

Advanced Testing in Playwright

Steve Harrison / 6 June 2023

<https://www.steveharrison.dev>



Testing email verification codes

Testing email verification codes

Less ideal options

- Navigating to a web email client in the middle of an automation test, logging in, polling the UI, and extracting the email code when it arrives.
- Building entirely custom infrastructure with your cloud provider to receive an email and then notify the testing client.

testmail.app

PRODUCT PRICING DOCS SIGNUP SIGNIN

Simple email testing

Loved by developers and QA teams

Get **unlimited** email addresses and mailboxes for automating **end-to-end tests** with our simple APIs

- ✓ Test new user signups
Email verifications, passwordless signins, etc.
- ✓ Test transactional and drip emails
Are they triggering when expected?
- ✓ Test delivery and spam scores
Are emails reaching the inbox?



[Get started with our free forever plan](#)
No credit card required



testmail.app

PRODUCT PRICING DOCS SIGNUP SIGNIN

Pricing

Free For side projects	Essential For small projects	Pro Most popular	Enterprise For power users	Unlimited Best of everything
\$0/m always free, forever	\$9/m billed annually or \$12 monthly	\$29/m billed annually or \$39 monthly	\$89/m billed annually or \$129 monthly	\$269/m billed annually or \$299 monthly
100 emails/month	10,000* emails/month	50,000* emails/month	1,000,000* emails/month	10,000,000* emails/month
Unlimited users, email addresses, and mailboxes	All Enterprise features and : Unlimited teams			
1 day retention	1-3 days retention	1-30 days retention	Custom retention	Unlimited custom domains
One random namespace	One random namespace	Unlimited custom namespaces	Unlimited custom namespaces	Custom contracts with enterprise compliance support
	Full API access	Full API access + visual viewer (beta)	Full API access + visual viewer (beta)	
		Priority support	Priority support	
			Enterprise invoicing	
			99.99% uptime SLA	
			Plus <u>add-ons</u>	

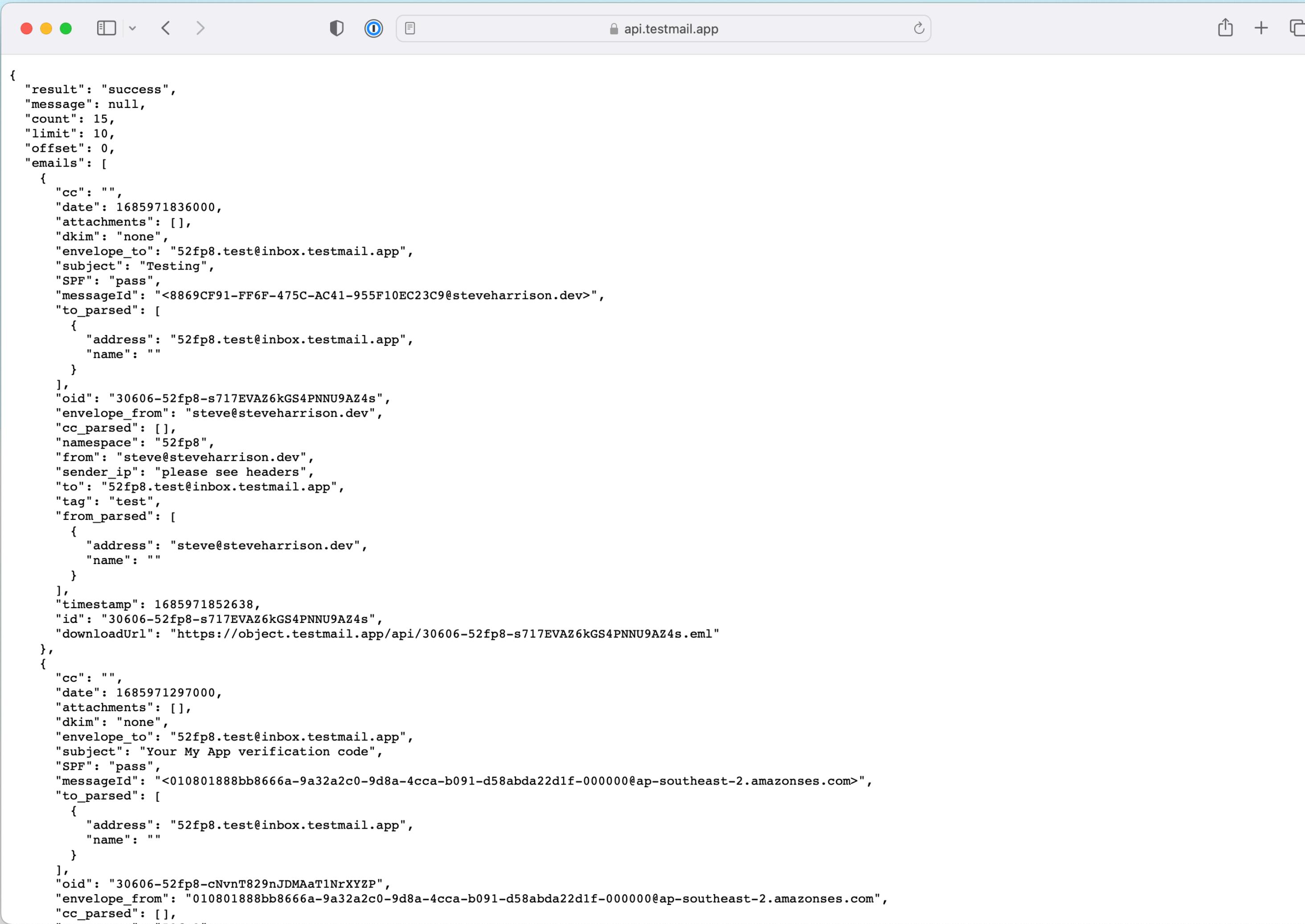
Custom
>10 million emails/m?

GET A QUOTE

Q

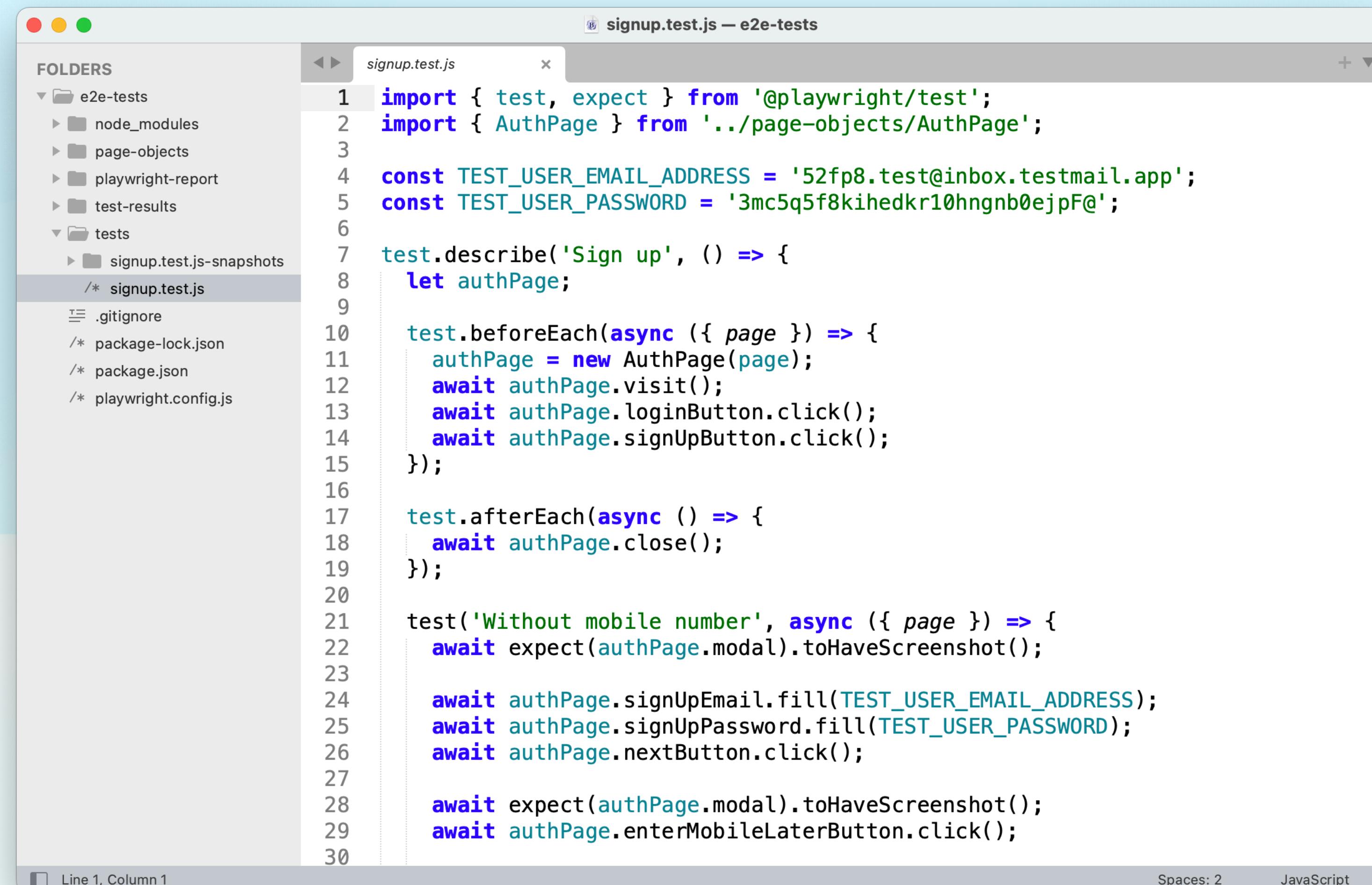
52fp8.test@inbox.testmail.app

<https://api.testmail.app/api/json?apikey=7ecc6f94-1b3f-4f21-9c1b-39558d033f84&namespace=52fp8&pretty=true>



The screenshot shows a web browser window with the URL `api.testmail.app` in the address bar. The page content displays a JSON object representing two email messages. The JSON structure includes fields like `result`, `message`, `count`, `limit`, `offset`, and `emails`. Each `email` object contains detailed information such as `cc`, `date`, `attachments`, `dkim`, `envelope_to`, `subject`, `SPF`, `messageId`, `to_parsed`, `cc_parsed`, `namespace`, `from`, `sender_ip`, `to`, `tag`, `from_parsed`, `timestamp`, `id`, and `downloadUrl`. The JSON is pretty-printed with indentation.

```
{
  "result": "success",
  "message": null,
  "count": 15,
  "limit": 10,
  "offset": 0,
  "emails": [
    {
      "cc": "",
      "date": 1685971836000,
      "attachments": [],
      "dkim": "none",
      "envelope_to": "52fp8.test@inbox.testmail.app",
      "subject": "Testing",
      "SPF": "pass",
      "messageId": "<8869CF91-FF6F-475C-AC41-955F10EC23C9@steveharrison.dev>",
      "to_parsed": [
        {
          "address": "52fp8.test@inbox.testmail.app",
          "name": ""
        }
      ],
      "oid": "30606-52fp8-s717EVAZ6kGS4PNNU9AZ4s",
      "envelope_from": "steve@steveharrison.dev",
      "cc_parsed": [],
      "namespace": "52fp8",
      "from": "steve@steveharrison.dev",
      "sender_ip": "please see headers",
      "to": "52fp8.test@inbox.testmail.app",
      "tag": "test",
      "from_parsed": [
        {
          "address": "steve@steveharrison.dev",
          "name": ""
        }
      ],
      "timestamp": 1685971852638,
      "id": "30606-52fp8-s717EVAZ6kGS4PNNU9AZ4s",
      "downloadUrl": "https://object.testmail.app/api/30606-52fp8-s717EVAZ6kGS4PNNU9AZ4s.eml"
    },
    {
      "cc": "",
      "date": 1685971297000,
      "attachments": [],
      "dkim": "none",
      "envelope_to": "52fp8.test@inbox.testmail.app",
      "subject": "Your My App verification code",
      "SPF": "pass",
      "messageId": "<010801888bb8666a-9a32a2c0-9d8a-4cca-b091-d58abda22d1f-000000@ap-southeast-2.amazonaws.com>",
      "to_parsed": [
        {
          "address": "52fp8.test@inbox.testmail.app",
          "name": ""
        }
      ],
      "oid": "30606-52fp8-cNvnT829nJDMAaT1NrXYZP",
      "envelope_from": "010801888bb8666a-9a32a2c0-9d8a-4cca-b091-d58abda22d1f-000000@ap-southeast-2.amazonaws.com",
      "cc_parsed": []
    }
  ]
}
```



A screenshot of a Mac OS X desktop environment showing a code editor window. The window title is "signup.test.js — e2e-tests". The left sidebar shows a file tree with the following structure:

- e2e-tests
- node_modules
- page-objects
- playwright-report
- test-results
- tests
 - signup.test.js-snapshots
 - /* signup.test.js
 - .gitignore
 - /* package-lock.json
 - /* package.json
 - /* playwright.config.js

The main editor area displays the content of the "signup.test.js" file, which is a Playwright test script. The code uses ES6 syntax and includes imports for `@playwright/test` and `AuthPage`. It defines constants for test user credentials and contains a test suite for sign-up functionality, including before/after each hooks and a specific test for missing mobile number.

```
import { test, expect } from '@playwright/test';
import { AuthPage } from '../page-objects/AuthPage';

const TEST_USER_EMAIL_ADDRESS = '52fp8.test@inbox.testmail.app';
const TEST_USER_PASSWORD = '3mc5qf8kihedkr10hngnb0ejpF@';

test.describe('Sign up', () => {
  let authPage;

  test.beforeEach(async ({ page }) => {
    authPage = new AuthPage(page);
    await authPage.visit();
    await authPage.loginButton.click();
    await authPage.signUpButton.click();
  });

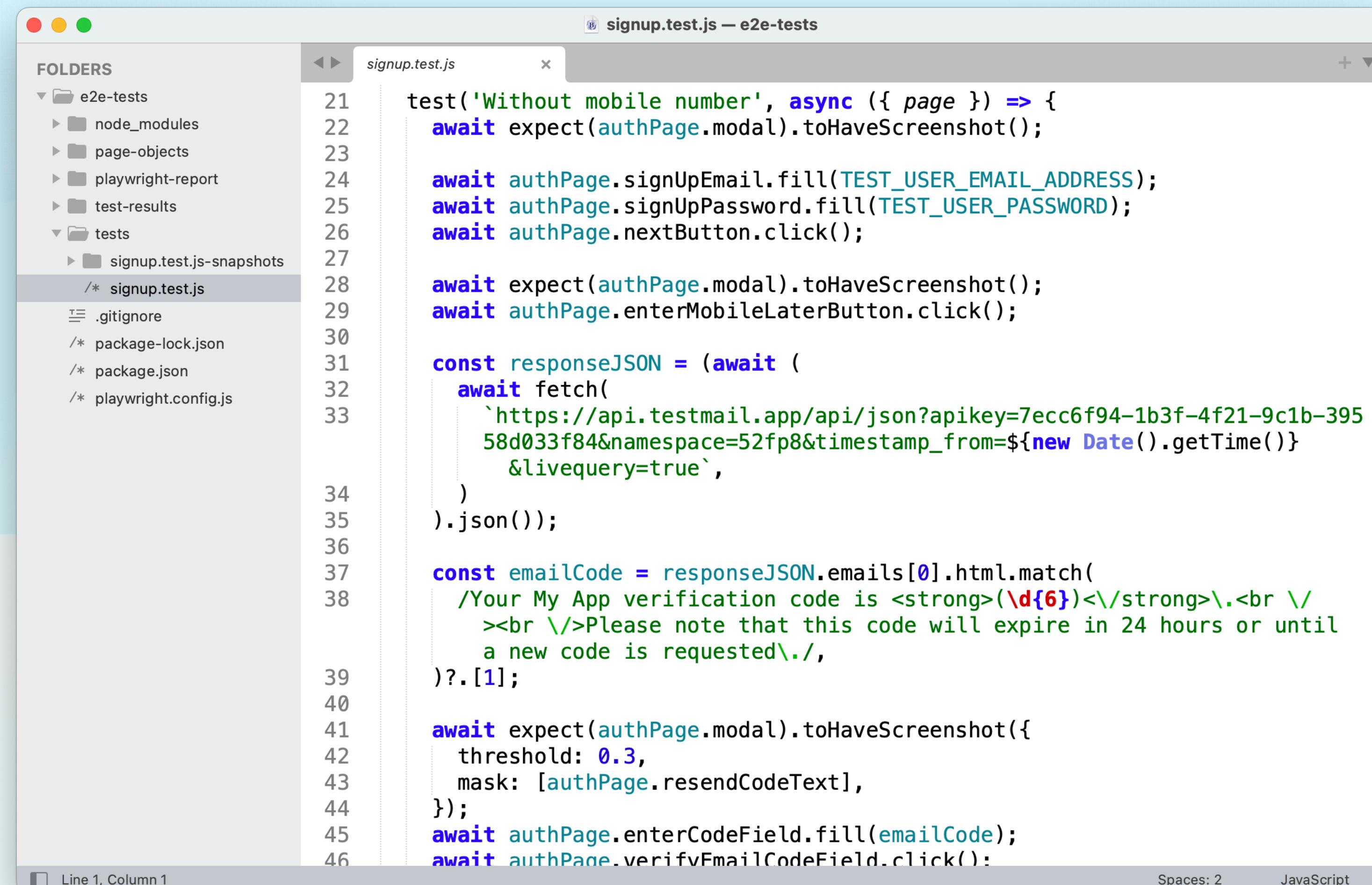
  test.afterEach(async () => {
    await authPage.close();
  });

  test('Without mobile number', async ({ page }) => {
    await expect(authPage.modal).toHaveScreenshot();

    await authPage.signInEmail.fill(TEST_USER_EMAIL_ADDRESS);
    await authPage.signInPassword.fill(TEST_USER_PASSWORD);
    await authPage.nextButton.click();

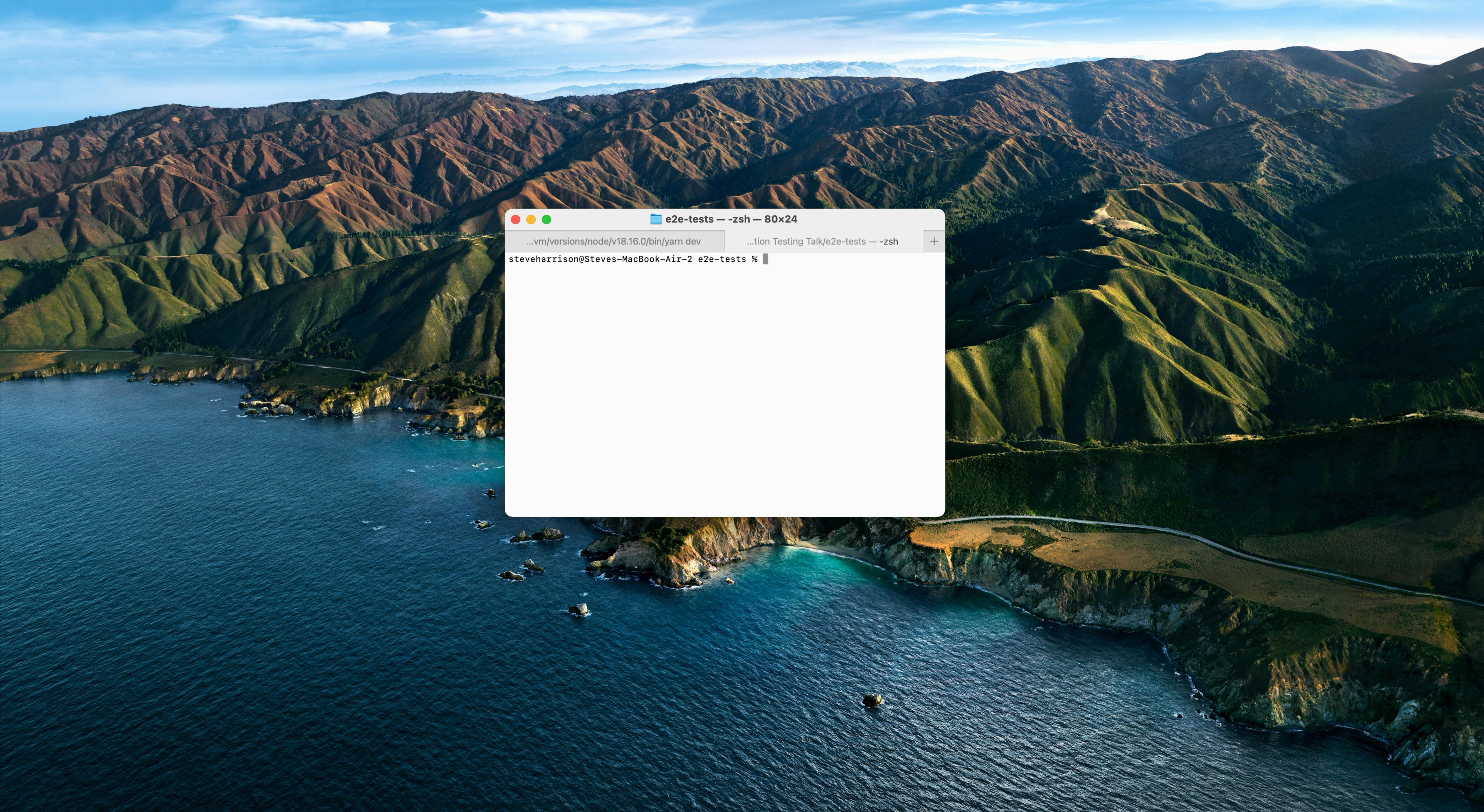
    await expect(authPage.modal).toHaveScreenshot();
    await authPage.enterMobileLaterButton.click();
  });
});
```

At the bottom of the editor, there are status indicators: "Line 1, Column 1", "Spaces: 2", and "JavaScript".

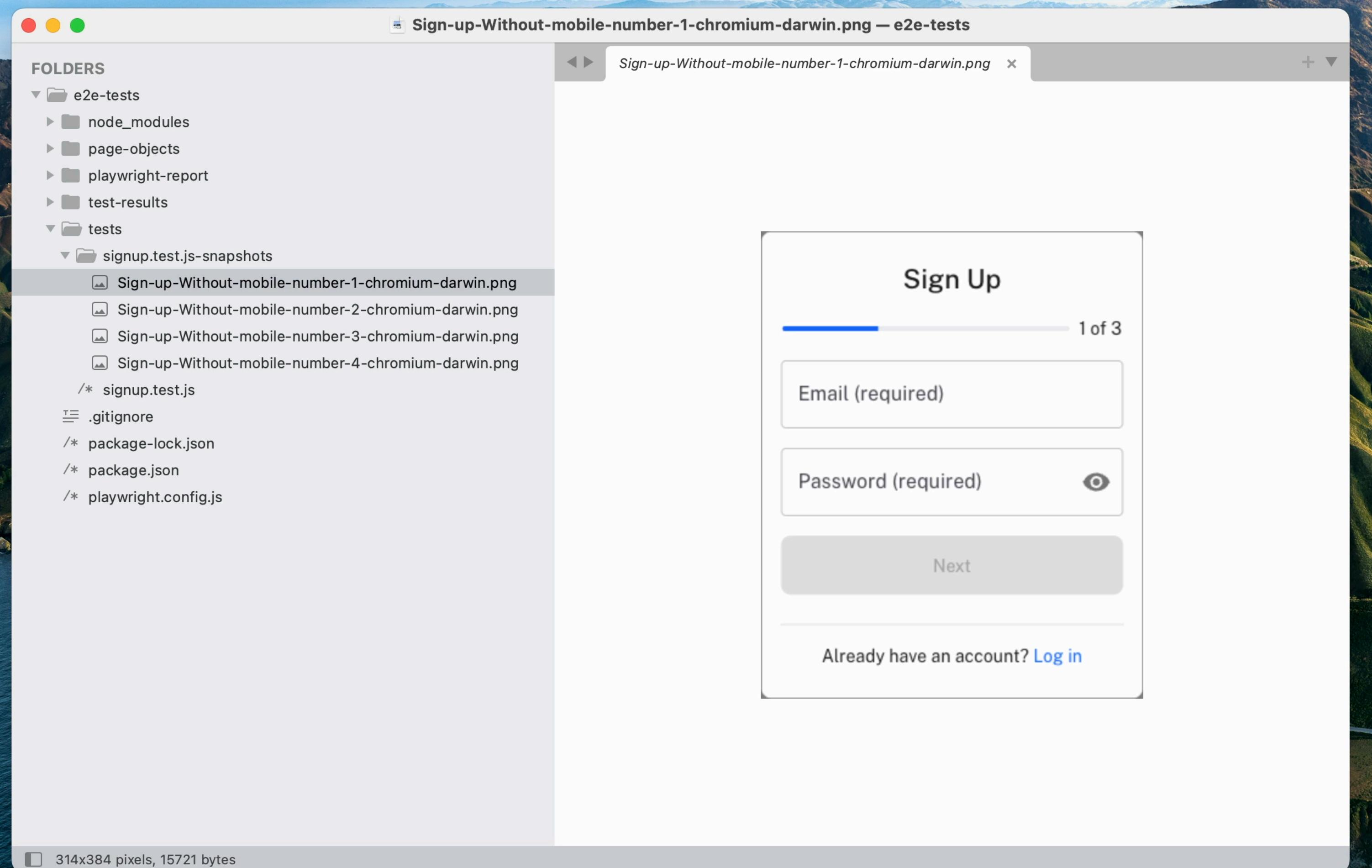


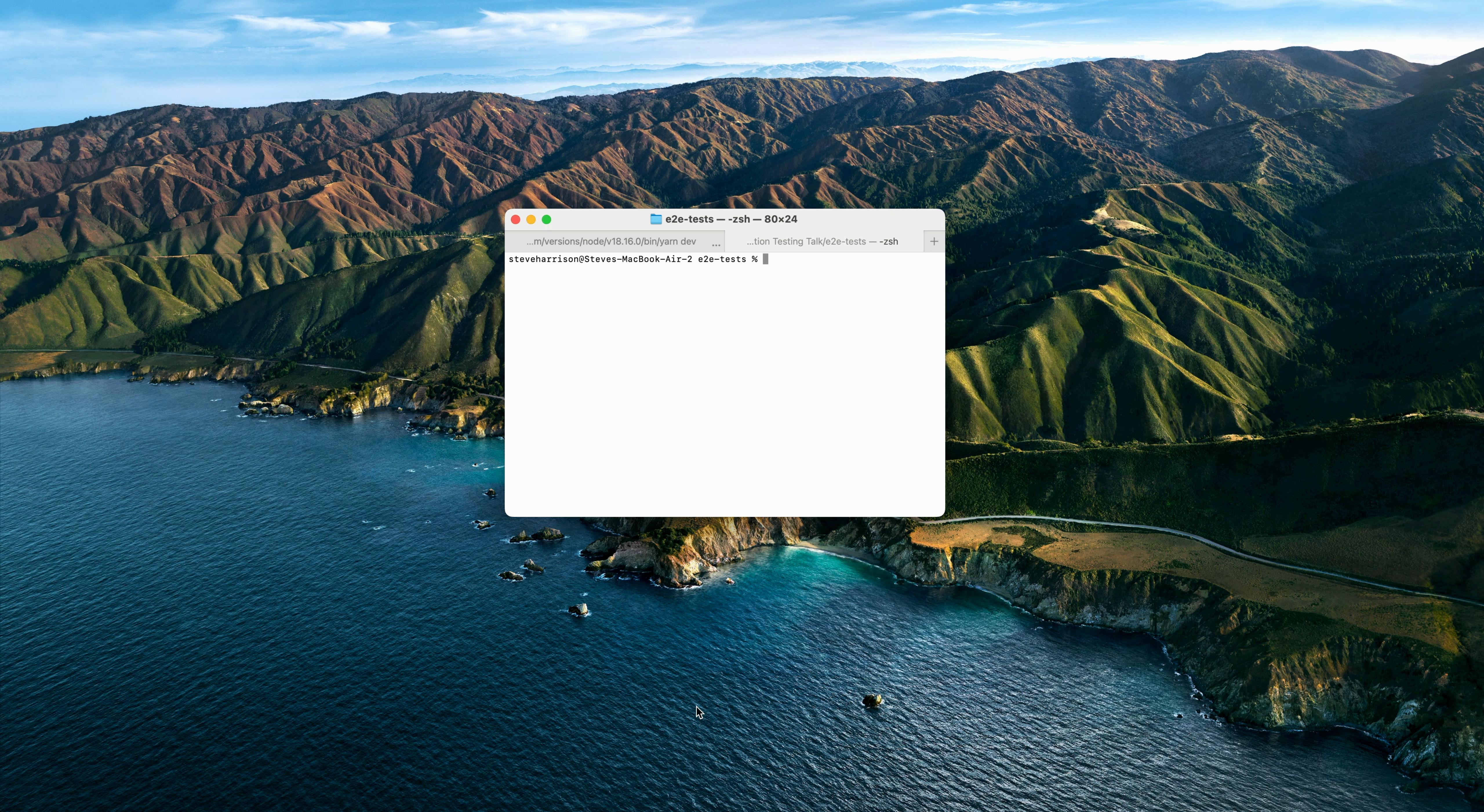
A screenshot of a Mac OS X desktop environment showing a code editor window titled "signup.test.js — e2e-tests". The window has a standard OS X title bar with red, yellow, and green buttons. The main area shows a file named "signup.test.js" with line numbers from 21 to 46. The code is written in JavaScript and uses several await statements to handle asynchronous operations. It includes a fetch call to an API endpoint to get an email verification code, and then waits for a modal to appear with the code. The code editor interface includes a sidebar labeled "FOLDERS" showing project structure, and status bars at the bottom indicating "Line 1, Column 1" and "Spaces: 2 JavaScript".

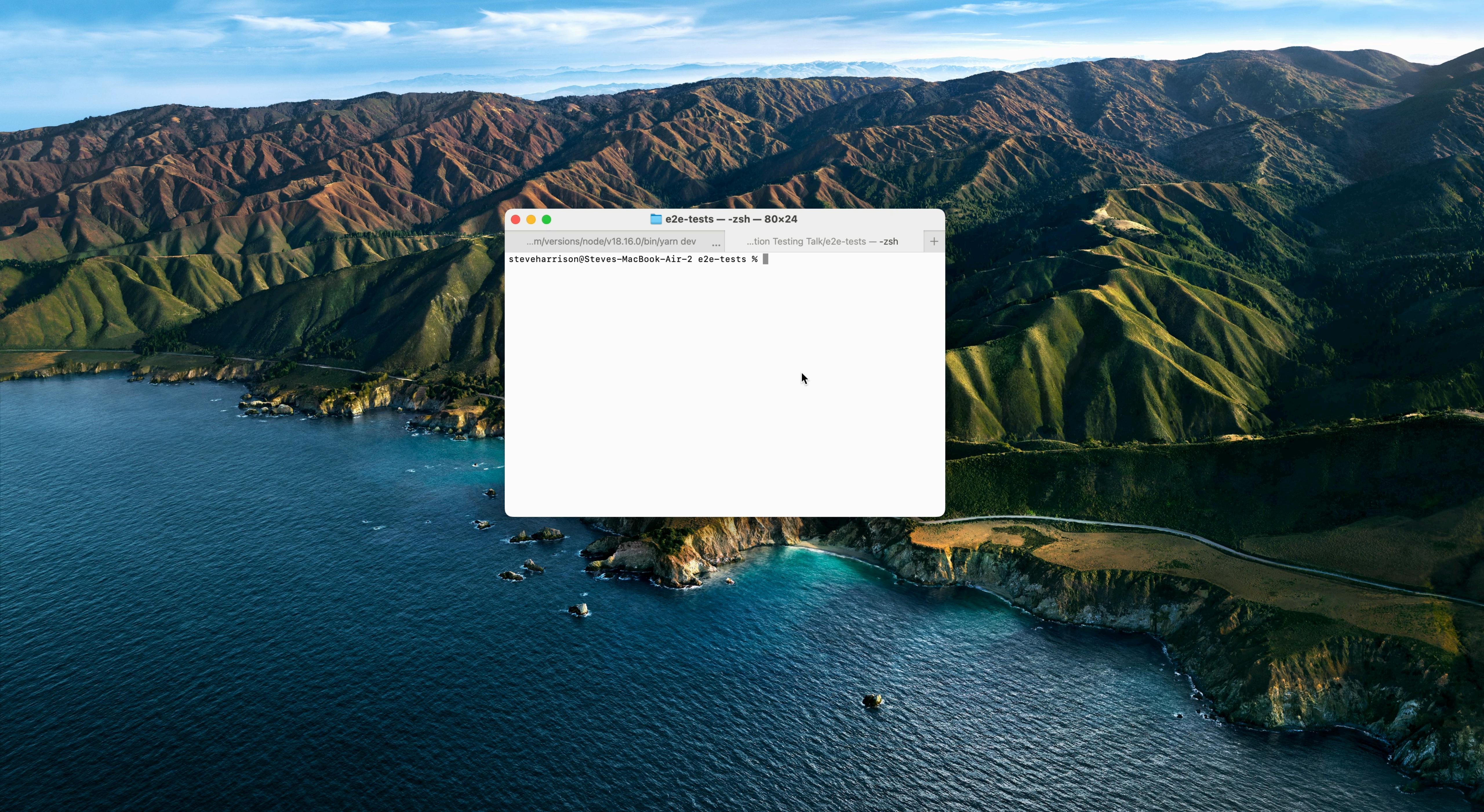
```
21 test('Without mobile number', async ({ page }) => {
22   await expect(authPage.modal).toHaveScreenshot();
23
24   await authPage.signUpEmail.fill(TEST_USER_EMAIL_ADDRESS);
25   await authPage.signUpPassword.fill(TEST_USER_PASSWORD);
26   await authPage.nextButton.click();
27
28   await expect(authPage.modal).toHaveScreenshot();
29   await authPage.enterMobileLaterButton.click();
30
31   const responseJSON = (await (
32     await fetch(
33       `https://api.testmail.app/api/json?apikey=7ecc6f94-1b3f-4f21-9c1b-395
34       58d033f84&namespace=52fp8&timestamp_from=${new Date().getTime()}
35       &livequery=true`,
36     )
37   ).json());
38
39   const emailCode = responseJSON.emails[0].html.match(
40     /Your My App verification code is <strong>(\d{6})</strong>.<br>/
41     ><br>Please note that this code will expire in 24 hours or until
42     a new code is requested./,
43   )?.[1];
44
45   await expect(authPage.modal).toHaveScreenshot({
46     threshold: 0.3,
47     mask: [authPage.resendCodeText],
48   });
49   await authPage.enterCodeField.fill(emailCode);
50   await authPage.verifyEmailCodeField.click();
```



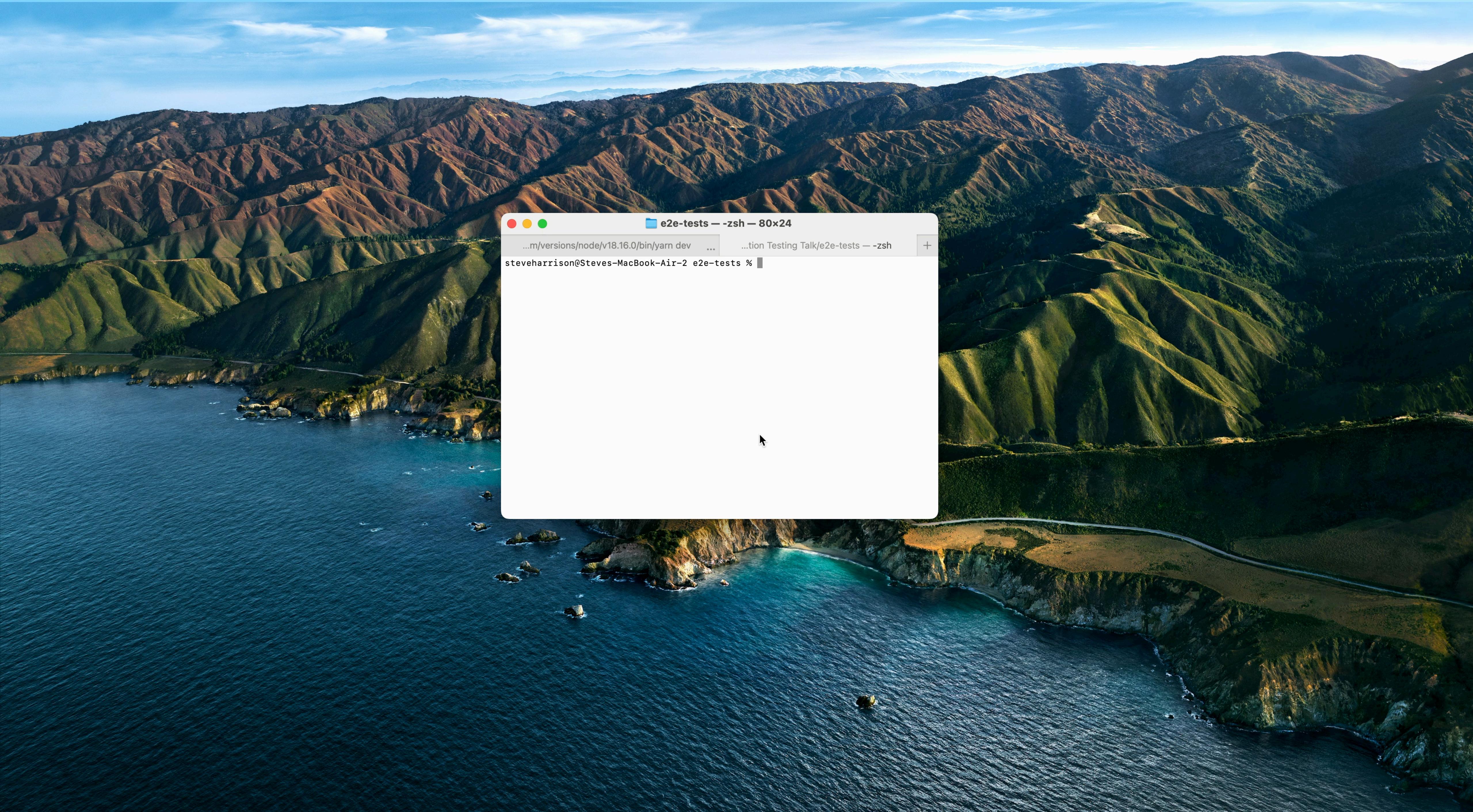
Visual diffing

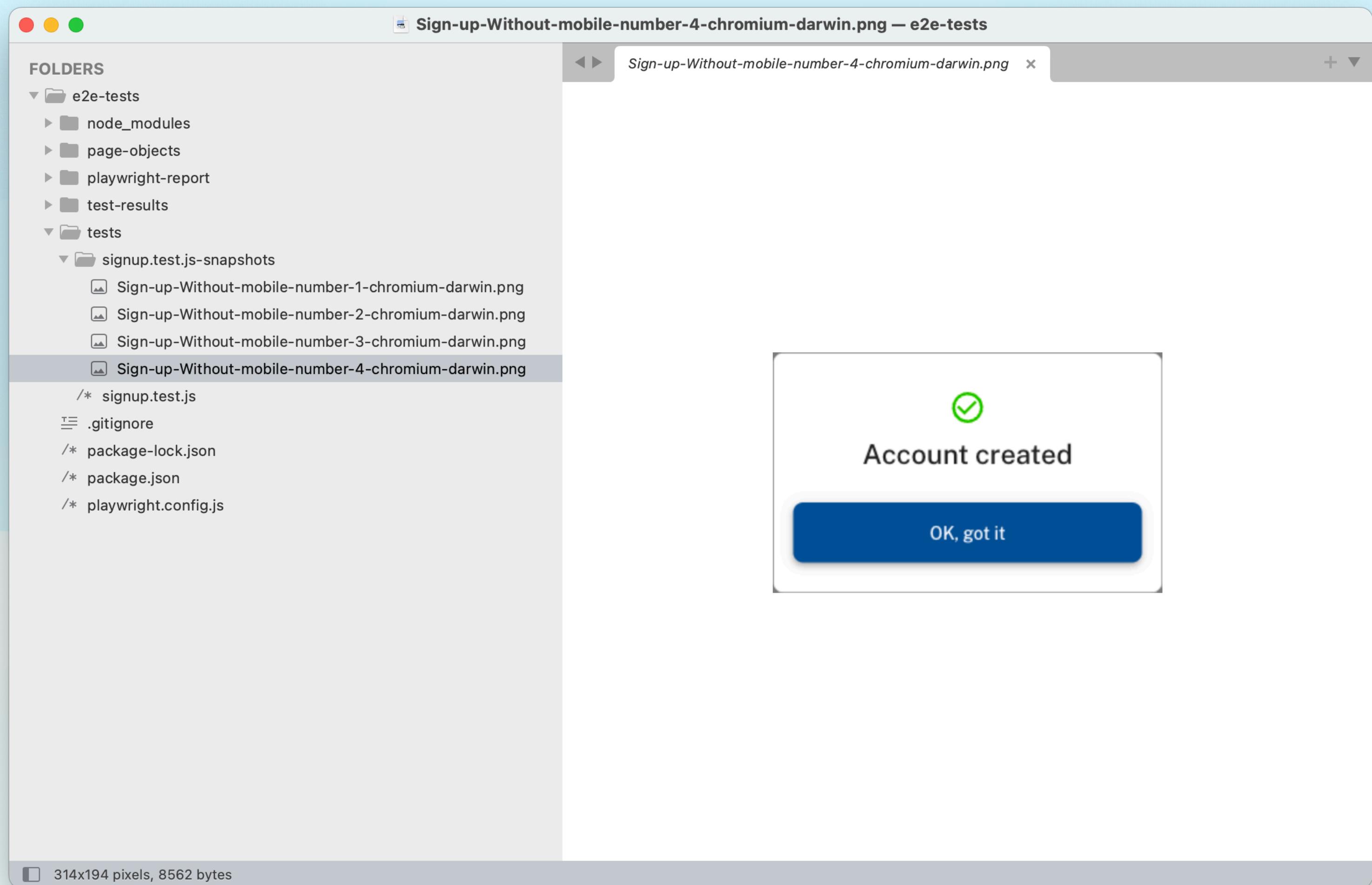






A screenshot of a macOS terminal window titled "e2e-tests — zsh — 80x24". The window has three tabs visible: "...m/versions/node/v18.16.0/bin/yarn dev", "...", and "...tion Testing Talk/e2e-tests — zsh". The main pane shows the command "steveharrison@Steves-MacBook-Air-2 e2e-tests %". A mouse cursor is visible in the center of the terminal window.





Testing SMS verification codes

The screenshot shows the MailSlurp website homepage. At the top, there's a navigation bar with links for Product, Guides, Documentation, Pricing, Support, a search bar, and sign-in/sign-up buttons. Below the navigation is a large banner with the headline "Test, build, automate" and a subtext about powerful Email and SMS API platforms. It includes two buttons: "Start free account" and "Documentation". To the right of the banner is a screenshot of the MailSlurp dashboard, which features a sidebar with options like Compose, Inboxes, Emails, TXT SMS, Webhooks, Verification, Bounces, Upgrade, and Settings. The main area of the dashboard shows usage statistics: Monthly new inboxes (401/500), Monthly sends (158/500), Monthly receives (500/500), Validation (67/∞), and other metrics like Custom domains (1/1), Team members (0/50), Daily bounces (0/10), and Max bounces (0/40). Below the dashboard is a section titled "Developer integrations" with a sub-section "Create and control email accounts and phone numbers in any language or framework". It includes tabs for C#, Go, Java, Javascript (which is selected), PHP, Python, and Ruby, and a code snippet for creating an email address using the MailSlurp client library.

MailSlurp

Product ▾ Guides ▾ Documentation ▾ Pricing Support ▾

Search

Sign in → Sign up →

Test, build, automate

Powerful Email and SMS API platforms. Routing, webhooks, mail catchers and forwarding. For developers, QA, and marketing.

Start free account → Documentation →

C# Go Java **Javascript** PHP Python Ruby

```
const MailSlurp = require('mailslurp-client').default;
const mailslurp = new MailSlurp({ apiKey: process.env.API_KEY!! });
// create an email address
```

Developer integrations

Create and control email accounts and phone numbers in any language or framework.

MailSlurp > dashboard

API KEY: cffea3bf010a3557d5c2b78476d53192317c8fd98726187edcd2ec53cd5d5

```
// npm i -s mailslurp-client
// import { MailSlurp } from 'mailslurp-client';
const mailslurp = new MailSlurp({ apiKey: "cffea3bf010a3557d5c2b78476d53192317c8fd98726187edcd2ec53cd5d5" });
const inbox = await mailslurp.inboxController.createInboxWithDefaults();
```

Usage

Monthly new inboxes: 401 / 500	Monthly sends: 158 / 500	Monthly receives: 500 / 500	Validation: 67 / ∞
Custom domains: 1 / 1	Team members: 0 / 50	Daily bounces: 0 / 10	Max bounces: 0 / 40

Help guides

- Quick start guide
- Integration testing
- Documentation
- Features
- Integrations

Legal Pricing Documentation Support

The screenshot shows a web browser window for the MailSlurp website (mailslurp.com). The page displays a guide titled "Wait for and capture SMS".

Sidebar Navigation:

- What is OTP and why would I use it?
- Testing MFA and OTP
- Playwright setup
 - Create a test project
 - Install playwright
 - Running tests
- Testing user sign up
 - Load the create account page
 - Use a test phone number
 - Fill sign-up form
 - Wait for and capture SMS
 - Confirm user SMS OTP
 - Sign in and view user data
- Conclusion

Tags: sms otp testing

Main Content:

Wait for and capture SMS

Now we can use MailSlurp to wait for an expected SMS to arrive at our phone number and extract the verification code:

```
await page.click('[data-test="sign-up-create-account-button"]');
// wait for verification code
const sms = await mailslurp.waitController.waitForLatestSms({
  waitForSingleSmsOptions: {
    phoneNumberId: phone.id,
    unreadOnly: true,
    timeout: 30_000,
  }
})
// extract the confirmation code (so we can confirm the user)
const code = /([0-9]{6})$/.exec(sms.body)?.[1] !!;
```

Confirm user SMS OTP

Use the code we extracted to fill the username field:

```
// enter confirmation code
await page.fill('[data-test="confirm-sign-up-confirmation-code-input"]', code);
```

Pricing

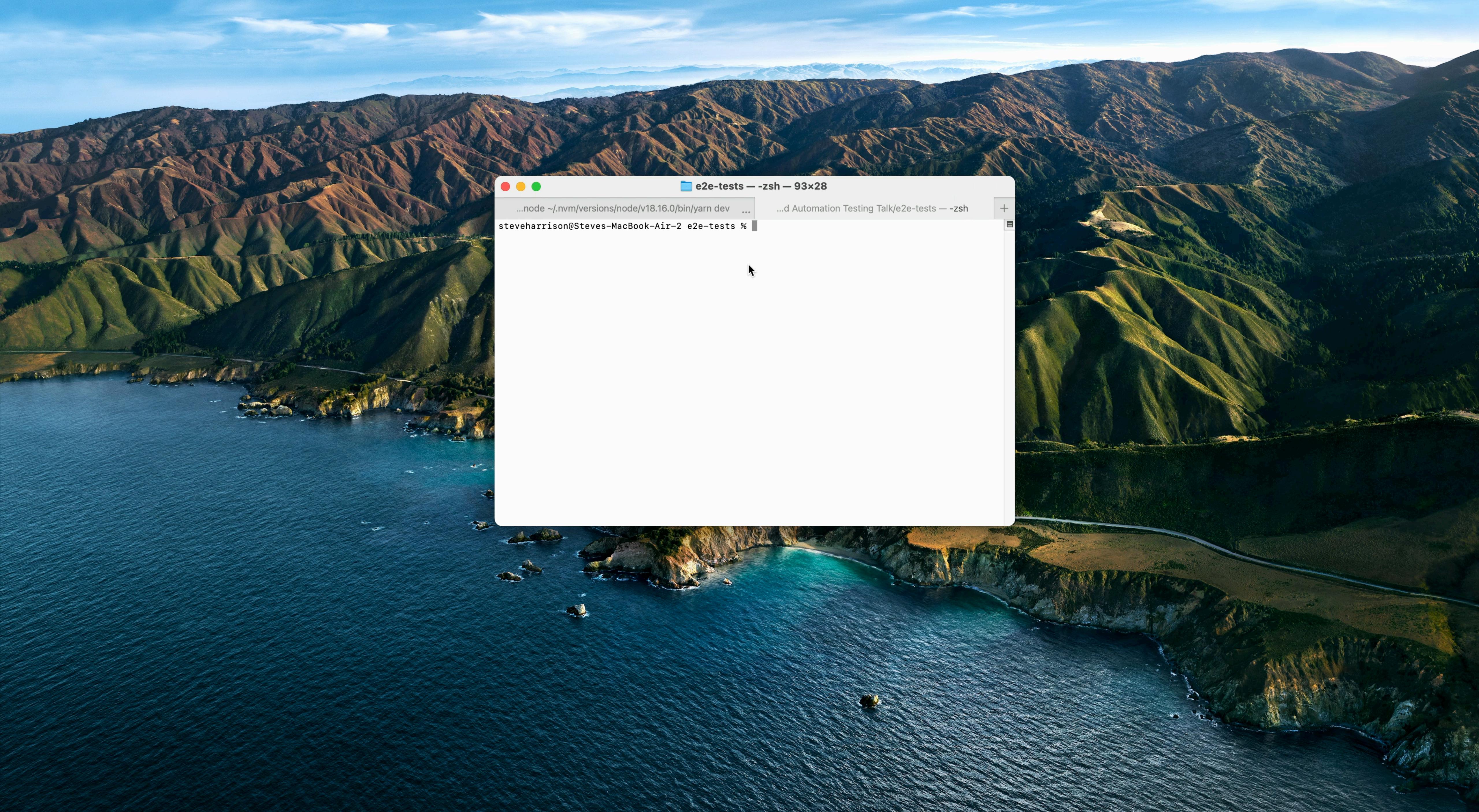
Provider	MailSlurp	Twilio	AWS
Plan	US\$20/month	–	–
Dedicated Australian phone number	AU\$8/month	US\$6.5/month	–
SMS	US\$0.10/SMS	US\$0.0515/month	–

signup.test.js — e2e-tests

```
FOLDERS
  e2e-tests
    node_modules
    page-objects
      AuthPage.js
    playwright-report
    test-results
    tests
      signup.test.js-snapshots
        signup.test.js
      .gitignore
      package-lock.json
      package.json
      playwright.config.js
      test.mjs

signup.test.js
  81  // Your My App verification code is <strong>(\d{6})</strong>.  
  82  // Please note  
  83  // that this code will expire in 24 hours or until a new code is requested./,  
  84  )?.[1];  
  85  await authPage.enterCodeField.fill(emailCode);  
  86  await authPage.verifyEmailCodeButton.click();  
  87  await expect(page.getText('0483 907 225')).toBeVisible();  
  88  
  89  const sms = await mailslurp.waitForLatestSms({  
  90    waitForSingleSmsOptions: {  
  91      phoneNumberId: '6c4af933-53ff-4dd5-ab68-ea24f2fefc45',  
  92      timeout: 30_000,  
  93      unreadOnly: true  
  94    }  
  95  });  
  96  
  97  const mobileCode = sms.body.match(/Your My App verification code is (\d{6})\.\n\nPlease note  
  98  // that this code will expire in 24 hours or until a new code is requested./)?.[1];  
  99  
100  await authPage.enterMobileCodeField.fill(mobileCode);  
101  await authPage.verifyMobileCodeButton.click();  
102  
103  await expect(page.getText('Account created')).toBeVisible();  
104  await authPage.okGotItButton.click();  
105  
106  await authPage.openMenuButton.click();  
107  await expect(authPage.myAccountMenuItem).toBeVisible();  
108  
109  // Clean up after ourselves  
110  await authPage.myAccountMenuItem.click();  
111  await page.getText('Delete account').click();  
112  await authPage.modalDeleteButton.click();  
113  await expect(page.getText('Your account has been deleted.')).toBeVisible();  
114 };  
115 }
```

Line 54, Column 39 Spaces: 2 JavaScript



Thank you



<https://www.steveharrison.dev>