Cool, let's turn "UI Coverage" into something *very* concrete with a tiny Node.js + Express CRUD app and Cypress.

We'll go step-by-step:

1. Build a **simple CRUD UI** (tasks TODO list).
2. Add **Cypress** and write one basic UI test.
3. Add **visual coverage**: screenshots + videos.
4. See how to **track UI changes on every commit** (CI idea).
5. **Reduce test duplication** (best practices).
6. Add **UI coverage reporting / metrics**.

I'll assume you have Node.js installed.

---

0. Create project & install dependencies

```
mkdir ui-coverage-crud
cd ui-coverage-crud
npm init -y

# Express app
npm install express

# For auto restart in dev (optional)
npm install --save-dev nodemon

# Cypress for UI tests
npm install --save-dev cypress
```

Update `package.json` scripts:

```
{
  "scripts": {
    "dev": "nodemon app.js",
    "start": "node app.js",
    "cypress:open": "cypress open",
    "cypress:run": "cypress run"
  }
}
```

---

1. Simple Node.js CRUD app (Tasks list)

1.1 Create `app.js`

```
// app.js
const express = require('express');
const path = require('path');

const app = express();
const PORT = 3000;

// In-memory "database"
let tasks = [
  { id: 1, title: 'Learn Cypress', done: false },
  { id: 2, title: 'Build CRUD app', done: false }
];

app.use(express.urlencoded({ extended: true }));
app.use(express.static(path.join(__dirname, 'public')));

// List tasks (Home)
app.get('/', (req, res) => {
  let html = `
    <html>
      <head>
        <title>Tasks CRUD</title>
      </head>
      <body>
        <h1 data-cy="page-title">Tasks CRUD</h1>

        <form action="/tasks" method="POST" data-cy="add-task-form">
          <input name="title" placeholder="New task" data-cy="new-task-input" />
          <button type="submit" data-cy="add-task-btn">Add</button>
        </form>

        <ul data-cy="task-list">
          ${tasks
          .map(
            (t) => `
              <li data-cy="task-item">
                <span data-cy="task-title">${t.title}</span>
                <a href="/tasks/${t.id}/edit" data-cy="edit-task-link">Edit</a>
                <form action="/tasks/${t.id}/delete" method="POST" style="display:inline;">
                  <button type="submit" data-cy="delete-task-btn">Delete</button>
                </form>
              </li>
            `
          )
          .join('')}
        </ul>
      </body>
    </html>
  `;
  res.send(html);
});

// Create task
app.post('/tasks', (req, res) => {
  const { title } = req.body;
  const id = tasks.length ? tasks[tasks.length - 1].id + 1 : 1;
  tasks.push({ id, title, done: false });
  res.redirect('/');
});

// Edit form
app.get('/tasks/:id/edit', (req, res) => {
  const id = Number(req.params.id);
  const task = tasks.find((t) => t.id === id);

  if (!task) return res.status(404).send('Task not found');

  let html = `
    <html>
      <head>
        <title>Edit Task</title>
      </head>
      <body>
        <h1 data-cy="edit-page-title">Edit Task</h1>
        <form action="/tasks/${task.id}/edit" method="POST" data-cy="edit-task-form">
          <input name="title" value="${task.title}" data-cy="edit-task-input" />
          <button type="submit" data-cy="save-task-btn">Save</button>
        </form>
        <a href="/" data-cy="back-link">Back</a>
      </body>
    </html>
  `;
  res.send(html);
});

// Update task
app.post('/tasks/:id/edit', (req, res) => {
  const id = Number(req.params.id);
  const { title } = req.body;
  const task = tasks.find((t) => t.id === id);

  if (task) task.title = title;
  res.redirect('/');
});

// Delete task
app.post('/tasks/:id/delete', (req, res) => {
  const id = Number(req.params.id);
  tasks = tasks.filter((t) => t.id !== id);
  res.redirect('/');
});

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

Run the app:

```
npm run dev
# or
npm start
```

Open `http://localhost:3000` – you should see your simple Tasks CRUD page.

---

2. Set up Cypress

2.1 Initialize Cypress

```
npx cypress open
```

- Choose **E2E Testing**.
- It will create a `cypress` folder and a `cypress.config.js` file.

Set `baseUrl` in `cypress.config.js`:

```
const { defineConfig } = require('cypress');

module.exports = defineConfig({
  e2e: {
    baseUrl: 'http://localhost:3000',
    video: true,              // record videos on cypress run
    screenshotOnRunFailure: true
  }
});
```

2.2 Create a basic UI test

Create file: `cypress/e2e/tasks-ui.cy.js`

```
// cypress/e2e/tasks-ui.cy.js
describe('Tasks CRUD UI', () => {
  beforeEach(() => {
    cy.visit('/');
  });

  it('shows the tasks list', () => {
    cy.get('[data-cy="page-title"]').should('contain', 'Tasks CRUD');
    cy.get('[data-cy="task-list"]').should('exist');
    cy.get('[data-cy="task-item"]').should('have.length.at.least', 1);
  });

  it('can add a new task', () => {
    cy.get('[data-cy="new-task-input"]').type('Write Cypress tests');
    cy.get('[data-cy="add-task-btn"]').click();

    cy.get('[data-cy="task-title"]').contains('Write Cypress tests');
  });
});
```

Run tests in headed mode:

```
npm run cypress:open
```

Run in headless mode:

```
npm run cypress:run
```

Headless mode is important for **videos & screenshots** (visual coverage).

---

3. Visual Coverage Monitoring

3.1 Automatic screenshots & videos

With these settings in `cypress.config.js`:

```
video: true,
screenshotOnRunFailure: true
```

When you run:

```
npm run cypress:run
```

Cypress will create:

- **Videos** in `cypress/videos/`
- **Screenshots on failure** in `cypress/screenshots/`

> These files are your basic *visual coverage* artefacts – they show what the UI looked like when tests ran.

3.2 Manual screenshots in tests

You can also capture screenshots at specific points to “cover” critical UI states.

Update `tasks-ui.cy.js`:

```
it('can add a new task (with screenshots)', () => {
  // Initial state
  cy.screenshot('home-before-add');

  cy.get('[data-cy="new-task-input"]').type('Write Cypress tests');
  cy.get('[data-cy="add-task-btn"]').click();

  // State after adding
  cy.get('[data-cy="task-title"]').contains('Write Cypress tests');

  cy.screenshot('home-after-add');
});
```

When you run `cypress run`, you'll get:

- `cypress/screenshots/tasks-ui.cy.js/home-before-add.png`
- `cypress/screenshots/tasks-ui.cy.js/home-after-add.png`

These show **before** and **after** UI snapshots.

You can repeat this pattern for:

- Edit page
- Delete confirmation
- Error message states (later if you add validation)

---

4. Tracking UI Changes on Every Commit

Idea: **Every time you push to Git**, CI runs Cypress in headless mode and stores screenshots & videos as artifacts.

I'll show GitHub Actions (concept is same for Jenkins/GitLab).

4.1 Add GitHub Actions workflow

Create `.github/workflows/cypress.yml`:

```
name: Cypress UI Tests

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  cypress-run:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Setup Node
        uses: actions/setup-node@v4
        with:
          node-version: '20'

      - name: Install dependencies
        run: npm install

      - name: Start app
        run: npm run start &
        env:
          NODE_ENV: test

      - name: Wait for app to be ready
        run: npx wait-on http://localhost:3000

      - name: Run Cypress tests
        run: npx cypress run

      - name: Upload Cypress artifacts (screenshots & videos)
        uses: actions/upload-artifact@v4
        if: always()
        with:
          name: cypress-artifacts
          path: |
            cypress/screenshots
      cypress/videos
```

What this gives you:

- On **every push / PR**:
    - App starts
    - Cypress runs
    - Screenshots/videos uploaded as **artifacts**
- You can open those from GitHub UI and compare runs over time → effectively tracking UI changes.

> For Jenkins/GitLab you'd do similar: run `npm install`, start server, run `cypress run`, archive `cypress/screenshots` and `cypress/videos`.

---

5. Reducing Test Duplication (Best Practices)

We don't want to repeat "visit home, add task" everywhere.

5.1 Use `beforeEach` and helper functions

Refactor `tasks-ui.cy.js`:

```
// cypress/e2e/tasks-ui.cy.js
function addTask(title) {
  cy.get('[data-cy="new-task-input"]').clear().type(title);
  cy.get('[data-cy="add-task-btn"]').click();
}

describe('Tasks CRUD UI', () => {
  beforeEach(() => {
    cy.visit('/');
  });

  it('shows the tasks list', () => {
    cy.get('[data-cy="page-title"]').should('contain', 'Tasks CRUD');
    cy.get('[data-cy="task-list"]').should('exist');
  });

  it('can add a new task', () => {
    addTask('Write Cypress tests');
    cy.get('[data-cy="task-title"]').contains('Write Cypress tests');
  });

  it('can edit a task', () => {
    addTask('Old Title');

    cy.contains('[data-cy="task-item"]', 'Old Title')
      .find('[data-cy="edit-task-link"]')
      .click();

    cy.get('[data-cy="edit-task-input"]')
      .clear()
      .type('Updated Title');

    cy.get('[data-cy="save-task-btn"]').click();

    cy.get('[data-cy="task-title"]').contains('Updated Title');
  });

  it('can delete a task', () => {
    addTask('Task to delete');

    cy.contains('[data-cy="task-item"]', 'Task to delete')
      .find('[data-cy="delete-task-btn"]')
      .click();

    cy.get('[data-cy="task-title"]').should('not.contain', 'Task to delete');
  });
});
```

Patterns that avoid duplication:

- Use **helper functions** (`addTask`) inside the spec.
- Use **`beforeEach`** to handle common “go to home page”.
- Use **`data-cy`** attributes for selectors so UI changes (CSS classes, text) don’t break tests.

5.2 Move helpers to `commands.js` (for reuse)

Edit `cypress/support/commands.js`:

```
// cypress/support/commands.js
Cypress.Commands.add('addTask', (title) => {
  cy.get('[data-cy="new-task-input"]').clear().type(title);
  cy.get('[data-cy="add-task-btn"]').click();
});
```

Then in your spec:

```
// cypress/e2e/tasks-ui.cy.js
describe('Tasks CRUD UI', () => {
  beforeEach(() => {
    cy.visit('/');
  });

  it('can add a new task', () => {
    cy.addTask('Write Cypress tests');
    cy.get('[data-cy="task-title"]').contains('Write Cypress tests');
  });

  // use cy.addTask in other tests too
});
```

Now any CRUD spec can reuse `cy.addTask` → less duplication.

---

6. UI Coverage Reporting & Metrics

For a beginner, start with **simple, practical metrics**:

6.1 What you already get from Cypress

- **Number of tests, passes, failures**
  Shown in CLI and HTML runner.

- **Screenshots & videos count**
  More specs + more states = broader visual coverage.

- **Spec organization**
  E.g., `tasks-ui.crud.cy.js`, `tasks-ui-errors.cy.js`, etc.

6.2 Basic reporting

1. Store Cypress outputs in CI artifacts (we already did in GitHub Action).
2. Optionally enable **JUnit reports** or **mochawesome** to get HTML/JSON reports:
   - Then you can track:
     - Total tests
     - Failing tests
     - Duration per spec

(We can wire this up later, step-by-step, if you want.)

6.3 Conceptual “UI coverage” checklist

For this CRUD app, you can say your UI is well covered when:

- **All key pages** have tests:
  - Home (list)
  - Edit page
- **All key flows** are tested:
  - Add task
  - Edit task
  - Delete task
- **Different states** have screenshots:
  - Empty list (if you add that behaviour later)

    - List with tasks
    - After edit
    - After delete
- **CI runs on every commit** and stores artifacts.

That's a simple but realistic definition of "UI coverage" for a beginner.

Recap

With this one small CRUD app you now covered:

1. **Visual Coverage Monitoring**
    - Cypress screenshots & videos (automatic + manual with `cy.screenshot()`).
2. **Tracking UI Changes on Every Commit**
    - CI pipeline (GitHub Actions example) running Cypress and uploading artifacts.
3. **Reducing Test Duplication**
    - `beforeEach`, helper functions, `cy.addTask()` custom command, `data-cy` selectors.
4. **UI Coverage Reporting & Metrics**
    - Tests + artifacts + simple coverage checklist, and CI reports.

If you want, next we can:

- Add **error states** (validation) and cover them visually.
- Or plug in a **visual regression tool** (Percy/Applitools) on top of this same app.