Meet Scoutr

Developer Documentation

Content

Scoutr

Tech Stack

Tech Stack

The technologies powering Scoutr. From the ones closer to the recruiters to the ones behind to the production server.

🎨 CSS	
⚛️ React	
🟦 TypeScript	
🪐 Cosmos	FRONTEND
⬢ Node.js	
Ⓧ Express	
🎭 Playwright (scrap)	
🍊 Cheerio	
🐘 PostgreSQL	BACKEND
⚡ Vitest	
🐳 Docker	
🐙 GitHub Actions	
🗼 Lighthouse*	
🎭 Playwright (E2E)*	
🔦 Sentry*	DevOps

* Planned for Autumn '24

CSS
Custom, handwritten CSS with no frameworks for the smallest package size.

React
To create the user interface (UI) uses dynamic imports and Suspense to match Next.js page load speed.

TypeScript
To have a strong typed, unified language in the frontend and backend.

Cosmos
To iterate and test UI components faster. It replaced Storybook to reduce the developer dependencies.

Frontend

Node.js
Runtime environment for executing JavaScript on the backend.

Express
Framework used for creating REST API's and the Server Side Events. (SSE)*

Playwright (scrapping)**
Tool used to create an "invisible" virtual browser for navigating and downloading LinkedIn profiles.

Cheerio
Tool used to obtain specific data from LinkedIn profiles.

Postgres
Database used for storing assignments and candidates.

Backend

* SSE Is a technique similar to WebSockets that allows the server to stream data contantly to the frontend. See the ETL diagram for more information about how we use it.
** Scrapping is the technique of using virtual browsers to navigate websites that do not have an API endpoint and extract its data.

Scoutr

Vitest
Test suite used for Test Driven Development (TDD) in the frontend and backend.

GitHub Actions
Platform to do Continuous Integration including running Vitest.

Docker
Containerization platform used for setting up the development environment and orchestrating the backend with the database.

DevOps

App Elements

Assignments
Novare refers to a recruitment process as an *assignment*. Each assignment includes multiple individuals referred to as *candidates*.

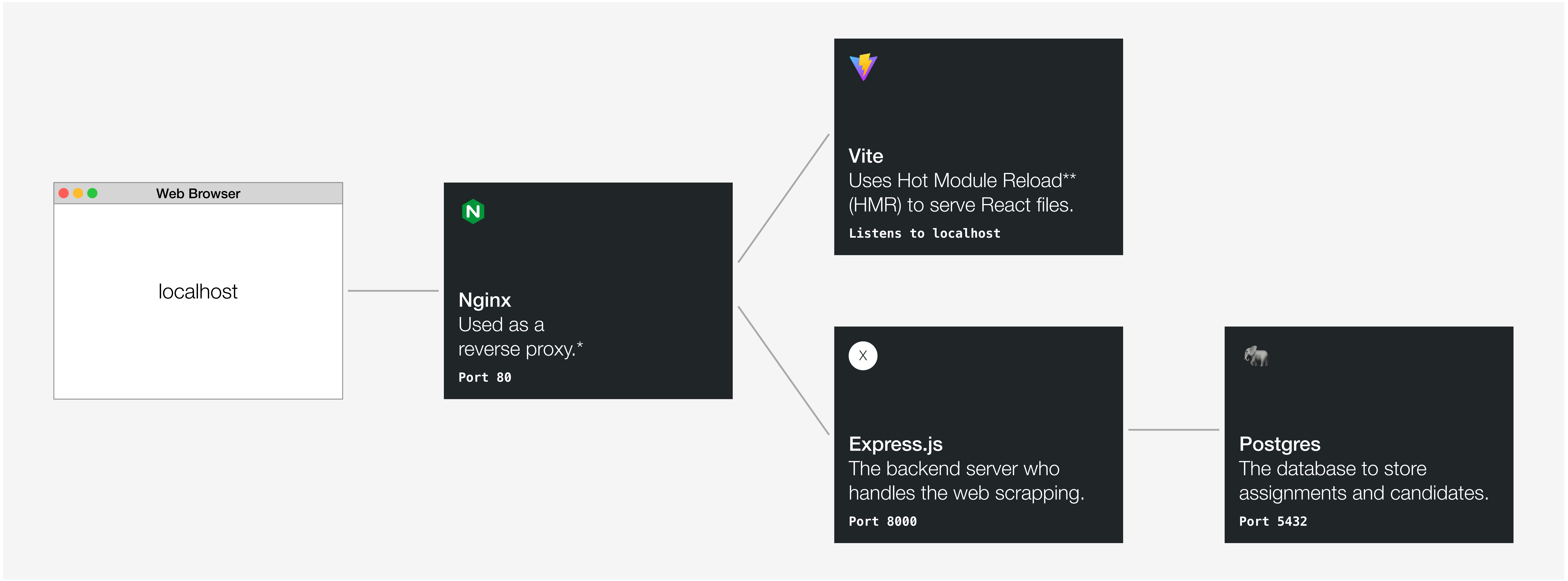
LinkedIn Profile
The LinkedIn user webpage contains work information such as current job, education, social feed, and more.

Candidates
The extracted LinkedIn data relevant for an assignment, along with additional data provided by the recruiter.

Relation between Assignments, LinkedIn, and Candidates

Project Architecture

Scoutr

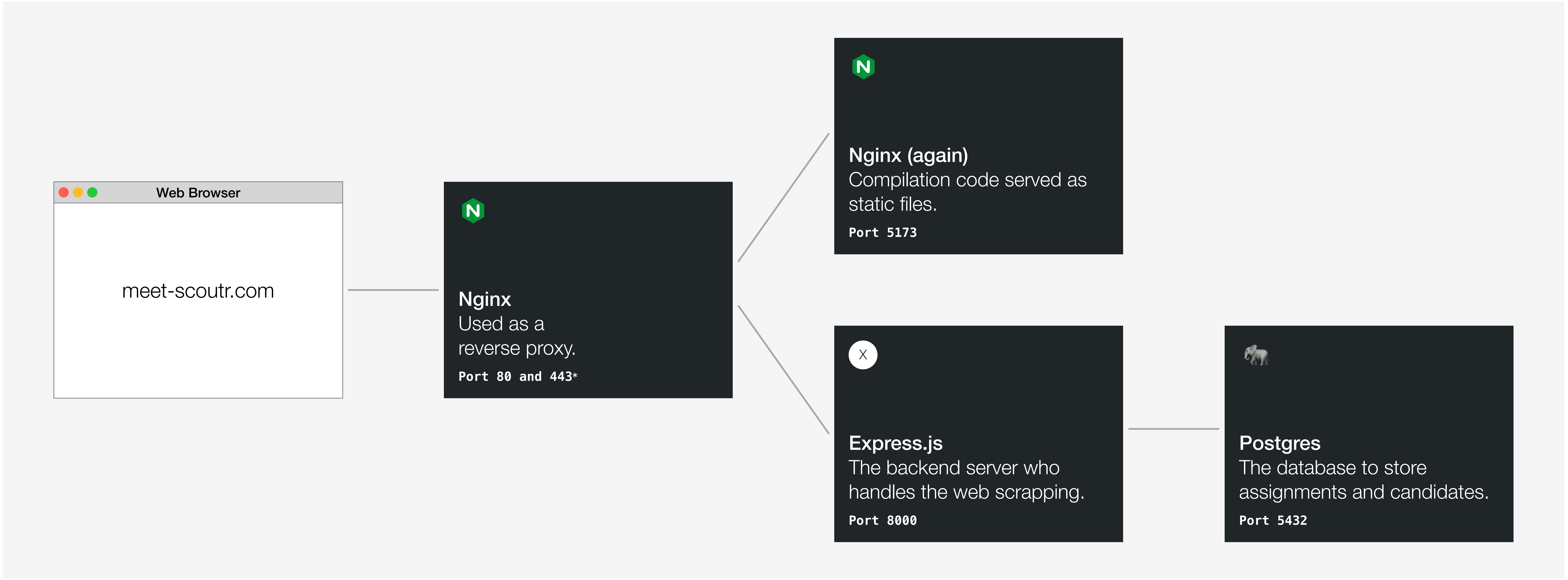


Web Browser
localhost
Nginx
Used as a reverse proxy.*
Port 80
Vite
Uses Hot Module Reload** (HMR) to serve React files.
Listens to localhost
X
Express.js
The backend server who handles the web scrapping.
Port 8000
Postgres
The database to store assignments and candidates.
Port 5432

Development Environment

* The diagram is inspired by the learnings of *Dockerizing a React application with Nodejs Postgres and NginX* (video source)
** HMR allows Vite, the frontend server to sent a signal to reload the browser every time the code changes without losing the current data. The signal is sent using Web Sockets.

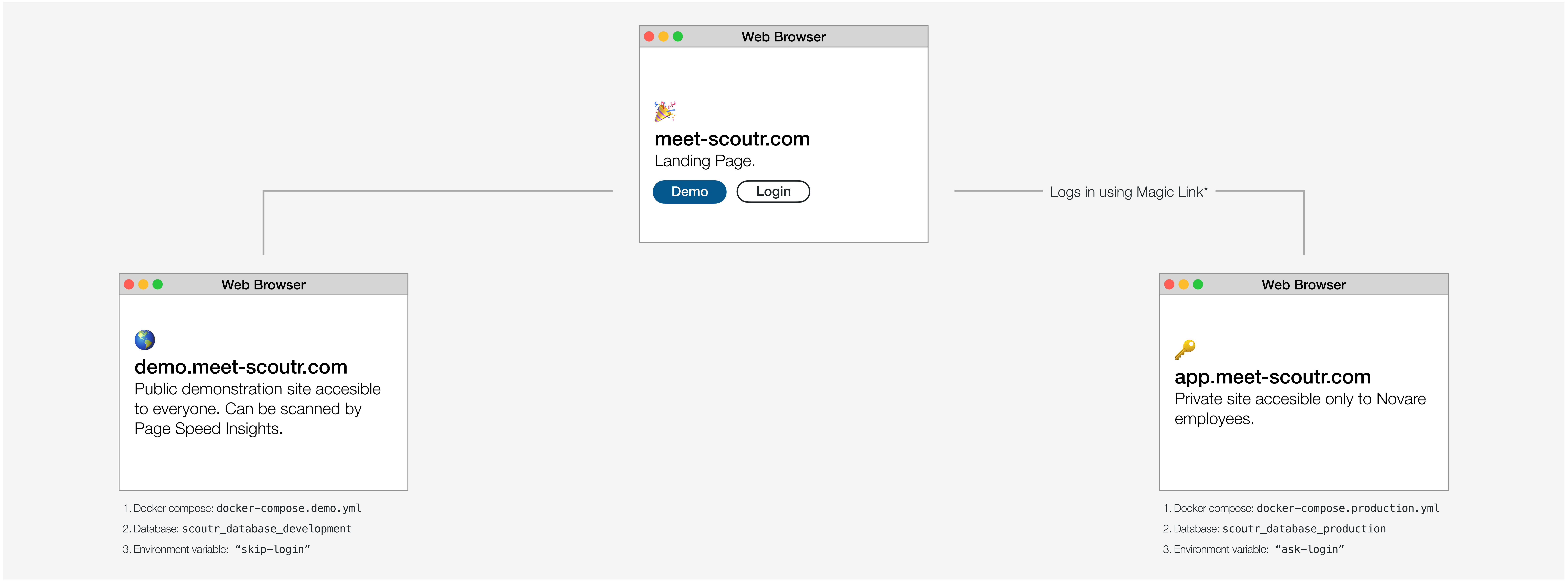
Scoutr


Web Browser
meet-scoutr.com
Nginx
Used as a reverse proxy.
Port 80 and 443*
Nginx (again)
Compilation code served as static files.
Port 5173
Express.js
The backend server who handles the web scrapping.
Port 8000
Postgres
The database to store assignments and candidates.
Port 5432

Production Environment

* Once the project is on the cloud, we utilize Amazon's Elastic Load Balancer (ELB) to redirect the port 80 to the port 443 to use https (secure) with a SSL certificate.

Scoutr



meet-scoutr.com
Landing Page.

demo.meet-scoutr.com
Public demonstration site accesible to everyone. Can be scanned by Page Speed Insights.

1. Docker compose: `docker-compose.demo.yml`
2. Database: `scoutr_database_development`
3. Environment variable: `“skip-login”`

app.meet-scoutr.com
Private site accesible only to Novare employees.

1. Docker compose: `docker-compose.production.yml`
2. Database: `scoutr_database_production`
3. Environment variable: `“ask-login”`

Landing Page Proposal

* Magic Link is a password-less login system where users enter their email and receive a code in their inbox, which they then use to log in on the website. (documentation)

Scoutr

Database Schema

Scoutr

Assignments
Stores information about each assignment as a separate project.

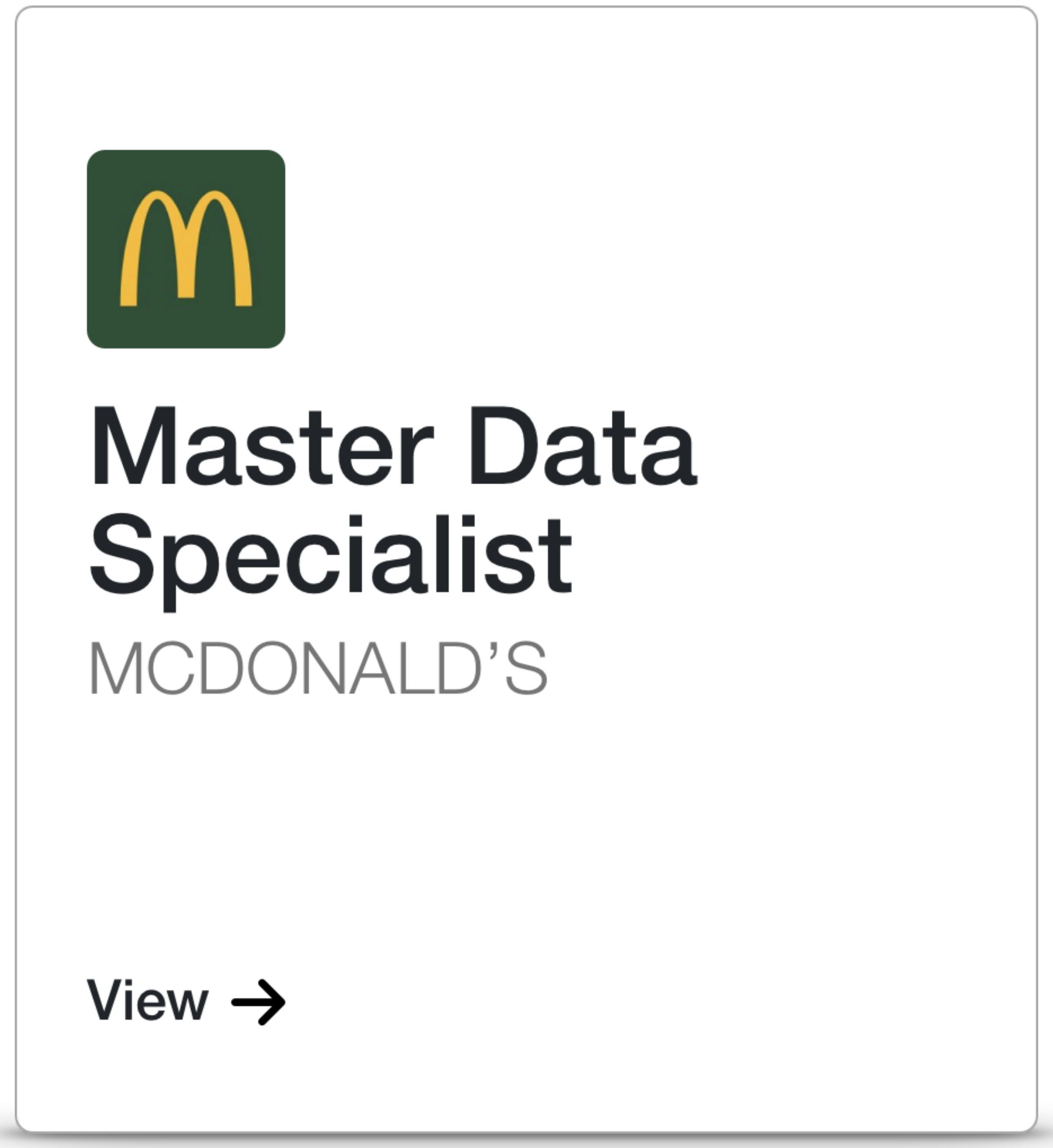
Candidates
Stores details of all candidates, with the foreign key `assignment_id` linking each candidate to its corresponding assignment.

Report logs
Records encountered errors during a candidate web scrapping parsing process.

Database Schema

💼 Assignments

Key	Type	Description
Id	`number`	Unique identifier for each assignment.
date_created	`date`	Date when the assignment was created.
assignment_name	`string`	The assignment to fulfill.
company_name	`string`	The company hiring for the assignment.
company_image_url	`string`	The URL of the company's logo.



Scoutr

👨🏻 Candidates

Key	Type	Description
Id	`number`	Unique identifier for each candidate.
assignment_id 🔑	`number`	Foreign key linking candidates to assignments.
date_created	`date`	Date when the candidate profile was added to the database.
linked_in_url	`string`	URL of the candidate's LinkedIn profile.
candidate_name	`string`	Full name of the candidate.
candidate_job_title	`string`	Job title of the candidate.
candidate_image_url	`string`	URL of the candidate's profile picture.
company_name	`string`	The name of the company the candidate is currently working.
company_duration_in_months	`number`	Duration the candidate has been working at the current company, in months.
company_image_url	`string`	URL of the company's logo.
notes	`string`	Any additional notes about the candidate written by the recruiter.
relevance	`number`	Rating from 1 to 5 indicating the candidate's relevance to the assignment, with higher values being more relevant.
contact_status	`number`	Rating from 1 to 6 indicating the status of contact with the candidate, with lower values indicating more recent contact.
contact_date	`date`	Date of the most recent contact with the candidate.

PROFILE #1

Candidate:	James Cameron DATA ANALYST
Company:	Folksam 1Y 3M
Revelance:	3 Yes
Contact:	2 Scheduled JAN-31
Notes:	Just transitioned to data engineering, before that was a sofware developer (which makes sense) but is too junior

Scoutr

📈 Report Logs

Key	Type	Description
Id	`number`	Unique identifier for each report log entry.
date_created	`date`	Date and time when the report log was generated.
linked_in_url	`string`	URL of the LinkedIn profile that triggered the error, facilitating later analysis.
severity	`number`	Severity level indicating the impact of the error: 1 Missing fields, 2 All fields missing but no crash, 3 System crash.
message	`string`	Detailed error message provided by the subsystem that encountered the error.

*

* There is no visual representation of the Report log, it is stored on the database for later analysis.

Scoutr

API Routes

Folder	Method	Route	Description
💼 Assignments	`GET`	`/api/assignments`	Get all the assignments.
💼 Assignments	`POST`	`/api/assignments`	Creates a new assignment and return its ID. Check the database schema of Assignments to know what values to pass
👨🏻 Candidates	`GET`	`/api/candidates/:assignment_id`	Get all candidates belonging to an assignment.
👨🏻 Candidates	`PATCH`	`/api/candidates/:id`	Update a candidate by id.
🌎 Web Scrapper	`GET`	`/sse/parse-links/:assignment_id`	Parses one or more LinkedIn links, stores them in the candidates table with their `assignment_id`. Finally, it returns these new candidates. Uses the prefix `/sse` instead of `/api`

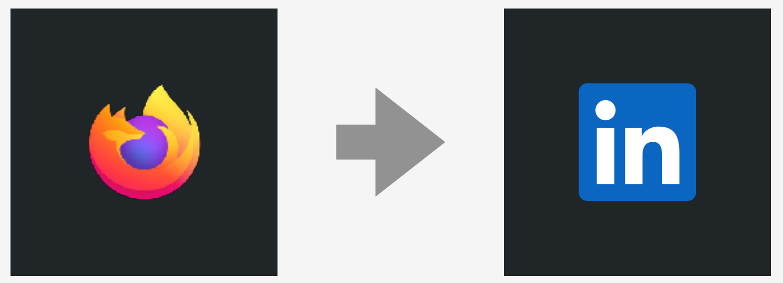
API Routes

Web Scrapper

Web Scrapper

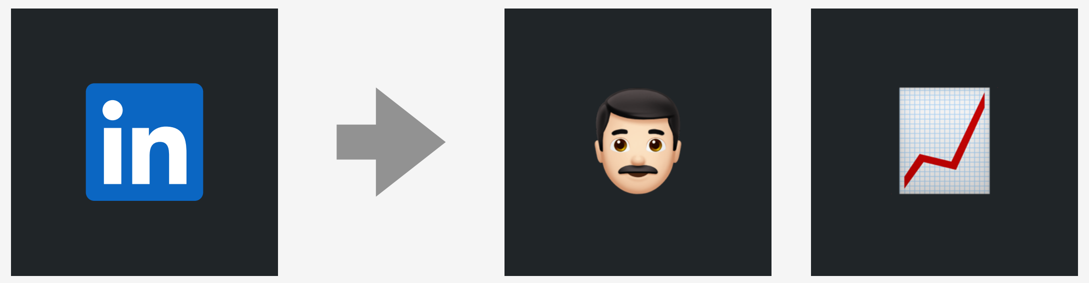
LinkedIn don't provide an API for profiles. Therefore, Scoutr uses a web scrapping model inspired by Extract, Transform, Load (ETL) framework used in data engineering.

ETL Diagram



Extract

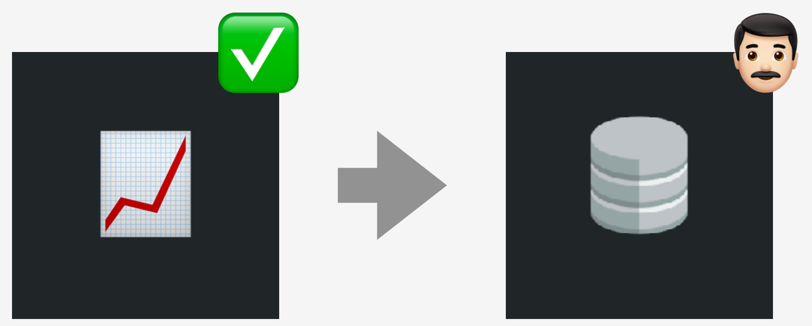
Playwright boots up a Firefox page with previously stored login credentials to navigate and then downloads the requested LinkedIn profile.



Transform

Obtains and converts the data that is relevant to us.

It also generates a report with information we could not obtain, for example, a profile picture due to privacy settings.



Load (Store)

The report is reviewed. If the required data is valid, it's stored in the database as a candidate. Else we notify the user that we could not scan the profile.

Scoutr

Code Documentation

browser
Has coding files to boot up a web browser with an always enabled authentication credentials.

database
Has coding files for connecting the backend to the database and initializing the tables.

queries*
Has plain text files with commands to interact with the database.

routes*
Has coding files to connect the backend to the frontend using the REST API protocol.

scan-profile
Has coding files to extract the LinkedIn profiles, transform them and storing them into the database.

types
Has TypeScript interfaces to enforce strong data typing and to self document the data.

Folder Structure

* The queries and routes folders only contain standard SQL queries and REST code, so they're not documented here.

Scoutr

Browser

Has coding files to boot up a web browser with an always enabled authentication credentials.

Get Auth

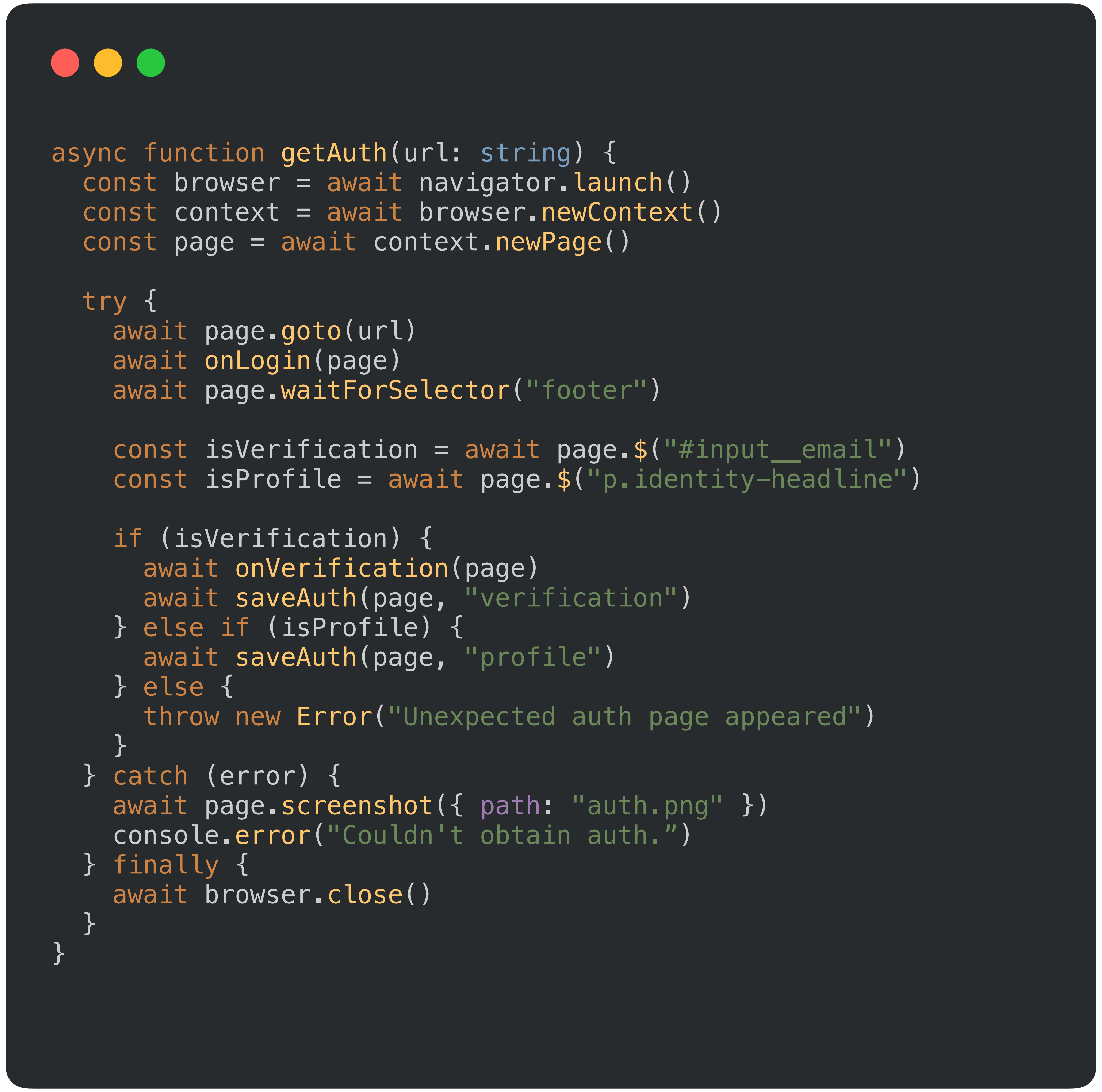
Runs a terminal program to create a LinkedIn credential file by asking for a LinkedIn email and password. The Scoutr product backlog has sample accounts for development and production.

On first use from a new IP address, LinkedIn sends a verification code to the provided email, this is skipped on subsequent logins.

This function does not pass data to other functions; instead, it saves the `LoginAuth.json` file to the root of the backend folder.

Input

Name	Type	Description	Example
url	`string`	The URL of LinkedIn's login page	linkedin.com/login



```
async function getAuth(url: string) {
  const browser = await navigator.launch()
  const context = await browser.newContext()
  const page = await context.newPage()

  try {
    await page.goto(url)
    await onLogin(page)
    await page.waitForSelector("footer")

    const isVerification = await page.$("#input__email")
    const isProfile = await page.$("p.identity-headline")

    if (isVerification) {
      await onVerification(page)
      await saveAuth(page, "verification")
    } else if (isProfile) {
      await saveAuth(page, "profile")
    } else {
      throw new Error("Unexpected auth page appeared")
    }
  } catch (error) {
    await page.screenshot({ path: "auth.png" })
    console.error("Couldn't obtain auth.”)
  } finally {
    await browser.close()
  }
}
```

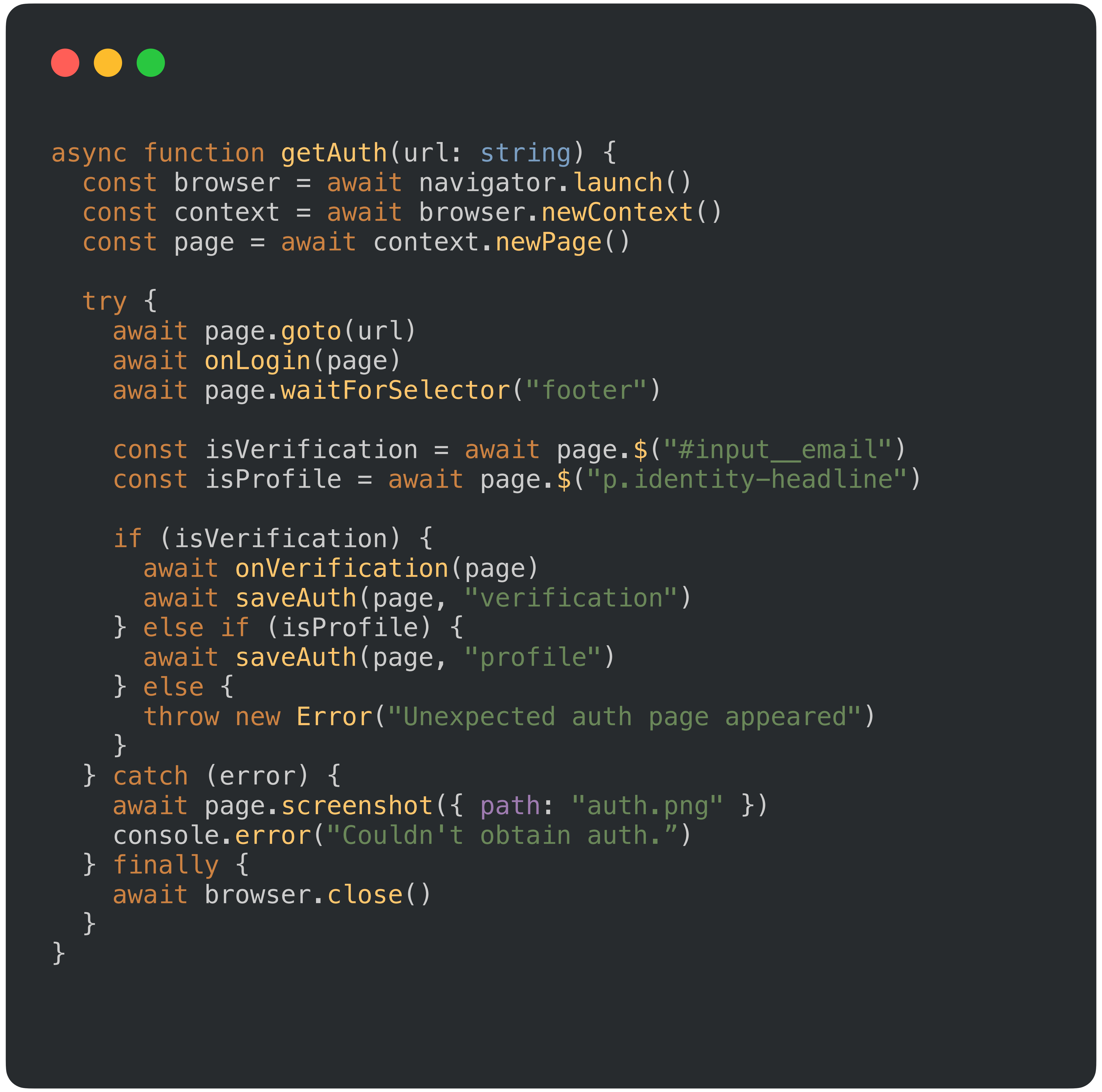
Get Auth

Runs a terminal program to create a LinkedIn credential file by asking for a LinkedIn email and password. The Scoutr product backlog has sample accounts for development and production.

On first use from a new IP address, LinkedIn sends a verification code to the provided email, this is skipped on subsequent logins.

This function does not pass data to other functions; instead, it saves the `LoginAuth.json` file to the root of the backend folder.

How to run this terminal command?
Follow the instructions listed on Scoutr's readme.md file



```
async function getAuth(url: string) {
  const browser = await navigator.launch()
  const context = await browser.newContext()
  const page = await context.newPage()

  try {
    await page.goto(url)
    await onLogin(page)
    await page.waitForSelector("footer")

    const isVerification = await page.$("#input__email")
    const isProfile = await page.$("p.identity-headline")

    if (isVerification) {
      await onVerification(page)
      await saveAuth(page, "verification")
    } else if (isProfile) {
      await saveAuth(page, "profile")
    } else {
      throw new Error("Unexpected auth page appeared")
    }
  } catch (error) {
    await page.screenshot({ path: "auth.png" })
    console.error("Couldn't obtain auth.”)
  } finally {
    await browser.close()
  }
}
```

Get Page with Context

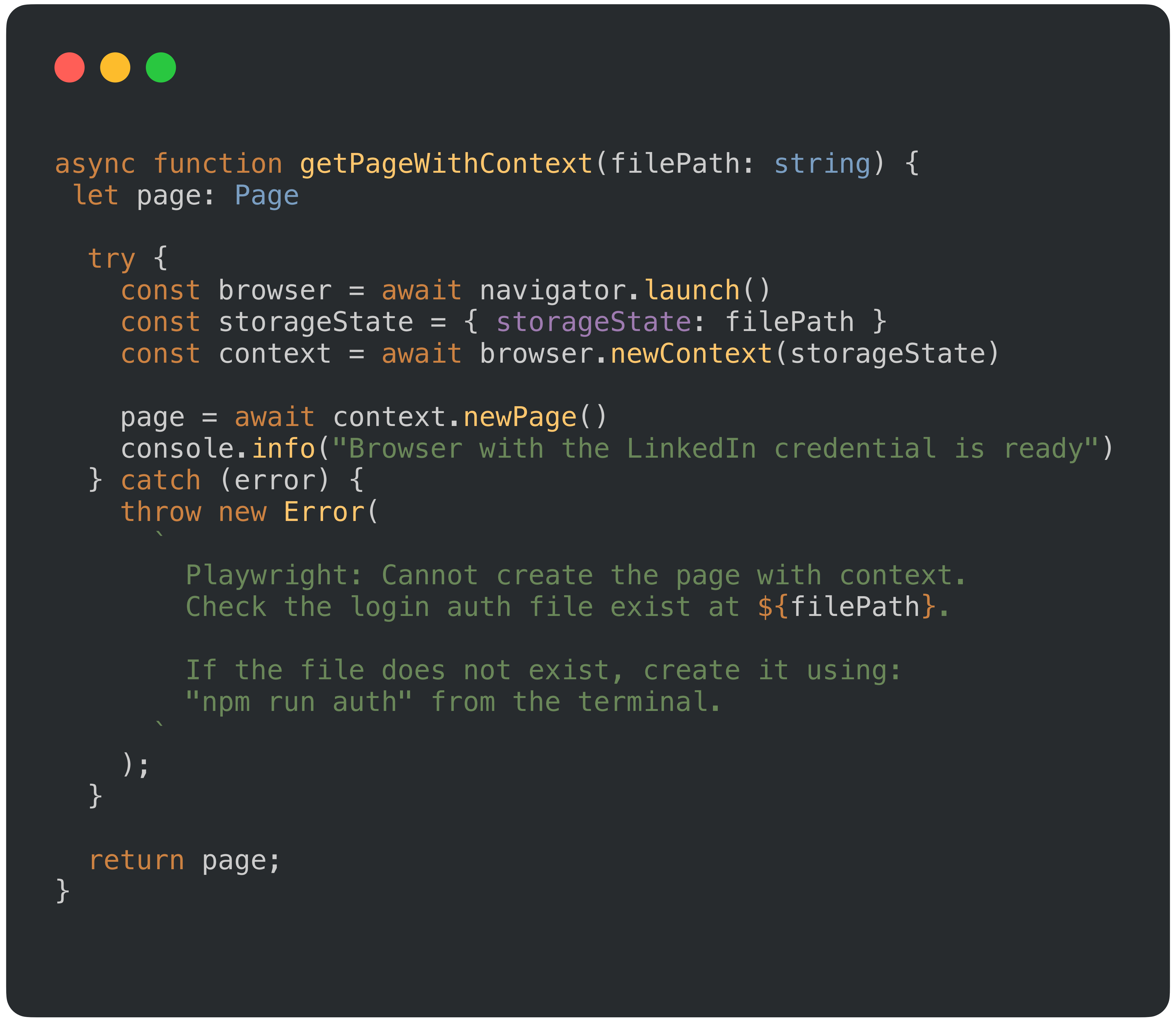
Starts a new Playwright browser instance with the credentials from `getAuth()`. Returns a Page (equivalent of a browser tab) with the login details for scraping or warns you if the Auth is missing.

Input

Name	Type	Description	Example
filePath	`string`	Location of the authentication file, relative to the backend folder.	`“LoginAuth.json”`

Name	Type	Description	Example
page	`Page`	An entire HTML page as a string.	(Check Playwright documentation)

Output



```
async function getPageWithContext(filePath: string) {
 let page: Page

  try {
    const browser = await navigator.launch()
    const storageState = { storageState: filePath }
    const context = await browser.newContext(storageState)

    page = await context.newPage()
    console.info("Browser with the LinkedIn credential is ready")
  } catch (error) {
    throw new Error(
      `
        Playwright: Cannot create the page with context.
        Check the login auth file exist at ${filePath}.

        If the file does not exist, create it using:
        "npm run auth" from the terminal.
      `
    );
  }

  return page;
}
```

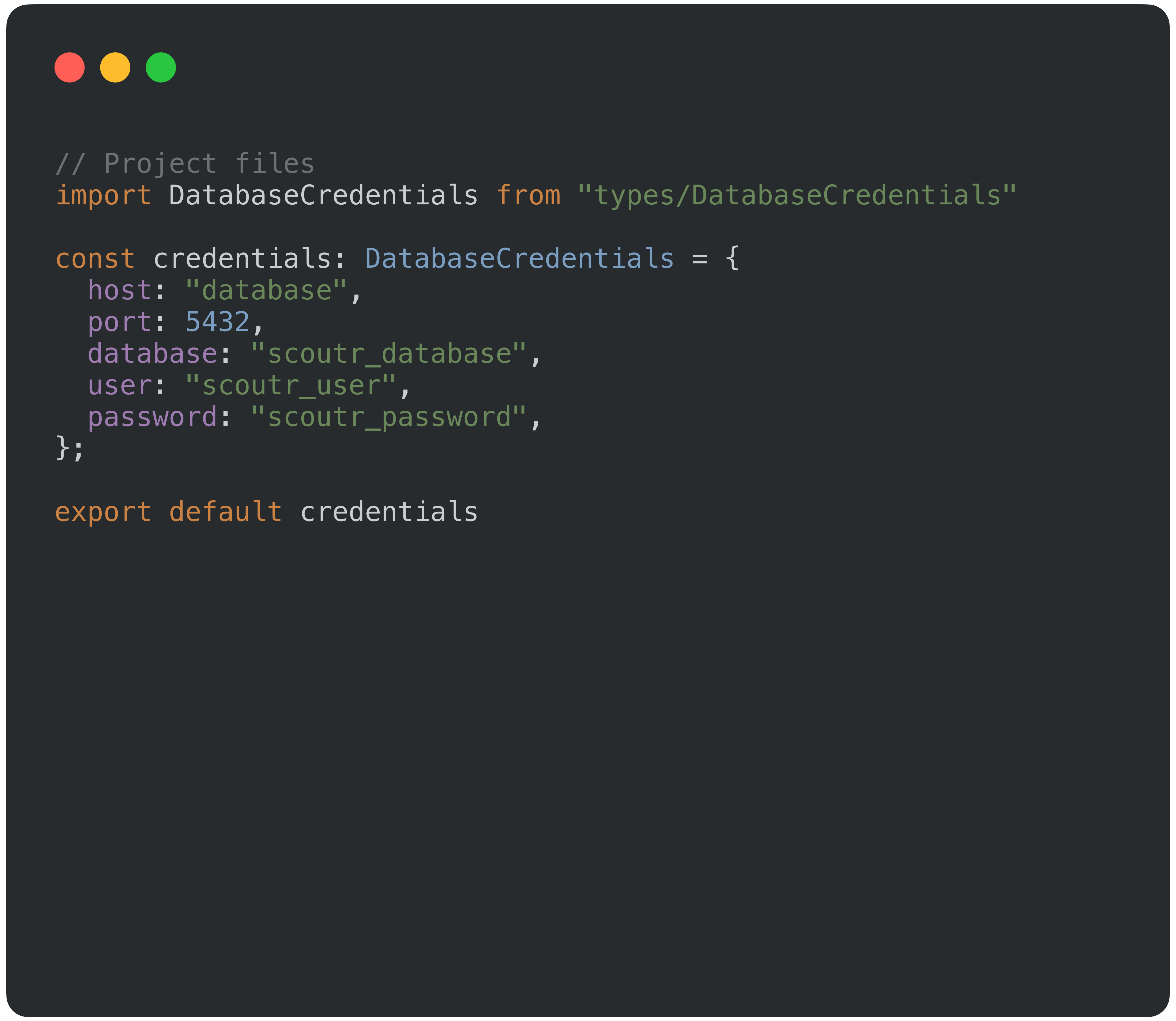
Database

Has coding files for connecting the backend to the database and initializing the tables.

Scoutr

Credentials

A file with the credentials and host address to connect to the Postgres database.



```
// Project files
import DatabaseCredentials from "types/DatabaseCredentials"

const credentials: DatabaseCredentials = {
  host: "database",
  port: 5432,
  database: "scoutr_database",
  user: "scoutr_user",
  password: "scoutr_password",
};

export default credentials
```

Postgres Client

Starts the connection between the backend and database servers, with error messages for common issues.

Input

Name	Type	Description	Example
credentials	`Database Credentials`	URL of the LinkedIn profile.	(Check the types section)

Name	Type	Description	Example
client	`client`	A database client ready to take requests to store and retrieve data.	(Check Node Postgres documentation)

Output

```
// Node modules
import pg from "pg"

async function postgresClient(credentials: Credentials) {
  const client = new pg.Client(credentials)
  const success: `Success message`
  const hostError: `Host error`
  const portError: `Port error`
  const databaseError: `Database`
  const authError: `Auth error`

  try {
    await client.connect()

    console.info(success)
  } catch (error) {
    if (error.code === “ENOTFOUND”) console.error(hostError)
    if (error.code === "ECONNREFUSED") console.error(portError)
    if (error.code === "3D000") console.error(databaseError)
    if (error.code === "28P01") console.error(authError)

    throw new Error(error)
  }

  return client
```

Scan Profile

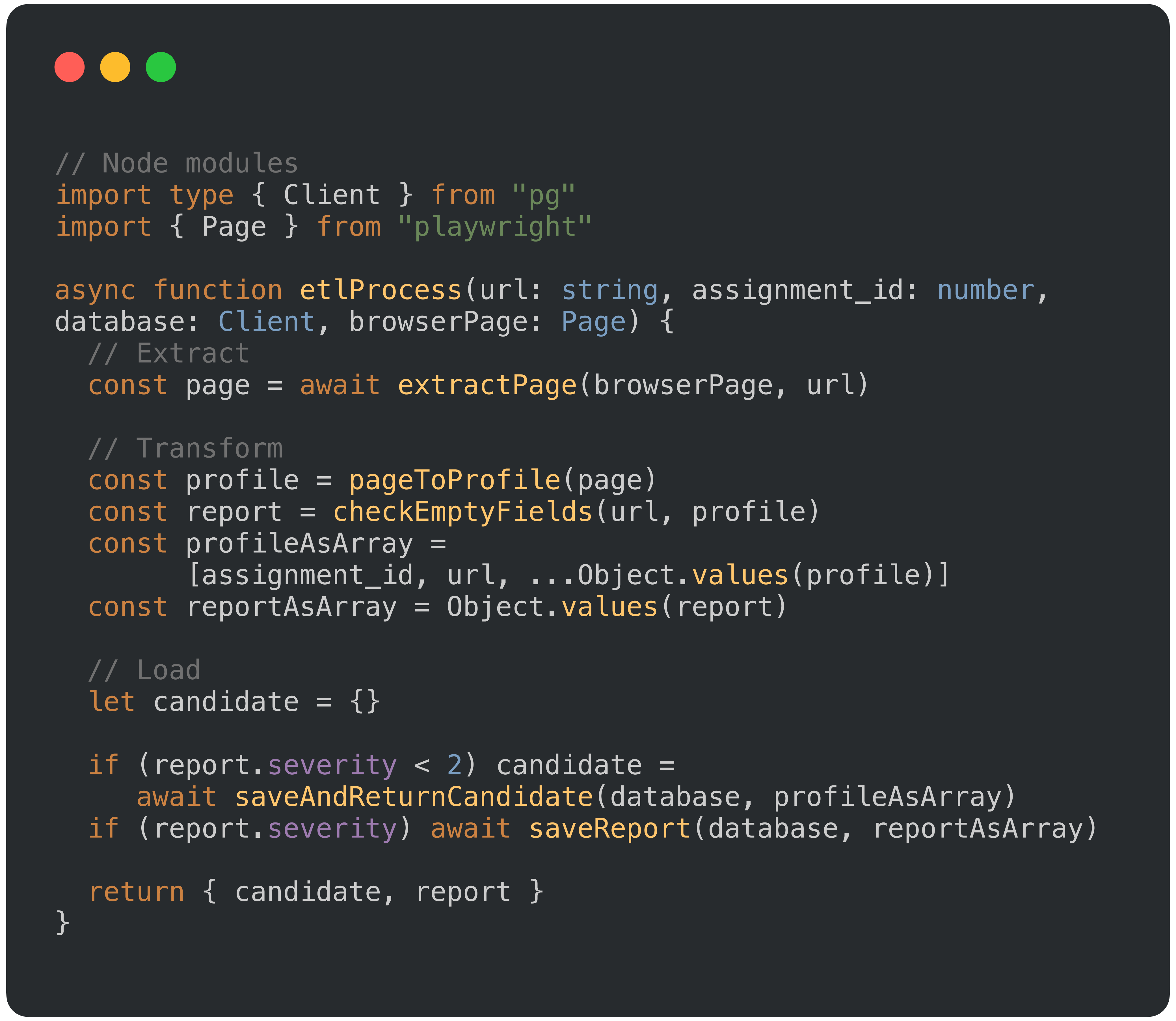
Has coding files to extract the LinkedIn profiles, transform, and store them into the database.

ETL Process

The function that orchestrates the ETL process. It calls sub methods for extract, transform, and load which are documented in the following pages.

Input

Name	Type	Description	Example
url	`string`	URL of the LinkedIn profile.	linkedin.com/in/eduardo-alvarez-nowak
assignment_id	`number`	Identifies the assignment to which the scraped candidate data belongs.	1 (Master Data Specialist - McDonalds)
database	`Client`	Postgres database instance.	(Check Node Postgres documentation)
browserPage	`Page`	Browser page with logged credentials.	(Check Playwright documentation)



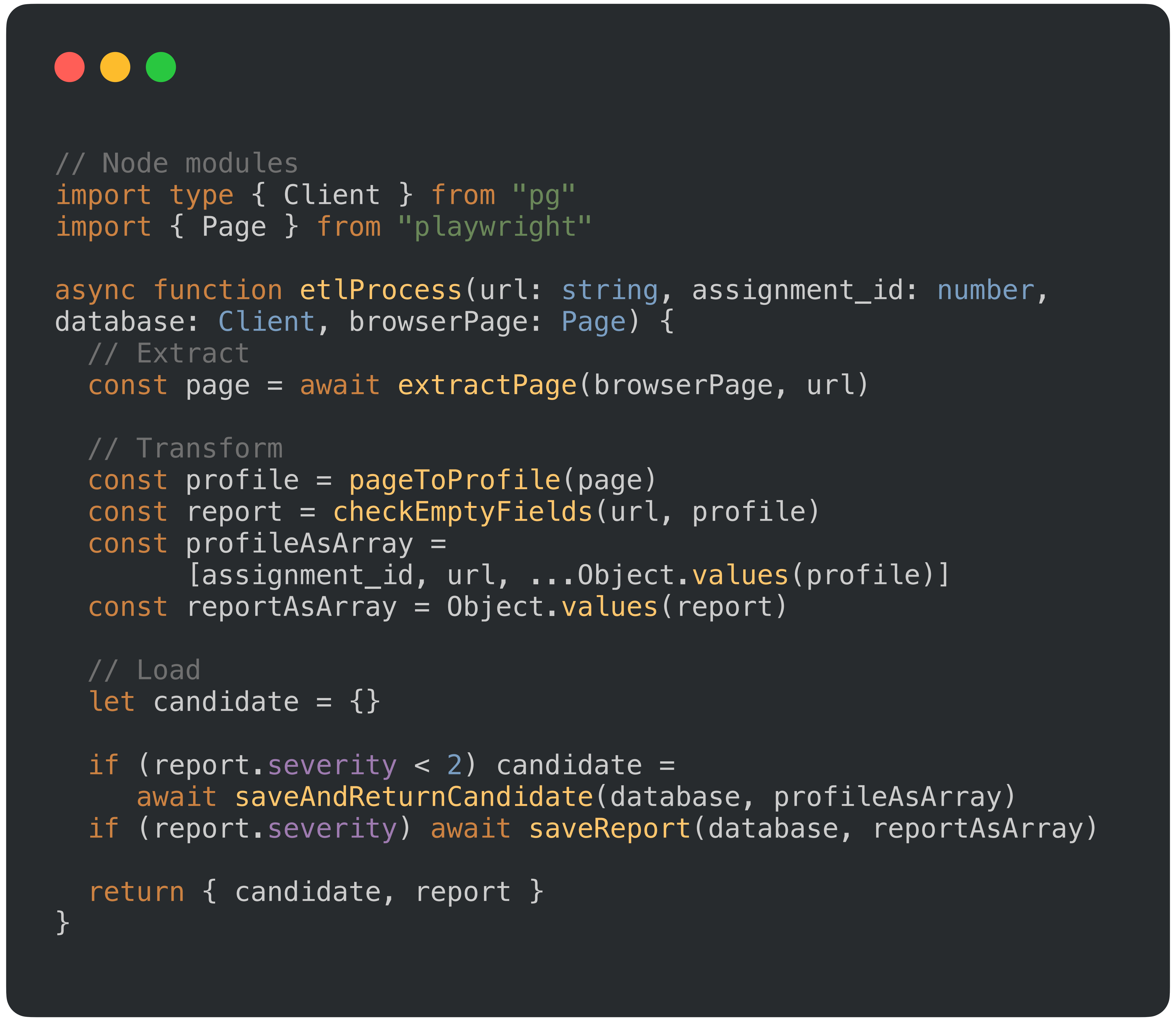
```
// Node modules
import type { Client } from "pg"
import { Page } from "playwright"

async function etlProcess(url: string, assignment_id: number,
database: Client, browserPage: Page) {
  // Extract
  const page = await extractPage(browserPage, url)

  // Transform
  const profile = pageToProfile(page)
  const report = checkEmptyFields(url, profile)
  const profileAsArray =
        [assignment_id, url, ...Object.values(profile)]
  const reportAsArray = Object.values(report)

  // Load
  let candidate = {}

  if (report.severity < 2) candidate =
     await saveAndReturnCandidate(database, profileAsArray)
  if (report.severity) await saveReport(database, reportAsArray)

  return { candidate, report }
}
```

ETL Process

The function that orchestrates the ETL process. It calls sub methods for extract, transform, and load which are documented in the following pages.

Name	Type	Description	Example
candidate	`Candidate`	The candidate data extracted from LinkedIn.	(Check the types section)
report	`ReportLog`	Information to know if the extraction was a success or contain errors.	(Check the types section)

Output



```
// Node modules
import type { Client } from "pg"
import { Page } from "playwright"

async function etlProcess(url: string, assignment_id: number,
database: Client, browserPage: Page) {
  // Extract
  const page = await extractPage(browserPage, url)

  // Transform
  const profile = pageToProfile(page)
  const report = checkEmptyFields(url, profile)
  const profileAsArray =
        [assignment_id, url, ...Object.values(profile)]
  const reportAsArray = Object.values(report)

  // Load
  let candidate = {}

  if (report.severity < 2) candidate =
     await saveAndReturnCandidate(database, profileAsArray)
  if (report.severity) await saveReport(database, reportAsArray)

  return { candidate, report }
}
```

Extract   Transform   Load

Scoutr

Extract Page

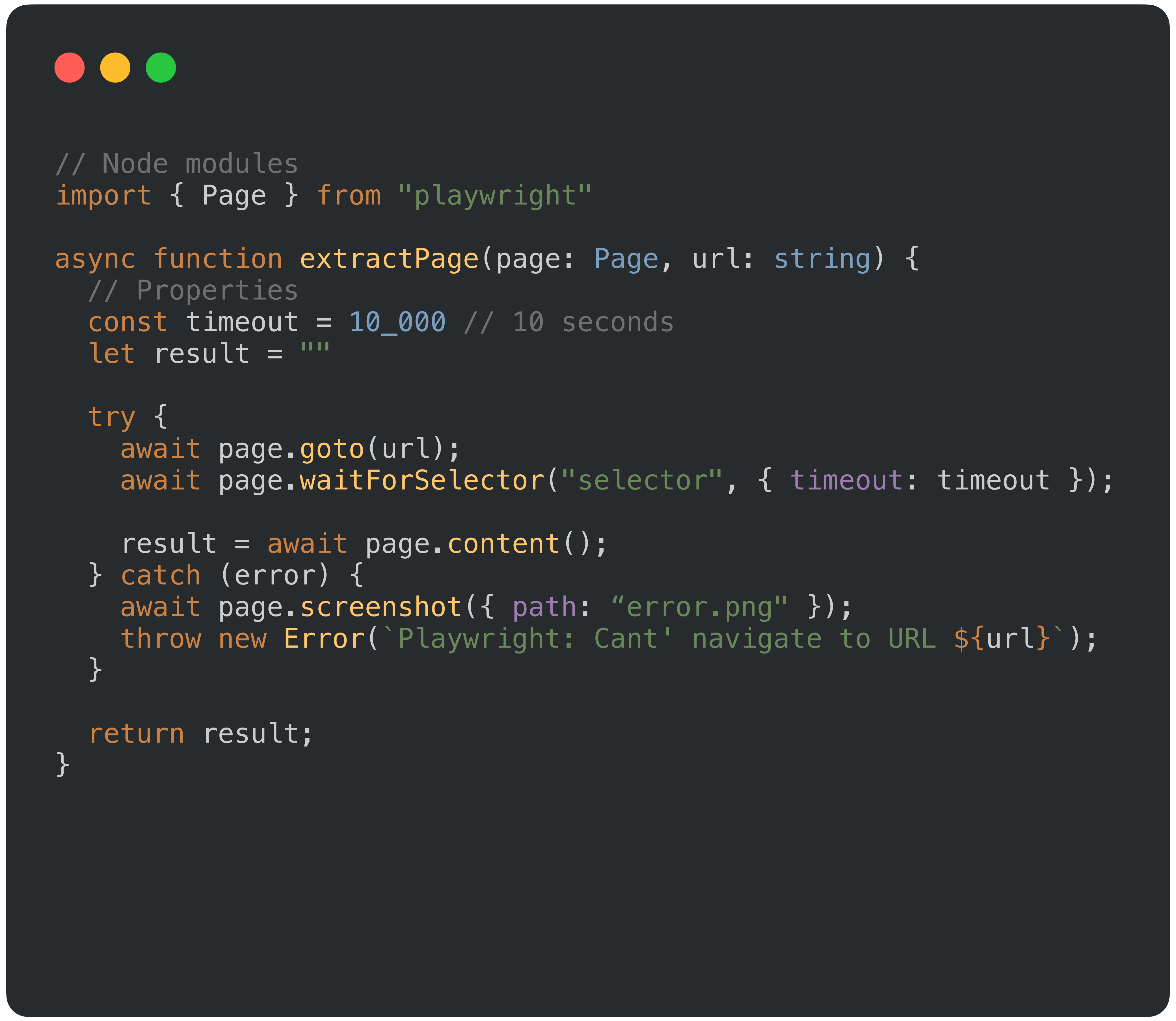
Generates a virtual web browser to navigate to the requested URL, waits until its loaded, extracts all HTML nodes, and returns them as a string

Input

Name	Type	Description	Example
url	`string`	URL of the LinkedIn profile.	linkedin.com/in/eduardo-alvarez-nowak

Name	Type	Description	Example
page	`string`	An HTML page converted to string.	`“<html><h1>Hi</h1></html>”`

Output



```
// Node modules
import { Page } from "playwright"

async function extractPage(page: Page, url: string) {
  // Properties
  const timeout = 10_000 // 10 seconds
  let result = ""

  try {
    await page.goto(url);
    await page.waitForSelector("selector", { timeout: timeout });

    result = await page.content();
  } catch (error) {
    await page.screenshot({ path: “error.png" });
    throw new Error(`Playwright: Cant' navigate to URL ${url}`);
  }

  return result;
}
```

Page to Profile

Obtains the fields for creating a candidate. As LinkedIn users may list the job duration as either a breakdown by promotion or a single total. The `getProfileType()` method manages these variations.

Input

Name	Type	Description	Example
page	`string`	An HTML page converted to string.	`“<html><h1>Hi</h1></html>”`

Name	Type	Description	Example
profile	`LinkedIn Profile`	Object containing transformed data from LinkedIn profile.	(Check the types section)

Output

```
// Node modules
import { load, CheerioAPI } from "cheerio";

function pageToProfile(page: string) {
  const doc: CheerioAPI = load(page)
  const selector = "#experience"
  const scope = document(selector).parent().find(“li”).html()
  const expDoc: CheerioAPI = load(scope)
  const type = getProfileType(expDoc)
  const size = 50; // database column size

  return {
    candidate_name: getCandidateName(doc, size),
    candidate_job_title: getCandidateJobTitle(expDoc, type, size)
    candidate_image_url: candidateImageURL(doc),
    company_name: companyName(expDoc, type, size),
    company_duration_in_months: companyDurationInM(expDoc, type),
    company_image_url: getCompanyImageURL(expDoc),
  };
}
```

Report Empty Fields

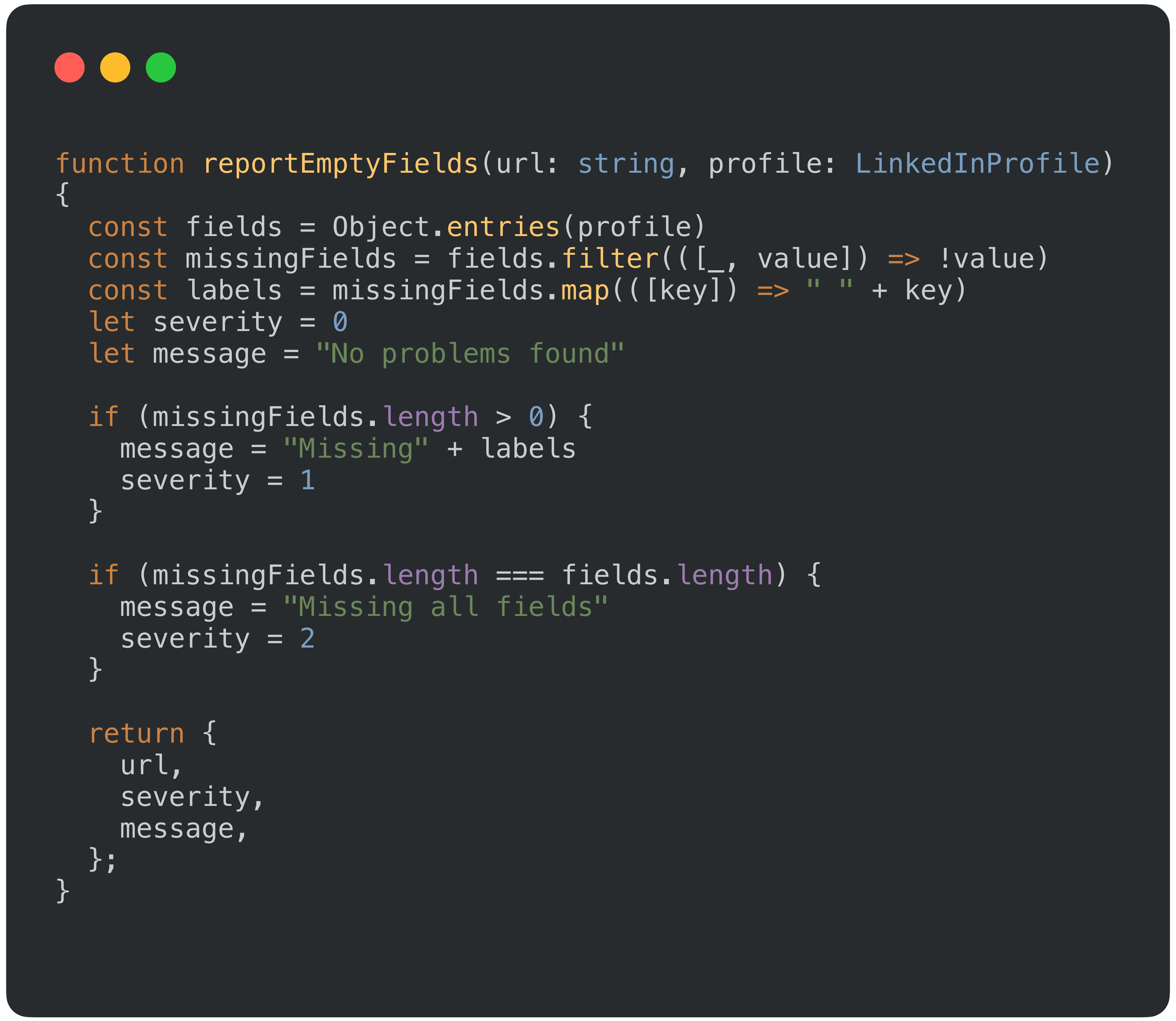
Analyzes a LinkedIn profile URL for empty or missing fields and generates a report outlining any identified issues.

Input

Name	Type	Description	Example
url	`string`	URL of the LinkedIn profile.	linkedin.com/in/eduardo-alvarez-nowak
profile	`LinkedIn Profile`	Object containing transformed data from LinkedIn profile.	(Check the interfaces section)

Name	Type	Description	Example
report	`ReportLog`	Report of empty fields in the profile URL.	(Check the types section)

Output



```
function reportEmptyFields(url: string, profile: LinkedInProfile)
{
  const fields = Object.entries(profile)
  const missingFields = fields.filter(([_, value]) => !value)
  const labels = missingFields.map(([key]) => " " + key)
  let severity = 0
  let message = "No problems found"

  if (missingFields.length > 0) {
    message = "Missing" + labels
    severity = 1
  }

  if (missingFields.length === fields.length) {
    message = "Missing all fields"
    severity = 2
  }

  return {
    url,
    severity,
    message,
  };
}
```

Save and Return Candidate

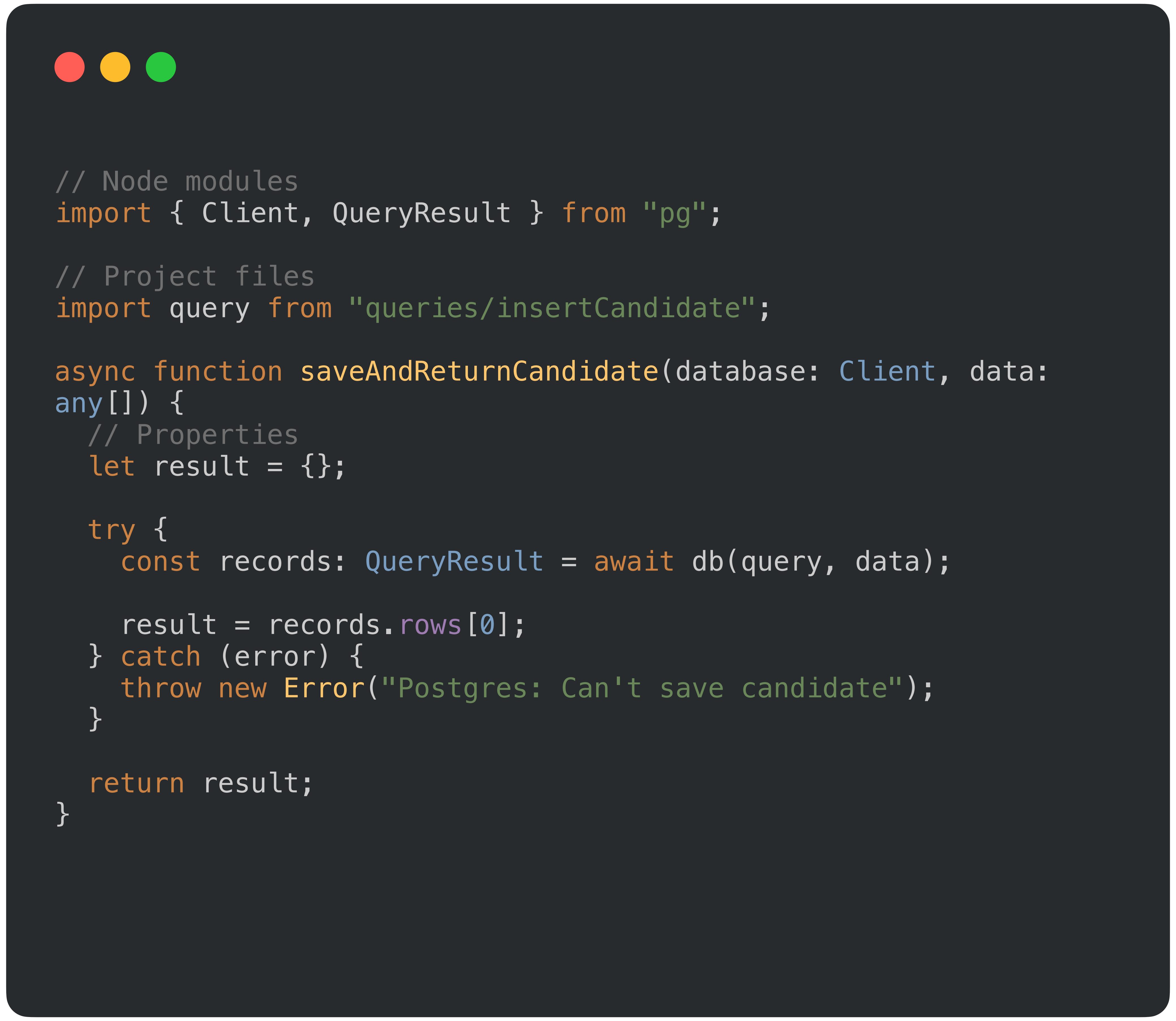
A thin abstraction layer that saves and retrieves the candidate from the database, and catches errors if a candidate field is too long.

Input

Name	Type	Description	Example
database	`Client`	The database instance used to store the data.	(Check the database section)
data	`any[]`	Data coming from a `LinkedInProfile` that will be converted to a candidate.	(Check LinkedInProfile in types section)

Name	Type	Description	Example
results	`Candidate`	The complete candidate data with LinkedIn data, recruiter notes, and ID.	(Check the types section)

Output



```
// Node modules
import { Client, QueryResult } from "pg";

// Project files
import query from "queries/insertCandidate";

async function saveAndReturnCandidate(database: Client, data:
any[]) {
  // Properties
  let result = {};

  try {
    const records: QueryResult = await db(query, data);

    result = records.rows[0];
  } catch (error) {
    throw new Error("Postgres: Can't save candidate");
  }

  return result;
}
```

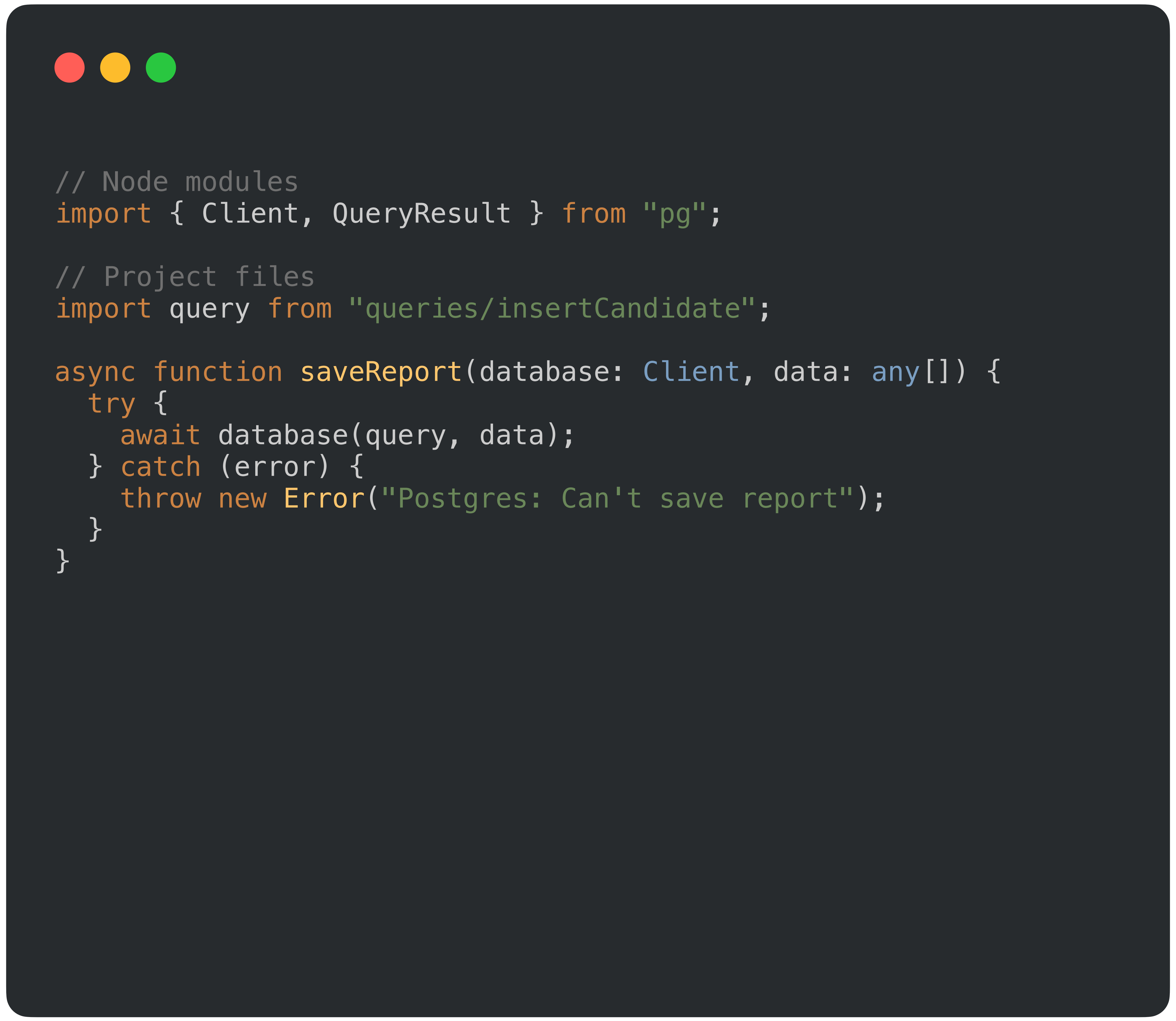
Save Report

A thin abstraction layer that saves the report and catches errors if a report field is too long.

This function does not pass data to other functions; instead, we sent the report made on the transform stage back to the frontend.

Input

Name	Type	Description	Example
database	`Client`	The database instance used to store the data.	(Check the database section)
data	`any[]`	Data coming from a `ReportLog` that will be converted to a candidate.	(Check ReportLog in types section)



```
// Node modules
import { Client, QueryResult } from "pg";

// Project files
import query from "queries/insertCandidate";

async function saveReport(database: Client, data: any[]) {
  try {
    await database(query, data);
  } catch (error) {
    throw new Error("Postgres: Can't save report");
  }
}
```

Types

Has TypeScript interfaces to enforce strong data typing and to self document the data.

Candidates
This interface is a 1 to 1 match to the candidate table schema.

Database Credentials
This interface has the keys Postgres requires to start an database instance.

LinkedIn Profile
This interface has the keys of the information obtained from LinkedIn.

Report Log
This interface has the keys of the analysis report done to a LinkedIn profile.

Report Severity
Key found inside the Report Log. It assigns a numerical value indicating how complete the data extraction is.

Types

Scoutr

👨🏻 Candidates

Key	Type	Description
Id	`number`	Unique identifier for each candidate.
assignment_id	`number`	Foreign key linking candidates to assignments.
date_created	`date`	Date when the candidate profile was added to the database.
linked_in_url	`string`	URL of the candidate's LinkedIn profile.
candidate_name	`string`	Full name of the candidate.
candidate_job_title	`string`	Job title of the candidate.
candidate_image_url	`string`	URL of the candidate's profile picture.
company_name	`string`	The name of the company the candidate is currently working.
company_duration_in_months	`number`	Duration the candidate has been working at the current company, in months.
company_image_url	`string`	URL of the company's logo.
notes	`string`	Any additional notes about the candidate written by the recruiter.
relevance	`number`	Rating from 1 to 5 indicating the candidate's relevance to the assignment, with higher values being more relevant.
contact_status	`number`	Rating from 1 to 6 indicating the status of contact with the candidate, with lower values indicating more recent contact.
contact_date	`date`	Date of the most recent contact with the candidate.

Scoutr

🔐 Database Credentials

Key	Type	Description
host	`string`	The IP address for the database is typically localhost when using Docker, as Docker manages the networking internally.
port	`number`	The port number to connect to the host.
database	`string`	The name of the database.
user	`string`	The admin credential.
password	`string`	The admin password.

LinkedIn Profile

Key	Type	Description
candidate_name	`string`	Full name of the candidate.
candidate_job_title	`string`	Job title of the candidate.
company_name	`string`	The name of the company the candidate is currently working.
company_duration_in_ months	`number`	Duration the candidate has been working at the current company, in months.
company_image_url	`string`	URL of the company's logo.

Scoutr

📈 Report Log

Key	Type	Description
url	`string`	URL of the LinkedIn profile that triggered the error, facilitating later analysis.
severity	`number`	Values taken from the `ReportSeverity` data type to represent the quality of the candidate data.
message	`string`	Detailed error message provided by the subsystem that encountered the error.

Scoutr

⚠️ Report Severity

Key	Type	Description
NO_ERROR	`number`	Candidate data is complete. (value `0`)
SOME_FIELDS_MISSING	`number`	Some fields are missing but usable. (value `1`)
ALL_FIELDS_MISSING	`number`	All fields are missing, data unusable. (value `2`)