# How to Handle Multiple Environments in Your Playwright + Cucumber (TypeScript) Project

Managing different environments (like dev, test, stage, prod) is important in test automation. Here's a step-by-step guide to set up environment-based configuration using `.env` files.

---

## Step 1: Create the `env` Package

- Go to your project folder.
- Create a folder:
  `src/helper/env/`
- Inside that, create a file:
  `env.ts`

---

## Step 2: Install Required Dependencies

Run the following commands:

```
npm install dotenv --save-dev
npm install cross-env --save-dev
```

- `dotenv` helps load environment variables from `.env` files.
- `cross-env` allows setting environment variables in scripts (cross-platform).

---

## Step 3: Setup `env.ts`

Add this code in `src/helper/env/env.ts`:

```
import * as dotenv from 'dotenv';

export const getEnv = () => {
  dotenv.config({
    override: true,
    path: '' // Will be updated dynamically later
  });
};
```

---

## Step 4: Create `.env` Files for Each Environment

Create separate environment files in `src/helper/env/`:

- **Production** → `.env.prod`

  ```
  BASEURL=https://prod.example.com
  BROWSER=chrome
  ```

- **Staging** → `.env.stage`

  ```
  BASEURL=https://stage.example.com
  BROWSER=firefox
  ```

- You can also create `.env.dev`, `.env.test`, etc., for other environments.

---

## Step 5: Dynamically Load Env File Based on `ENV` Variable

Update `getEnv()` in `env.ts`:

```
import * as dotenv from 'dotenv';

export const getEnv = () => {
  dotenv.config({
    override: true,
    path: `src/helper/env/.env.${process.env.ENV}`
  });
};
```

This will automatically pick the correct `.env` file based on the `ENV` value passed in the command.

---

## Step 6: Use Env Variables in Your Code

Example (in `login.ts` or step file):

```
Given("User navigates to the application", async function() {
  await pageFixture.page.goto(process.env.BASEURL as string);
});
```

This uses the `BASEURL` from the selected `.env` file.

---

## Step 7: Update `package.json` Scripts

In your `package.json`, update your scripts to pass the `ENV`:

```
"scripts": {
  "pretest": "npx ts-node src/helper/init.ts",
  "test": "cross-env ENV=prod cucumber-js || true",
  "test:stage": "cross-env ENV=stage cucumber-js",
  "posttest": "npx ts-node src/helper/report.ts",
  "test:failed": "cucumber-js -p rerun @rerun.txt"
}
```

---

## How to Run Tests for Specific Environments

- **Run tests in Production:**

  ```
  npm run test
  ```

- **Run tests in Staging:**

  ```
  npm run test:stage
  ```

- **Alternative (direct command):**

  ```
  cross-env ENV=stage npm run test
  ```

On Windows, always use `cross-env`.
On Unix/macOS, `ENV=stage npm run test` also works.

---

## Summary Table

| Step | What It Does |
|---|---|
| `env.ts` | Loads environment file dynamically |
| `.env.*` files | Stores configuration for each environment |
| `process.env.ENV` | Controls which env file gets loaded at runtime |
| `package.json` | Passes the environment name via script using `ENV` var |

---