `.feature` file.

7.3 Run headless in CLI

npm run cypress:run

This will pick `login.feature` (because of `specPattern`) and execute both scenarios headless (CI-style).

---

8. How each of your bullets is covered

5. Integration of Cucumber with Cypress

1. **Setting up Cypress-Cucumber preprocessor**

Installed:

npm install --save-dev @badeball/cypress-cucumber-preprocessor @bahmutov/cypress-esbuild-preprocessor

-
- Config in `cypress.config.js` (`setupNodeEvents`, `file:preprocessor`, `specPattern`). (LambdaTest)
- `package.json` `cypress-cucumber-preprocessor` block to resolve stepDefs.

2. **Writing BDD-style tests**
    - `login.feature` with `Feature`, `Scenario`, `Given/When/Then`.
    - Parameterized steps: `{string}`, `{int}` etc.
    - Step definitions in `login.js` mapping Gherkin → Cypress commands.

3. **Running feature files with Cypress**
    - `specPattern: "cypress/e2e/**/*.feature"` so `.feature` files are first-class specs.
    - `npm run cypress:open` (GUI, good for beginners).
    - `npm run cypress:run` (headless, CI-friendly).

If you want, next step we can:

- Convert this to **UI E2E** (simple HTML login page) using the same BDD steps, or
- Add **tags** in feature files (`@smoke`, `@regression`) and run only specific BDD scenarios.