Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Dominik Uršič

Avtomatizirano spletno strganje podatkov o javni električni infrastrukturi v Združenem kraljestvu

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Matjaž Kukar

Ljubljana, 2026

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Kandidat: Dominik Uršič

Naslov: Avtomatizirano spletno strganje podatkov o javni električni infrastrukturi v Združenem kraljestvu

Vrsta naloge: Diplomska naloga na univerzitetnem programu prve stopnje Računalništvo in informatika

Mentor: izr. prof. dr. Matjaž Kukar

Opis:

Cilj diplomske naloge je razvoj in implementacija avtomatiziranega sistema za pridobivanje, obdelavo in shranjevanje podatkov o javni električni infrastrukturi v Združenem kraljestvu (UK), s poudarkom na podatkih distributerja National Grid (NG). Sistem bo redno prenašal javno dostopne Excel datoteke s spletne strani NG, jih shranjeval v Google Cloud Storage za arhiviranje, ter jih nato s pomočjo Python skripte obdelal. Posebna pozornost bo namenjena polju "Razpoložljiva kapaciteta"(Demand headroom) ta predstavlja razliko med zanesljivo nosilnostjo omrežnega elementa (transformatorska postaja) in pričakovano najvišjo obremenitvijo. Ta kazalnik določa, koliko dodatne električne moči lahko omrežje še prevzame, preden so potrebne infrastrukturne nadgradnje.. Obdelani podatki bodo naloženi v centralizirano podatkovno bazo PostgreSQL. Celoten proces bo avtomatiziran z uporabo Google Cron Job, ki bo skrbel za redno izvajanje in nadzorom nad napakami.

Title: Automated Web Scraping of Public Electrical Infrastructure Data in the United Kingdom

Description:

The goal of this thesis is to develop and implement an automated system for acquiring, processing and storing data on public electrical infrastructure in the United Kingdom (UK), with emphasis on data from distributor National Grid (NG). The system will regularly download publicly available Excel files

from the NG website, store them in Google Cloud Storage for archiving, and then process them using Python scripts. Special attention will be given to the Demand headroom field, which represents the difference between the reliable capacity of a network element (transformer station) and the expected peak load. Processed data will be loaded into a centralized PostgreSQL database. The entire process will be automated using Google Cron Job for regular execution and error monitoring.

Zahvaljujem se mentorju izr. prof. dr. Matjažu Kukarju za strokovno vodenje in podporo pri izdelavi diplomske naloge. Posebna zahvala gre tudi družini in prijateljem za razumevanje in spodbudo v času študija.

Kazalo

Povzetek

Abstract

1 Uvod **1**

1.1 Namen 2

1.2 Pričakovane koristi in deležniki 2

1.3 Struktura diplomskega dela 3

2 Teoretično ozadje in pregled področja **5**

2.1 Zgodovina spletnega strganja podatkov in izbira orodja 5

2.2 Javna električna infrastruktura v Združenem kraljestvu 7

2.3 National Grid API 7

2.4 Spletno strganje podatkov 8

2.5 Selenium WebDriver 8

2.6 Podatkovni cevovodi 9

2.7 Računalništvo v oblaku 10

2.8 Relacijske baze podatkov 10

2.9 PostgreSQL in delo z grafi 11

3 Metodologija in zasnova sistema **13**

3.1 Identifikacija vira podatkov 13

3.2 Arhitektura sistema 14

3.3 Izbira orodij 14

3.4 Kontrola kakovosti podatkov 16
3.5 Podatkovne baze in pogoni s podporo za grafe 17
3.6 Izbira tehnologije 19

4 Implementacija sistema **21**
4.1 Koraki delovanja sistema 22
4.2 Pridobivanje in shranjevanje podatkov 35
4.3 Obdelava podatkov 36
4.4 Avtomatizacija in nadzor 37
4.5 Kontrola kakovosti podatkov 37

5 Rezultati in evalvacija **39**
5.1 Merila uspešnosti 39
5.2 Način evalvacije 39
5.3 Kriteriji uspeha 39
5.4 Rezultati 40

6 Zaključek **41**
6.1 Zaključki 41
6.2 Možnosti nadaljnjega razvoja 42

Literatura **43**

Seznam uporabljenih kratic

kratica	angleško	slovensko
ETL	Extract, Transform, Load	izlušči, preoblikuj, naloži
GCP	Google Cloud Platform	platforma Google Cloud
GCS	Google Cloud Storage	shramba Google Cloud
NG	National Grid	državno omrežje
Bucket	Bucket	sektor
Scheduler	Scheduler	Razporejevalnik

Povzetek

Naslov: Avtomatizirano spletno strganje podatkov o javni električni infrastrukturi v Združenem kraljestvu

Avtor: Dominik Uršič

Cilj diplomske naloge je razvoj in implementacija avtomatiziranega sistema za pridobivanje, obdelavo in shranjevanje podatkov o javni električni infrastrukturi v Združenem kraljestvu, s posebnim poudarkom na podatkih distributerja National Grid. Takšen sistem omogoča pregleden in enostaven dostop do ključnih informacij o električni infrastrukturi, kar je bistvenega pomena za podjetja, ki se ukvarjajo z nameščanjem električnih polnilnic. Dostop do celovitih podatkov o posameznih transformatorskih postajah namreč omogoča hitrejše in stroškovno učinkovitejše načrtovanje ter postavitev polnilne infrastrukture. Trenutno so ti podatki razpršeni po različnih virih, njihovo ročno zbiranje in posodabljanje pa je zamudno in podvrženo napakam. Sistem bo redno prenašal javno dostopne Excel datoteke s spletne strani National Grid, jih shranjeval v Google Cloud Storage za arhiviranje, ter jih nato s pomočjo Python skripte obdelal. Posebna pozornost bo namenjena polju "Demand Headroom", ki označuje razpoložljivo kapaciteto omrežnega elementa. Ta kazalnik predstavlja razliko med zanesljivo nosilnostjo posameznega omrežnega elementa in njegovo pričakovano najvišjo obremenitvijo ter tako določa maksimalno dodatno obremenitev, ki jo element še lahko prenese brez potrebe po infrastrukturnih nadgradnjah. Obdelani podatki bodo naloženi v centralizirano podatkovno bazo PostgreSQL. Celoten proces bo avtomatiziran z uporabo Google Cron Job, ki bo skrbel za redno izvajanje in

nadzor nad napakami. Sistem bo zasnovan po principu ETL (Extract, Transform, Load), kar bo zagotovilo enostavno vzdrževanje, ažurnost podatkov in zanesljivost za končne uporabnike.

Ključne besede: spletno strganje, avtomatizacija, podatkovni cevovodi, električna infrastruktura, National Grid.

Abstract

Title: Automated Web Scraping of Public Electrical Infrastructure Data in the United Kingdom

Author: Dominik Uršič

The objective of this thesis is to develop and implement an automated system for acquiring, processing, and storing data on public electrical infrastructure in the United Kingdom, with particular emphasis on data from the distributor National Grid. Such a system enables clear and easy access to key information about electrical infrastructure, which is essential for companies involved in the installation of electric charging stations. Access to comprehensive data on individual transformer stations enables faster and more cost-effective planning and deployment of charging infrastructure. Currently, this data is scattered across various sources, and its manual collection and updating is time-consuming and error-prone. The system will regularly download publicly available Excel files from the National Grid website, store them in Google Cloud Storage for archiving, and then process them using Python scripts. Special attention will be given to the "Demand Headroom" field, which indicates the available capacity of a network element. This indicator represents the difference between the reliable capacity of an individual network element and its expected peak load, thus determining the maximum additional load that the element can still handle without requiring infrastructure upgrades. The processed data will be loaded into a centralized PostgreSQL database. The entire process will be automated using Google Cron Job, which will handle regular execution and error monitoring. The

system will be designed according to the ETL (Extract, Transform, Load) principle, ensuring easy maintenance, data currency, and reliability for end users.

Keywords: web scraping, automation, data pipelines, electrical infrastructure, National Grid.

Poglavje 1

Uvod

V sodobnem svetu se električna energija smatra za eno najpomembnejših infrastruktur, ki omogoča delovanje industrije, gospodinjstev in storitev. Zanesljiv dostop do ažurnih podatkov o električni infrastrukturi je ključen za različne analize, načrtovanje in sprejemanje odločitev na energetskem področju.

V Združenem kraljestvu je National Grid eden pomembnejših operaterjev elektroenergetskega omrežja, ki pokriva geografska področja, vključno z regijami East Midlands, West Midlands, South Wales ter South West England. Kot ključni sistemski operater je National Grid odgovoren za distribucijo električne energije na svojem območju ter zagotavlja stabilno in zanesljivo oskrbo z električno energijo za milijone gospodinjstev in podjetij. Ti podatki so javno dostopni v obliki Excel datotek na njihovi spletni strani, vendar je njihovo ročno zbiranje in posodabljanje zamudno in podvrženo napakam.

Diplomska naloga naslavlja problem neučinkovitega in neurejenega dostopa do podatkov o električni infrastrukturi z razvojem avtomatiziranega sistema za pridobivanje, obdelavo in shranjevanje podatkov National Grid. Sistem bo implementiran z uporabo modernih tehnologij in metod za obdelavo podatkov, vključno s spletnim strganjem, podatkovnimi cevovodi in računalništvom v oblaku.

Glavni cilj naloge je razvoj in implementacija avtomatiziranega sistema,

ki bo sposoben redno prenašati Excel datoteke s spletne strani National Grid, jih shranjevati v Google Cloud Storage, obdelovati s Python skriptami in nalagati v centralizirano PostgreSQL podatkovno bazo. Sistem bo zasnovan po principu ETL (Extract, Transform, Load) in bo vključeval mehanizme za kontrolo kakovosti podatkov ter avtomatsko obveščanje o stanju procesa.

1.1 Namen

Primarni namen sistema je *avtomatizirati* zajem javno dostopnih podatkov National Grid Electricity Distribution (NGED) ter zagotoviti standardizirano, časovno žigosano in sledljivo kopijo podatkov za nadaljnje analize. Sistem zmanjšuje ročno delo, ter možnost napak pri ročnem prenosu podatkov.

Sistem bistveno skrajša čas do podatkov, saj avtomatizira prenos in validacijo ter s tem zmanjšuje ročno delo in napake, hkrati pa uvaja verzioniranje za sledljivost. Rezultat so enotni in ažurni nabori, primerni za ključne analize pa do scenarijev rasti odjema in ocen "headrooma". [16]

Sistem zagotavlja trdno analitično podlago za taktično in strateško odločanje v energetskem sektorju:. Vse od operaterjev in načrtovalcev omrežja do razvijalcev ter investitorjev, javnih ustanov in regulatorjev ter raziskovalcev in svetovalcev. Hkrati je orodje uporabno za vse, ki načrtujejo nove objekte elektroinfrastrukture na primer lokacije električnih polnilnic, ker sistem omogoča hitro preverjanje razpoložljivih kapacitet, omejitev in možnosti priključitve na obstoječe omrežje, ter s tem podpira utemeljeno izbiro lokacije.

1.2 Pričakovane koristi in deležniki

Implementacija avtomatiziranega sistema za pridobivanje podatkov o električni infrastrukturi bo prinesla oprijemljive koristi različnim deležnikom v energetskem sektorju. Sistem bo zmanjšal časovne zahteve za pridobivanje podatkov iz trenutnih 2-4 ur mesečnega ročnega dela na nekaj minut avto-

matiziranega procesa, kar predstavlja velik prihranek delovnega časa.

Ključni deležniki sistema bodo energetska podjetja in razvijalci projektov, ki bodo pridobili takojšen dostop do ažurnih podatkov o razpoložljivih kapacitetah (Demand Headroom), kar bo omogočilo hitrejše in bolj informirane odločitve o lokacijah novih projektov. Svetovalna podjetja v energetskem sektorju bodo lahko svojim strankam ponudila natančnejše analize in hitreje pripravila študije izvedljivosti, medtem ko bodo investitorji v obnovljive vire energije pridobili kritične informacije za oceno primernosti lokacij za solarne elektrarne, vetrne parke ali baterijske sisteme. Dolgoročno bo sistem omogočil enostavno razširitev na dodatne distributerje električne energije po celotni Veliki Britaniji, saj bo vzpostavljena arhitektura zlahka prilagodljiva za integracijo podatkov iz UK Power Networks, Scottish Power Energy Networks in drugih operaterjev. Strukturirana zgodovinska baza podatkov bo omogočila napredno analitiko za prepoznavanje trendov porabe, napovedovanje prihodnjih kapacitet in identifikacijo kritičnih točk v omrežju, kar bo podprlo strateško načrtovanje investicij v energetsko infrastrukturo in pospešilo prehod na trajnostne vire energije.

1.3 Struktura diplomskega dela

V 2. poglavju bomo predstavili teoretično ozadje in pregled področja, vključno s pregledom javne električne infrastrukture v UK, tehnik spletnega strganja in podatkovnih cevovodov. 3. poglavje bo opisalo metodologijo in zasnovo sistema, medtem ko bo 4. poglavje predstavilo podrobnosti implementacije. V 5. poglavju bomo analizirali rezultate in evalvacijo sistema, sledil pa bo zaključek z diskusijo o doseženih ciljih in možnostih za nadaljnje delo.

Poglavje 2

Teoretično ozadje in pregled področja

2.1 Zgodovina spletnega strganja podatkov in izbira orodja

Avtomatizirano zajemanje podatkov s spletnih strani se je v zadnjih dvajsetih letih precej spremenilo, ker so se spremenile tudi same spletne strani. V začetkih interneta so bile strani večinoma statične v obliki HTML dokumenta, zato je za pridobivanje podatkov zadostovala preprosta analiza izvorne kode. Takrat so bile v uporabi predvsem rešitve, ki so temeljile na regularnih izrazih in osnovnem razčlenjevanju HTML strukture. Pomemben mejnik je predstavljal razvoj specializiranih knjižnic za delo z HTML dokumenti. Leta 2004 je prišel Beautiful Soup, ki je spletno strganje naredil veliko bolj dostopno. Ta Python knjižnica je z intuitivno sintakso omogočila enostavno ekstrakcijo podatkov iz kompleksnih HTML struktur, vendar je bila še vedno omejena na statično vsebino. V tem času so se podobne knjižnice razvijale tudi za druge programske jezike (jsoup za Javo).

Leta 2008 je Scrapy prinesel nov pristop s svojo asinhrono arhitekturo, ki je omogočila učinkovito obdelavo velikega števila strani hkrati. To Python ogrodje je postalo nekakšen standard za večje projekte spletnega strganja, saj

je omogočalo distribuirano delo, avtomatsko upravljanje s piškotki in sejami ter robustno obravnavo napak. Takrat pa se je začel tudi velik premik v spletnem razvoju. Z uvedbo AJAX tehnologij in Single Page Applications (SPA) so spletne strani postale veliko bolj dinamične. Vsebina se je začela nalagati asinhrono, elementi so se generirali z JavaScript kodo, podatki pa so se pridobivali preko API klicev šele po začetnem nalaganju strani. Tradicionalne metode strganja naenkrat niso več zadostovale.

Rešitev je prišla z orodji za avtomatizacijo brskalnikov. Selenium WebDriver, ki je bil sicer prvotno razvit za testiranje spletnih aplikacij, se je izkazal za odlično orodje tudi za strganje kompleksnih strani. Za razliko od prejšnjih pristopov Selenium upravlja pravi brskalnik – lahko izvaja JavaScript, čaka na dinamično naložene elemente, simulira klike in druge uporabniške interakcije ter se spopada s kompleksnimi navigacijskimi tokovi. Selenium deluje preko WebDriver protokola, ki ga je leta 2018 standardiziral W3C konzorcij. Ta protokol omogoča komunikacijo med programsko kodo in brskalnikom na način, ki deluje prek različnih brskalnikov (Chrome, Firefox, Safari, Edge) in operacijskih sistemov. Osnova je client-server model, kjer aplikacija pošilja ukaze gonilniku brskalnika, ta pa jih izvaja in vrača rezultate.

Za projekt avtomatizacije National Grid platforme je bil Selenium najboljša izbira iz več razlogov. Platforma zahteva avtentikacijo preko prijavnega obrazca, uporablja dinamično nalaganje vsebine, vključuje interaktivne zemljevide in vizualizacije ter ima tudi določene zaščitne mehanizme proti avtomatizaciji. Zato smo uporabili še undetected-chromedriver, ki z različnimi tehnikami (modifikacija navigator objekta, odstranjevanje WebDriver zastavic, simulacija realističnih vzorcev gibanja miške) poskrbi, da sistem ne zazna avtomatizacije. Prehod od Beautiful Soup preko Scrapy do Selenium tako zrcali razvoj spletnih tehnologij. Medtem ko enostavnejša orodja še vedno dobro služijo za statične strani, kompleksne moderne aplikacije zahtevajo polno simulacijo brskalnika. Selenium z zmožnostjo izvajanja JavaScript kode, čakanja na asinhrono naložene elemente in simulacije uporabniških in-

terakcij trenutno predstavlja najzmogljivejšo rešitev za avtomatizirano pridobivanje podatkov iz sodobnih spletnih platform.

2.2 Javna električna infrastruktura v Združenem kraljestvu

Združeno kraljestvo ima kompleksen sistem električne infrastrukture, kjer različni akterji upravljajo prenos in distribucijo električne energije. Država je razdeljena na 14 raličnih geografskih območij za katera je odgovornih 6 podjetij. National Grid je eden glavnih operaterjev prenosnega omrežja, ki povezuje elektrarne z distribucijskimi omrežji [10].

National Grid redno objavlja podatke o zmogljivostih omrežja, vključno s podatki o "Demand Headroom"parametru, ki opisuje razpoložljivo kapaciteto omrežja za nove priključitve. Ti podatki so ključni za energetske analize, načrtovanje infrastrukture in sprejemanje poslovnih odločitev. [8]

2.3 National Grid API

API Connected Data Portal podjetja National Grid predstavlja standardiziran vmesnik za programski dostop do javno objavljenih energetskih in omrežnih podatkov. Kljub temu, da pokriva podatkovno domeno, ki se vsebinsko delno prekriva z obravnavanim področjem naloge, sistem izkazuje ključne strukturne omejitve, ki onemogočajo njegovo učinkovito implementacijo v kontekstu obravnavanih podatkov. Fundamentalna omejitev izhaja iz vnaprej definirane arhitekture podatkovnih nizov, pri čemer ponudnik določa tako obseg kot strukturo razpoložljivih podatkov. API v svoji trenutni konfiguraciji ne omogoča fleksibilnega prilagajanja podatkovnih zahtevkov specifičnim potrebam posameznega primera uporabe, temveč zgolj izpostavlja že obstoječe podatkovne nize v fiksni obliki, kot so bili izvirno objavljeni.

2.4 Spletno strganje podatkov

Spletno strganje (web scraping) je tehnika avtomatskega pridobivanja podatkov s spletnih strani [9]. V Pythonu obstajajo različne knjižnice, sam pa bom uporabil Selenium, ki omogoča programsko upravljanje spletnega brskalnika in interakcijo z dinamičnimi spletnimi stranmi. Selenium deluje tako, da simulira dejanja pravega uporabnika v brskalniku, lahko klikne gumbe, izpolni obrazce, počaka na nalaganje elementov in izvozi podatke. Proces se tipično začne z inicializacijo brskalnika (v našem primeru Chromium z undetected-chromedriver za izogibanje detekciji avtomatizacije), nato pa skript sistematično navigira po spletni strani. Najprej se izvede prijava z vnosom uporabniškega imena in gesla, sledi navigacija do želenega dela aplikacije, kjer se sproži izvoz podatkov. Posebni funkciji (WebDriverWait in expected-conditions) zagotavljata, da skript počaka na popolno nalaganje elementov, preden z njimi upravlja, kar preprečuje napake zaradi asinhronega nalaganja vsebine. Ko je datoteka prenesena, se avtomatsko preimenuje s časovnim žigom in shrani v določeno mapo za nadaljnjo obdelavo. Pri spletnem strganju je pomembno upoštevati etične in pravne vidike, vključno s spoštovanjem robots.txt datotek, omejitev frekvence zahtev in pogojev uporabe spletnih strani [4]. V našem primeru gre za pridobivanje javno dostopnih podatkov z uporabo legitimnih prijavnih podatkov, kar zagotavlja skladnost s pogoji uporabe National Grid platforme.

2.5 Selenium WebDriver

Selenium WebDriver je odprtokodno orodje za avtomatizacijo spletnih brskalnikov, ki omogoča programsko interakcijo s spletnimi stranmi [13]. Temelji na arhitekturi tipa odjemalec–strežnik: aplikacija pošilja ukaze brskalniku prek protokola WebDriver, brskalnikov gonilnik pa ta navodila izvede in vrne rezultat. Ko Python skripta pokliče Seleniumovo metodo, se ta pretvori v HTTP zahtevo, ki jo gonilnik brskalnika (na primer ChromeDriver za Google Chrome) izvede neposredno v uporabniškem vmesniku.

Selenium omogoča različne načine za iskanje elementov na spletni strani, kot so uporaba identifikatorjev, CSS selektorjev, izrazov XPath ali imen razredov. Z uporabo mehanizmov WebDriverWait in expected conditions lahko skripta eksplicitno počaka, da se določen pogoj izpolni (na primer, da element postane klikljiv), preden nadaljuje z izvajanjem. Tak pristop je ključen pri dinamičnih spletnih straneh, kjer se vsebina nalaga asinhrono prek JavaScripta.

2.6 Podatkovni cevovodi

Podatkovni cevovodi (data pipelines) so avtomatizirani procesi za prenos podatkov od virov do končnih destinacij z možnostjo transformacije med potjo [15]. Tradicionalni ETL (Extract, Transform, Load) pristop se v zadnjem času vse bolj nadomešča z ELT (Extract, Load, Transform) pristopom, ki omogoča večjo fleksibilnost pri obdelavi podatkov. Ključna razlika med pristopoma je v zaporedju operacij. Pri ETL pristopu se podatki najprej ekstraktirajo iz vira, nato transformirajo v vmesnem okolju in šele nato naložijo v ciljno podatkovno bazo. Nasprotno pa ELT pristop najprej naloži surove podatke neposredno v podatkovno bazo, kjer se transformacije izvajajo z uporabo SQL poizvedb in drugih orodij znotraj samega RDBMS sistema. V našem sistemu implementiramo klasični ETL pristop, ki se je izkazal za najbolj primernega glede na naravo podatkov in zahteve sistema. Uporaba ETL pristopa pozitivno vpliva na zmogljivost RDBMS sistema, saj PostgreSQL prejme le čiste, validirane podatke, kar zmanjšuje potrebo po kompleksnih SQL transformacijah in s tem obremenitev podatkovne baze. To omogoča, da se PostgreSQL osredotoči na svoje primarne naloge, učinkovito shranjevanje, indeksiranje in serviranje podatkov končnim uporabnikom. Manjša obremenitev baze pomeni hitrejše odzivne čase pri poizvedbah, nižjo porabo sistemskih virov in večjo skalabilnost sistema. Dodatno ETL pristop omogoča lažje odkrivanje in reševanje napak v podatkih, saj se te obravnavajo še pred vnosom v produkcijsko bazo, kar zagotavlja večjo integriteto podatkov in

zanesljivost celotnega sistema.

2.7 Računalništvo v oblaku

Google Cloud Platform (GCP) ponuja različne storitve za delo s podatki, vključno z Google Cloud Storage za shranjevanje datotek in Google Cloud Scheduler za avtomatizirano izvajanje opravil [7]. Te storitve omogočajo skalabilno in zanesljivo infrastrukturo za podatkovne cevovode.

2.8 Relacijske baze podatkov

PostgreSQL je zmogljiva odprtokodna objektno-relacijska podatkovna baza, ki se pogosto uporablja za shranjevanje strukturiranih podatkov v podatkovnih aplikacijah [11]. Omogoča kompleksne poizvedbe, ACID transakcije, različne razširitve za specifične potrebe ter napredno indeksiranje. Za naš sistem avtomatiziranega pridobivanja podatkov o električni infrastrukturi so ključne funkcionalne zahteve PostgreSQL sistema naslednje: podpora za velike količine časovnih serij podatkov (preko 1300 zapisov, seveda lahko postgreSQL obdela še veliko več zapisov), zmožnost hitrega vstavljanja novih podatkov preko bulk **INSERT** operacij, učinkovito indeksiranje na polju Demand Headroom za hitre poizvedbe, podpora za JSON podatkovne tipe za shranjevanje semi-strukturiranih metapodatkov, ter zmožnost izvajanja kompleksnih analitičnih poizvedb z window funkcijami. Sistem mora zagotavljati tudi verzioniranje podatkov, kjer se ohranjajo vse zgodovinske verzije za revizijske sledi in analizo trendov. Pomembna je tudi konfiguracija avtomatskega vzdrževanja preko autovacuum procesa, ki zagotavlja optimalno zmogljivost tudi pri velikem številu **UPDATE** in **DELETE** operacij. Dodatno mora sistem podpirati replikacijo za visoko razpoložljivost, omogočati point in time recovery za zaščito pred izgubo podatkov, ter imeti nastavljeno redno varnostno kopiranje (pg dump) vsaj enkrat dnevno. PostgreSQL razširitve kot so pg cron za avtomatizirane naloge znotraj baze in timescaledb za optimizirano

delo s časovnimi serijami dodatno izboljšajo funkcionalnost sistema za naše specifične potrebe pri upravljanju podatkov.

2.9 PostgreSQL in delo z grafi

Poleg klasičnih relacijskih podatkovnih modelov se v sodobnih podatkovnih sistemih vse pogosteje pojavlja potreba po obdelavi grafovskih struktur, kjer so podatki predstavljeni kot vozlišča (ang. *nodes*) in povezave med njimi (ang. *edges*). Takšen pristop je posebej primeren za modeliranje omrežij, kot so energetska infrastruktura, prometna omrežja ali socialne mreže, kjer relacije med entitetami nosijo enako pomembno informacijo kot same entitete [1].

PostgreSQL kljub temu, da primarno sodi med relacijske podatkovne baze, omogoča učinkovito delo z grafi na več različnih načinov. Osnovni pristop temelji na modeliranju grafovskih struktur z uporabo relacijskih tabel, kjer se vozlišča hranijo v eni tabeli, povezave pa v drugi tabeli z referencami (tujimi ključi) na izvorno in ciljno vozlišče. Takšen model omogoča uporabo standardnih SQL poizvedb za osnovne grafovske operacije, kot so iskanje sosedov, stopnje vozlišč in enostavne poti [5].

Za zahtevnejše grafovske analize PostgreSQL ponuja podporo z razširitvami. Ena najpomembnejših je *Apache AGE*, ki razširja PostgreSQL z lastnostmi večmodelne baze podatkov in dodaja podporo za grafovni podatkovni model ter poizvedovalni jezik openCypher. Apache AGE omogoča izvajanje kompleksnih grafovskih poizvedb, kot so iskanje najkrajših poti, detekcija povezanih komponent in analiza omrežnih vzorcev, neposredno znotraj PostgreSQL okolja, brez potrebe po ločeni grafovni bazi podatkov.

Alternativni pristop predstavlja razširitev *pgRouting*, ki je specializirana za delo z grafi v prostorskih podatkih in se pogosto uporablja v kombinaciji z razširitvijo PostGIS. Čeprav je primarno namenjena prometnim in geografskim omrežjem, se lahko njeni algoritmi za iskanje poti (Dijkstra, A*, Bellman-Ford) uporabijo tudi za analizo energetskih ali infrastrukturnih

omrežij, kjer so povezave utežene z zmogljivostmi ali obremenitvami.

V kontekstu električne infrastrukture je grafovski model posebej primeren za predstavitev prenosnega in distribucijskega omrežja. Vozlišča lahko predstavljajo transformatorske postaje, razdelilne točke ali geografska območja, povezave pa fizične ali logične povezave med njimi. Atributi povezav, kot so maksimalna zmogljivost, trenutna obremenitev ali razpoložljiv *Demand Headroom*, omogočajo izvajanje analitičnih poizvedb, ki podpirajo odločanje pri načrtovanju novih priključitev ali nadgradenj omrežja.

Uporaba PostgreSQL za delo z grafi prinaša pomembno prednost v obliki enotne podatkovne platforme. Relacijski, časovni in grafovski podatki so shranjeni v istem sistemu, kar poenostavi arhitekturo, zmanjša operativne stroške in omogoča kombiniranje klasičnih SQL poizvedb z grafovskimi analizami. Takšen hibridni pristop je posebej primeren za sisteme, kjer grafovske analize dopolnjujejo obstoječe relacijske in časovne podatkovne modele.

Poglavje 3

Metodologija in zasnova sistema

3.1 Identifikacija vira podatkov

Glavni vir podatkov je spletna stran National Grid, kjer so objavljene Excel datoteke z informacijami o zmogljivostih omrežja. URL za dostop do podatkov je `https://www.nationalgrid.co.uk/our-network/network-capacity-map-application`.

Datoteke vsebujejo nabor tehničnih parametrov električne infrastrukture, strukturiranih v CSV formatu. Ključna polja vključujejo Substation Name (ime postaje), Asset Type (vrsto postaje), ter koordinate lokacij (Latitude in Longitude) za lažje geografsko pozicioniranje. Posebno pozornost namenjamo polju Demand Headroom (MW), ki predstavlja razpoložljivo kapaciteto za nove priključitve in je zelo pomeben parameter za razvijalce projektov pri ocenjevanju izvedljivosti novih povezav. Dodatni tehnični parametri vključujejo Peak Demand (najvišja obremenitev) in Network Reference ID, ki omogočajo enolično identifikacijo vsake lokacije v nacionalnem omrežju.

3.2 Arhitektura sistema

Sistem bo implementiran po ETL principu, pri čemer bomo dodali še vmesno staging fazo v oblaku za večjo zanesljivost in sledljivost procesov. [14] V prvi fazi ekstrakcije uporabljamo Python skripte s Selenium avtomatizacijo, ki se povežejo na spletno stran National Grid in prenesejo CSV datoteke. Ta del deluje skoraj kot pravi uporabnik, saj simulira klike, čaka na nalaganje dinamičnih elementov in uporablja vse interaktivne komponente na strani. Ker gre za kompleksno spletno aplikacijo z JavaScript generiranimi elementi, je ta pristop nujen. Surovi podatki nato ne gredo direktno v bazo, ampak jih najprej pošljemo v Google Cloud Storage, kjer poteka vmesna transformacija. V tej staging fazi naredimo osnovno validacijo podatkov, počistimo manjkajoče vrednosti in standardiziramo formate. Te delno transformirane podatke shranimo kot staging datoteke, kar nam omogoča, da se lahko kadarkoli vrnemo nazaj in pogledamo, kako so podatki izgledali v določenem trenutku. To je zelo koristno, če naletimo na kakšne težave ali potrebujemo ponovno procesiranje. Šele nato pride na vrsto končna transformacija, ki jo izvajamo s Pandas knjižnico. Ko je vse pripravljeno, podatke naložimo v PostgreSQL podatkovno bazo, kjer so nato na voljo za analizo in uporabo. Ta pristop z vmesno staging fazo v GCP nam zagotavlja večjo zanesljivost celotnega sistema. Če kdaj pride do napake v katerikoli fazi, imamo vedno shranjene vmesne rezultate in lahko proces ponovno poženemo od točke, kjer se je nekaj zalomilo. To je še posebej pomembno pri avtomatiziranih sistemih, kjer ni vedno nekoga, ki bi takoj opazil težavo. Celoten proces bo seveda avtomatiziran, uporabili bomo Google Cloud Scheduler, ki bo skripte zaganjal periodično po vnaprej določenem urniku. Tako bo sistem deloval samostojno in podatki se bodo osvežili brez kakršnegakoli ročnega posredovanja.

3.3 Izbira orodij

Za implementacijo sistema smo izbrali Python 3.11, predvsem zaradi njegovega bogatega ekosistema knjižnic in odlične podpore za avtomatizacijo.

Python se je v zadnjih letih uveljavil kot eden vodilnih jezikov za delo s podatki, kar se odraža tudi v razpoložljivosti kvalitetnih orodij za naše potrebe. Pri izbiri knjižnic smo kombinirali preverjene standardne rešitve s prilagojenimi komponentami. Za interakcijo z brskalnikom uporabljamo Selenium v kombinaciji z undetected-chromedriver, ki nam omogoča upravljanje z brskalnikom na način, da se izognemo detekciji avtomatizacije. To je zelo pomebno, saj večina sodobnih spletnih platform implementira zaščitne mehanizme proti avtomatiziranim dostopom. [3] Za delo s podatki se zanašamo na Pandas, ki je praktično postal standard za branje, transformacijo in obdelavo tabelaričnih podatkov v Pythonu. Ta knjižnica nam omogoča učinkovito delo z CSV datotekami in izvajanje kompleksnih transformacij podatkov. Za komunikacijo z Google Cloud Storage uporabljamo uradno google-cloud-storage knjižnico, ki poskrbi za vso interakcijo s cloudnim shranjevanjem in upravljanje staging datotek. Za beleženje vseh dogodkov in napak uporabljamo vgrajeni logging modul, ki nam omogoča strukturirano spremljanje delovanja sistema. Poleg standardnih knjižnic smo razvili tudi nekaj prilagojenih komponent. **AbstractScriptRunner** je abstraktni razred, ki standardizira izvajanje ETL skript in vključuje vgrajeno logiko za elegantno obravnavo napak. Tako vse naše skripte sledijo enakemu vzorcu izvajanja, kar olajša vzdrževanje in razumevanje kode. Centralizirano konfiguracijo celotnega sistema upravljamo preko config modula, ki vključuje nastavitve za logiranje, povezavo z GCS klientom in podatkovne direktorije. Tako imamo vse nastavitve na enem mestu, kar občutno olajša prilagajanje sistema različnim okoljem. Razvili smo tudi gdutil (Generic Dataset Utilities), nabor prilagojenih funkcij za delo z geografskimi podatkovnimi nizi, ki rešujejo specifične izzive našega projekta.

3.3.1 Infrastruktura

Celoten sistem temelji na kombinaciji oblačnih storitev in podatkovne baze, kar nam omogoča fleksibilnost pri shranjevanju in obdelavi podatkov. Za shranjevanje surovih in delno transformiranih datotek uporabljamo Google

Cloud Storage. To je objektno shranjevanje, ki ga ponuja Google Cloud Platform in se je izkazalo za idealno rešitev za naš staging layer. Tukaj se shranjujejo CSV datoteke, ki jih sistem prenese iz National Grid platforme, še preden jih procesiramo in naložimo v bazo. Prednost GCS je v tem, da nam omogoča praktično "neomejenoshranjevanje po relativno nizki ceni, hkrati pa so podatki vedno dostopni in zanesljivo shranjeni. Poleg tega lahko kadarkoli dostopamo do zgodovinskih verzij datotek, če potrebujemo ponovno procesiranje ali analizo, kako so se podatki spreminjali skozi čas. Za končno shranjevanje strukturiranih, obdelanih podatkov pa uporabljamo PostgreSQL podatkovno bazo. PostgreSQL smo izbrali zaradi njegove robustnosti, odlične podpore za kompleksne poizvedbe in geografske podatke preko PostGIS razširitve. Gre za relacijsko bazo, ki omogoča učinkovito indeksiranje in iskanje po podatkih, kar je ključno za kasnejšo analizo in vizualizacijo. Podatki v PostgreSQL so strukturirani v tabele z jasno definiranimi relacijami, kar olajša delo z njimi in zagotavlja integriteto podatkov. Za avtomatizacijo celotnega procesa skrbi Google Cloud Scheduler. To je cron storitev v oblaku, ki omogoča zanesljivo periodično izvajanje naših skript. Nastavimo lahko natančne urnike, kdaj naj se sistem zažene (vsak dan ob določeni uri ali vsak teden v določen dan). Cloud Scheduler je zanesljiv, ne zahteva vzdrževanja strežnika, ki bi moral biti vedno prižgan, in nam pošlje obvestila, če pride do napak pri izvajanju. Tako je celoten ETL proces popolnoma avtomatiziran in deluje brez potrebe po ročnem posredovanju.

3.4 Kontrola kakovosti podatkov

Za zagotavljanje zanesljivosti sistema in kakovosti podatkov bomo implementirali več nivojev preverjanj, ki bodo našli potencialne težave že v zgodnjih fazah procesiranja. Naslednja pomembna kontrola je detekcija podvojenih vnosov. Ker sistem deluje periodično in prenašamo podatke redno, obstaja možnost, da bi določeni podatki bili v bazo naloženi večkrat. Sistem bo zato preverjal, ali vnos s kombinacijo ključnih atributov že obstaja v bazi, preden

ga poskuša vstaviti. Tako preprečimo nepotrebno podvajanje in zagotovimo integriteto podatkov. Validacija podatkovnih tipov je ključna za pravilno delovanje celotnega sistema. Sistem bo preveril, ali so numerične vrednosti res številke, ali so datumi v pravilnem formatu in ali besedilna polja ne vsebujejo nepričakovanih znakov ali vsebin. Če naleti na vrednosti, ki ne ustrezajo pričakovanemu tipu, bo zabeležil opozorilo in se odločil, ali lahko vrednost pretvori ali jo mora zavrniti. Posebno pozornost bomo namenili tudi preverjanju in upravljanju manjkajočih vrednosti. Sistem bo identificiral, kateri stolpci imajo manjkajoče podatke in glede na pomembnost polja odločil, kako ravnati. Pri nekritičnih poljih lahko manjkajoče vrednosti nadomestimo z privzetimi vrednostmi ali jih pustimo prazne, medtem ko bodo pri kritičnih poljih manjkajoče vrednosti povzročile zavrnitev celotnega vnosa. Vse te odločitve bodo jasno dokumentirane v logih za kasnejši pregled.

3.5 Podatkovne baze in pogoni s podporo za grafe

Z naraščajočo kompleksnostjo podatkov in potrebo po učinkovitem modeliranju odnosov med entitetami so grafno usmerjene podatkovne rešitve postale pomemben del sodobnih podatkovnih arhitektur. Sistemi, ki podpirajo grafe, omogočajo učinkovite večkorakovne poizvedbe, analizo omrežij in modeliranje kompleksnih relacij, kar je pogosto neučinkovito ali neizvedljivo v klasičnih relacijskih bazah. V nadaljevanju so predstavljeni izbrani sistemi, ki bodisi delujejo kot namenske grafne baze bodisi kot razširitve ali poizvedovalni pogoni nad obstoječimi podatkovnimi viri.

3.5.1 PuppyGraph

PuppyGraph ni klasična grafna baza podatkov, temveč *grafni poizvedovalni pogon*, ki omogoča izvajanje grafnih poizvedb neposredno nad obstoječimi podatkovnimi viri, kot so relacijske baze, podatkovna skladišča ali podat-

kovna jezera. Ključna značilnost sistema je arhitektura brez ETL procesov, saj podatkov ni treba kopirati ali transformirati v ločeno grafno shrambo.

PuppyGraph logično preslika obstoječe podatke v grafovni model in podpira standardne grafne poizvedovalne jezike, kot sta openCypher in Gremlin. Tak pristop omogoča hitro uvedbo grafnih analiz v obstoječe sisteme, pri čemer se ohranijo prednosti primarnega podatkovnega vira, kot so konsistentnost, varnost in upravljanje podatkov.

3.5.2 AgensGraph

AgensGraph je grafna baza podatkov, ki temelji na relacijski bazi PostgreSQL. Gre za hibridni sistem, ki združuje relacijski in grafni podatkovni model znotraj enotnega podatkovnega strežnika. Uporabnikom omogoča, da v isti podatkovni bazi kombinirajo klasične SQL poizvedbe in grafne operacije.

Zaradi izpeljave iz PostgreSQL AgensGraph podeduje transakcijski model ACID, zanesljivost ter bogat ekosistem orodij. Takšna zasnova je primerna za primere uporabe, kjer je potrebno tesno prepletanje relacijskih in grafnih podatkov brez uvajanja dodatne infrastrukture.

3.5.3 RedisGraph

RedisGraph je modul za Redis, ki implementira lastnostni grafni model v pomnilniku. Namenjen je predvsem scenarijem, kjer je ključnega pomena nizka latenca in visoka hitrost poizvedb. Sistem uporablja openCypher kot poizvedovalni jezik ter interno predstavlja graf z uporabo redkih matrik in linearno-algebrskih operacij.

Zaradi in-memory narave je RedisGraph posebej primeren za operativne in realnočasovne grafne primere uporabe, kot so priporočilni sistemi ali analiza omrežij v živo. Slabost pristopa je večja poraba pomnilnika in omejena primernost za zelo velike, trajne grafe.

3.5.4 Cayley

Cayley je odprtokodna grafna baza podatkov, prvotno zasnovana po vzoru grafnega sistema Freebase. Poseben poudarek namenja podpori za povezane podatke in ogrodja za opis virov (RDF standarde).

Sistem podpira več poizvedovalnih jezikov in lahko deluje nad različnimi hrambnimi mehanizmi, vključno z relacijskimi in ključ-vrednost bazami. Zaradi svoje prilagodljivosti je Cayley primeren za raziskovalne namene, semantične grafe in znanstvene aplikacije, kjer interoperabilnost podatkov igra pomembno vlogo.

3.5.5 Apache AGE

Apache AGE (*A Graph Extension*) je razširitev za PostgreSQL, ki tej relacijski bazi doda podporo za grafni podatkovni model. Namesto ločene grafne baze AGE omogoča shranjevanje vozlišč in povezav znotraj PostgreSQL ter poizvedovanje z uporabo openCypher jezika.

Tak pristop omogoča hibridno uporabo SQL in grafnih poizvedb nad enotnim podatkovnim skladiščem. Apache AGE je še posebej primeren za organizacije, ki že uporabljajo PostgreSQL in želijo grafne funkcionalnosti dodati brez večjih arhitekturnih sprememb.

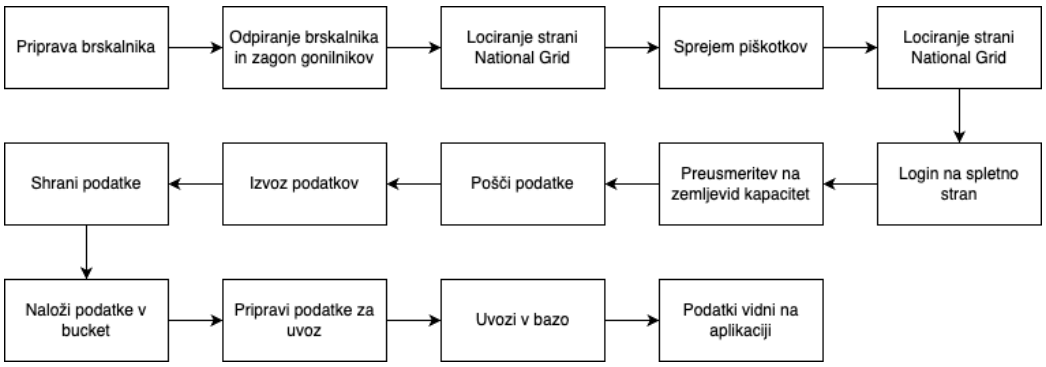
3.6 Izbira tehnologije

Glede na to, da je v obravnavanem primeru podatkovna baza že vzpostavljena v sistemu PostgreSQL, se kot najprimernejša izbira izkaže Apache AGE. Razširitev omogoča enostavno integracijo grafnega podatkovnega modela neposredno v obstoječo bazo, brez potrebe po migraciji podatkov ali uvajanju ločenega grafnega strežnika. Apache AGE podpira poizvedovalni jezik openCypher, ki je postal standard za delo z lastnostnimi grafi, kar poenostavi izražanje kompleksnih relacijskih vzorcev in večkorakovnih poizvedb. Poleg tega AGE izkorišča preverjen transakcijski mehanizem PostgreSQL,

zagotavlja ACID lastnosti ter omogoča sočasno uporabo SQL in grafnih poizvedb nad istimi podatki. Takšna hibridna zasnova zmanjšuje arhitekturno kompleksnost sistema, poenostavi vzdrževanje in omogoča postopno uvajanje grafnih pristopov v obstoječe relacijsko okolje.

Poglavje 4

Implementacija sistema



Slika 4.1: Postopek za strganje podatkov, odlaganje v cloud, obdelavo ter uvoz v bazo.

4.1 Koraki delovanja sistema

4.1.1 Priprava brskalnika in zagon gonilnikov

Prvi korak implementacije vključuje konfiguracijo Selenium WebDriver z uporabo razreda Options. Sistem inicializira Chrome brskalnik v headless načinu, pri čemer so dodani argumenti `--no-sandbox, --disable-dev-shm-usage in --window-size=3840,2160`. Argument `--no-sandbox` omogoča delovanje brskalnika brez Chrome-ovega sandboxing mehanizma, kar je pogosto nujno v kontejneriziranih okoljih, kjer sandbox lahko povzroča konflikt z omejenimi sistemskimi privilegiji. Argument `--disable-dev-shm-usage` preusmeri uporabo deljenega pomnilnika s `/dev/shm` na disk, kar preprečuje napake v okoljih z omejenim ali premajhnim deljenim pomnilnikom (npr. Docker). Določitev velikosti okna zagotavlja pravilno renderiranje strani tudi v headless načinu. Brskalnik se inicializira preko `webdriver.Chrome(options=options)`, medtem ko WebDriverWait skrbi za zanesljivo upravljanje nalaganja dinamičnih elementov.

```
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import
   expected_conditions as EC
from google.cloud import storage

options = Options()
options.add_argument( --headless )
options.add_argument( --no-sandbox )
options.add_argument( --disable-dev-shm-usage )
options.add_argument( --window-size=3840,2160 )

driver = webdriver.Chrome(options=options)
```

Listing 4.1: Priprava brskalnika

4.1.2 Odpiranje brskalnika in zagon gonilnikov

Ko je brskalnik inicializiran, se izvede navigacija na glavni portal National Grid (`https://www.nationalgrid.co.uk/network-opportunity-map/`). Sistem počaka 5 sekund za popolno nalaganje strani, skripta uporablja čakanje z metodo *.sleep()* za zagotovitev, da so vsi elementi pripravljeni za interakcijo.

```
1    driver.get( https://www.nationalgrid.co.uk/network-
        opportunity-map/ )
2    time.sleep(5)
```

Listing 4.2: Odpiranje strani

4.1.3 Lociranje strani National Grid

Po uspešni inicializaciji sistem poišče in klikne na povezavo za prijavo, ki vodi na prijavni portal. Skripta uporablja CSS selektorje za natančno lociranje elementov, pri čemer se navigacijska logika prilagaja morebitnim spremembam v strukturi strani. Sistem beleži vsak korak navigacije v log datoteko za kasnejšo analizo in odpravljanje težav.

4.1.4 Sprejem piškotkov

Upravljanje s piškotki je izvedeno neposredno z iskanjem gumba za sprejem vseh opcijskih piškotkov. Sistem z uporabo `WebDriverWait` in pogoja `element_to_be_clickable` poišče element *Accept all optional cookies* ter ga, ko je dostopen, klikne. Tak pristop zagotavlja, da se klik izvede šele, ko je gumb dejansko interaktiven. Ker koda ne vključuje dodatnega preverjanja ali try-except bloka, se predpostavlja, da je banner vedno prisoten; v primeru manjkajočega elementa bi se sprožila izjema.

```
1 cookie_btn = wait.until(EC.element_to_be_clickable(
2         (By.CSS_SELECTOR, 'a[title=Accept all optional
             cookies]')))
3 cookie_btn.click()
```

Listing 4.3: Potrjevanje piškotkov

4.1.5 Lociranje strani National Grid (navigacija)

Navigacija do podatkovnega portala poteka v več korakih. Po prijavi sistem navigira na specifični URL zemljevida kapacitet

`(/our-network/network-capacity-map-application)`. Sistem počaka na popolno nalaganje aplikacije, preden nadaljuje z naslednjimi koraki. Blokirati pa je potrebno nalaganje zemljevida in vseh povezav, ki se na njem prikazujejo, v nasprotnem primeru se okno brskalnika poruši.

```
driver.execute_cdp_cmd( Network.enable , {})
driver.execute_cdp_cmd( Network.setBlockedURLs , {
     urls : [
          *mapbox.com/* ,
          *tiles.mapbox.com/* ,
          *tile.openstreetmap.org/* ,
          *tilelayer* ,
          *VectorTile* ,
          *features* ,
          *geojson*
     ]
})
```

Listing 4.4: Potrjevanje piškotkov

4.1.6 Login na spletno stran

Email in geslo se vneseta v ustrezna polja z ID-ji `customer-portal-form-field_-_emailAddress` in `customer-portal-form-field__password`. Skripta uporablja JavaScript executor za zanesljiv klik na prijavni gumb, kar obvladuje tudi primere, ko standardni Selenium klik ne deluje. Po prijavi sistem počaka in preveri URL za potrditev uspešne prijave. Pred vsakim vpisovanjem v polje email ali password pokličemo metodo `.clear()`, da je polje zagotovo prazno.

```
login_btn = wait.until(EC.element_to_be_clickable((By.
    CSS_SELECTOR, 'a[href^=/customer-portal/login]')))
driver.execute_script( arguments[0].click(); , login_btn
    )
time.sleep(3)

email_input = wait.until(EC.
    visibility_of_element_located((By.ID,  customer-
    portal-form-field__emailAddress )))
password_input = wait.until(EC.
    visibility_of_element_located((By.ID,  customer-
    portal-form-field__password )))

email_input.clear()
email_input.send_keys(EMAIL)
password_input.clear()
password_input.send_keys(PASSWORD)
```

Listing 4.5: Login

4.1.7 Preusmeritev na zemljevid kapacitet

Po uspešni prijavi se izvede navigacija na aplikacijo zemljevida kapacitet. Tukaj je dejanski zemljevid blokiran saj headless browser ne more naložiti zemljevida samega. Sistem najprej sprejme pogoje uporabe s klikom na consent checkbox in potrditvenim gumbom. Nato odpre levi navigacijski panel in klikne na zavihek "Data", kjer so dostopni izvozni podatki. Vsak korak vključuje preverjanje prisotnosti elementov in ustrezno obravnavo napak.

```
1    driver.get( https://www.nationalgrid.co.uk/our-
       network/network-capacity-map-application )
2    wait.until(lambda d: d.execute_script( return
       document.readyState ) ==  complete )
```

Listing 4.6: Lociranje zemljevida

4.1.8 Pošči podatke

Izvoz podatkov se sproži preko postopnega odpiranja ustreznih uporabniških vmesnikov. Najprej sistem s pomočjo objekta `WebDriverWait` počaka, da je element z identifikatorjem `data-pill` ključen za nadaljevanje interakcije, nakar se klik izvede preko `execute_script`, kar zagotavlja zanesljivo aktivacijo tudi v primerih, ko standardni klik ni zadosten. Sledi aktivacija gumba za odprtje stranske vrstice, izbranega preko CSS selektorja. Ko je stranska vrstica uspešno odprta, sistem poišče oznako `Data` z uporabo XPATH izraza. Pred klikanjem se element premakne v vidno območje z uporabo `scrollIntoView(true)`, kar zagotavlja zanesljivo interakcijo tudi v *headless* načinu. S tem je uporabniški vmesnik ustrezno pripravljen za nadaljnje korake izvoza podatkov.

```
data_button = wait.until(EC.element_to_be_clickable((By.
    ID, "data-pill")))
driver.execute_script("arguments[0].click();",
    data_button)
open_sidebar_btn = wait.until(EC.element_to_be_clickable
    ((
            By.CSS_SELECTOR,
             "button.btn.btn--continue.btn--default.btn--
                small"
        )))
        open_sidebar_btn.click()

data_label = wait.until(EC.element_to_be_clickable(
            (By.XPATH, "//label[contains(text(), 'Data')
                ]")))
        driver.execute_script("arguments[0].
            scrollIntoView(true);", data_label)
        data_label.click()
```

Listing 4.7: Podatki

4.1.9 Izvoz podatkov

Prenos CSV datoteke se sproži s klikom na gumb za izvoz, izbran preko CSS selektorja. Klik na gumb se izvede preko `execute_script`, kar omogoča zanesljivo sprožitev dogodka tudi v primerih, ko standardni klik ni zadosten. Po sprožitvi izvoza sistem z uporabo `WebDriverWait` preveri, da element z razredom `btn--loading` izgine, kar označuje konec procesa generiranja CSV datoteke. Ta mehanizem omogoča deterministično zaznavanje zaključka izvoza.

```
export_button = wait.until(EC.element_to_be_clickable(
        (By.CSS_SELECTOR, "button.btn.btn--primary.
           export-button")))
    driver.execute_script("arguments[0].scrollIntoView(
       true);", export_button)
    time.sleep(0.5)
    driver.execute_script("arguments[0].click();",
       export_button)

    WebDriverWait(driver, 30).until_not(
        EC.presence_of_element_located(
            (By.CSS_SELECTOR, "button.btn--primary.
               export-button.btn--loading")
        )
    )
```

Listing 4.8: Izvoz

4.1.10 Shrani podatke in naloži podatke v bucket

Podatke je za nadaljno uporabo potrebno shraniti v Google Cloud Storage bucket. Datoteke se organizirajo v mapno strukturo po datumih (leto/mesec/dan) za lažje upravljanje. GCS zagotavlja verzioniranje, kar omogoča dostop do vseh zgodovinskih verzij podatkov.

```
    timestamp = datetime.now(timezone.utc).strftime( %Y
        -%m-%d_%H-%M-%S )
    upload_to_gcs(
        bucket_name= diplomska-461311_cloudbuild ,
        source_file=full_path,
        destination_blob= exports/
            wpd_network_capacity_map_{}.csv .format(
            timestamp)
        )
```

Listing 4.9: Nalaganje v GCS

4.1.11 Pripravi podatke za uvoz

V tej fazi se izvede transformacija podatkov s Pandas knjižnico. CSV datoteka se naloži v GeoDataFrame, kjer se izvedejo naslednje operacije: odstranjevanje praznih vrstic, standardizacija imen stolpcev, pretvorba podatkovnih tipov, validacija vrednosti Demand Headroom in pretvorba koordinat v standardni format. Dodajo se tudi metapodatki o času izvoza in verziji podatkov.

```
gdf = dl_reader.read_csv(
    cls.dl_all_path,
    crs= epsg:4326 ,
    xcol= Longitude ,
    ycol= Latitude ,
    use_saved=True,
)
columns = {
     Substation Name :  name ,
     Bulk Supply Point Name :  bsp ,
     Substation Number :  xref ,
     Upstream Voltage :  up ,
     Downstream Voltage :  down ,
     Demand Headroom (MVA) :  demand_headroom_mva ,
     Generation Headroom (MVA) :
       generation_headroom_mva ,
     Fault Level Headroom (kA) :
       fault_level_headroom_ka ,
}
gdf = gdf[list(columns.keys())]
gdf.rename(columns=columns, inplace=True)
gdf[ name ] = (
    gdf[ name ].str.extract( ^(.+?)(?=\s+\d| \(\d|\s-\s|
       $) )[0].str.title()
)
gdf[ bsp ] = (
    gdf[ bsp ].str.extract( ^(.+?)(?=\s+\d| \(\d|\s-\s|$
       ) )[0].str.title()
)
gdf[ last_updated ] = str(cls.all_last_updated)
gdf.drop_duplicates(inplace=True)
```

Listing 4.10: Priprava podatkov

4.1.12 Priprava baze

Pred vsakim uvozom podatkov je potrebno ustvariti novo tabelo saj tako lažje poskrbimo za verzionizacijo podatkov. V SQL se uporabljajo stavki `CREATE TABLE`, za boljšo performanco in hitrost pa ustvarimo tudi index na tabeli.

```
CREATE TABLE IF NOT EXISTS uk_dataset. core/ng/
   substation/bsp/v2025_10
(
    id integer NOT NULL DEFAULT nextval('uk_dataset .
       core/ng/substation/bsp/v2025_10_id_seq '::
       regclass),
    geometry geometry(Geometry,4326) NOT NULL,
    geometry_projected geometry(Geometry,27700) NOT NULL
       ,
    properties jsonb DEFAULT '{}'::jsonb,
    xref character varying COLLATE pg_catalog. default ,
    CONSTRAINT  core/ng/substation/bsp/v2025_10_pkey
       PRIMARY KEY (id),
    CONSTRAINT  core/ng/substation/bsp/v2025_10_xref_key
         UNIQUE (xref)
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS uk_dataset. core/ng/substation/bsp
   /v2025_10
    OWNER to dataset_user;
-- Index: core/ng/substation/bsp/v2025_10_geometry

-- DROP INDEX IF EXISTS uk_dataset. core/ng/substation/
   bsp/v2025_10_geometry ;

CREATE INDEX IF NOT EXISTS  core/ng/substation/bsp/
```

```
    v2025_10_geometry
20    ON uk_dataset. core/ng/substation/bsp/v2025_10
          USING gist
21    (geometry)
22    TABLESPACE pg_default;
23 -- Index: core/ng/substation/bsp/
    v2025_10_geometry_projected
24
25 -- DROP INDEX IF EXISTS uk_dataset. core/ng/substation/
    bsp/v2025_10_geometry_projected ;
26
27 CREATE INDEX IF NOT EXISTS  core/ng/substation/bsp/
    v2025_10_geometry_projected
28    ON uk_dataset. core/ng/substation/bsp/v2025_10
          USING gist
29    (geometry_projected)
30    TABLESPACE pg_default;
31 -- Index: core/ng/substation/bsp/v2025_10_xref
32
33 -- DROP INDEX IF EXISTS uk_dataset. core/ng/substation/
    bsp/v2025_10_xref ;
34
35 CREATE INDEX IF NOT EXISTS  core/ng/substation/bsp/
    v2025_10_xref
36    ON uk_dataset. core/ng/substation/bsp/v2025_10
          USING btree
37    (xref COLLATE pg_catalog. default  ASC NULLS LAST)
38    TABLESPACE pg_default;
```

Listing 4.11: Priprava baze

4.1.13 Uvoz v bazo

Transformirani podatki se naložijo v PostgreSQL bazo. Implementirana je transakcijska logika, ki zagotavlja atomskost operacij, tako se vsi podatki se vnesejo ali pa se celotna transakcija razveljavi.

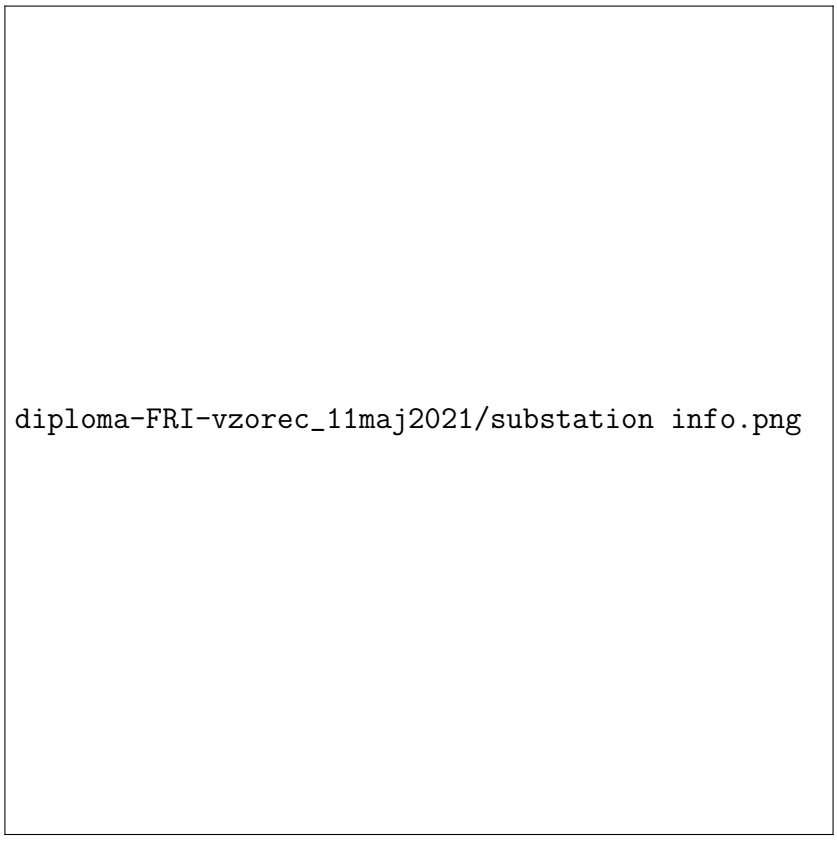
```
for taxonomy in [ core.ng.substation.bsp ,  core.ng.
    substation.pss ]:
            table_name = make_table_name(taxonomy)
            con.execute(
                f
                UPDATE dataset. {table_name}
                SET properties = (properties - '
                    demand_headroom_mva') ||
                    jsonb_build_object('dhr', (properties
                     -> 'demand_headroom_mva')::float)
                WHERE properties ->> '
                    demand_headroom_mva' IS NOT NULL;

            )
            add_leaf_dataset(taxonomy, con)
```

Listing 4.12: Uvoz v bazo

4.1.14 Podatki vidni na aplikaciji

Podatki, ki so rezultat celotnega obdelovalnega procesa, so po zaključku vseh validacijskih in objavnih korakov neposredno vidni v aplikaciji. To pomeni, da se ob vsakem uspešnem zagonu sistema najnovejši, preverjeni in standardizirani podatki samodejno posodobijo v uporabniškem vmesniku, kjer jih lahko končni uporabniki takoj pregledajo, filtrirajo in uporabljajo za nadaljnje analize ali operativne odločitve. Dostopni so v realnem času, prek interaktivnih preglednic, kartografskih prikazov ali dinamičnih vizualizacij, odvisno od funkcionalnosti posamezne aplikacijske komponente. S tem se za-

gotavlja popolna transparentnost med procesom zajema podatkov in njihovo končno uporabo, saj aplikacija vedno prikazuje zadnjo potrjeno verzijo informacij, sinhronizirano z osrednjo bazo. Takšna integracija omogoča enoten vpogled v stanje omrežja, objektov in procesov, ne glede na izvor podatkov ali njihovo tehnično kompleksnost v ozadju.



Slika 4.2: Primer iz UI aplikacije

4.2 Pridobivanje in shranjevanje podatkov

Proces pridobivanja in shranjevanja podatkov je zasnovan kot popolnoma avtomatiziran sistem, ki zagotavlja zanesljivo, ponovljivo in časovno sledljivo osveževanje informacij iz zunanjih virov. Implementirana Python skripta

uporablja knjižnico Selenium za dinamično upravljanje z brskalnikom in simulacijo interakcije uporabnika s spletno stranjo National Grid.

Po uspešnem prenosu se datoteke avtomatsko naložijo v namenski Google Cloud Storage bucket, kjer se hranijo v strukturirani mapni hierarhiji po letu, mesecu in dnevu prenosa. Ta organizacija omogoča enostavno arhiviranje, hitro iskanje in učinkovito upravljanje zgodovinskih podatkovnih posnetkov. GCS infrastruktura omogoča tudi vklop verzioniranja, kar pomeni, da se ob vsakem novem prenosu ohrani popolna zgodovina sprememb, tako se stare datoteke ne prepišejo, temveč ostanejo dostopne za primerjalne analize ali rekonstrukcijo prejšnjih stanj. [12]

4.3 Obdelava podatkov

Za obdelavo prenesenih Excel datotek je bila razvita namensko prilagojena Python skripta, ki temelji na uporabi knjižnice `pandas`. Skripta po vzpostavitvi povezave z Google Cloud Storage najprej prebere ustrezne datoteke, shranjene v oblaku, ter jih pretvori v podatkovni okvir GeoDataFrame) za nadaljnjo obdelavo. V naslednjem koraku se izvede sistematično čiščenje podatkov, ki vključuje odstranjevanje praznih vrstic, podvajanj in morebitnih nekonsistentnih zapisov, hkrati pa se standardizirajo imena stolpcev in podatkovni formati, kar omogoča enotno strukturo za vse nadaljnje faze obdelave. Po čiščenju skripta izvede validacijo osnovnih vrednosti in preveri skladnost tipov podatkov z zahtevanimi shemami v ciljni podatkovni bazi. Tako pripravljeni podatki se nato dopolnijo z dodatnimi metapodatki, kot so datum izvoza, verzija podatkov in identifikacijska oznaka vira. Končni rezultat tega postopka je homogen in validiran podatkovni nabor, pripravljen za nadaljnje nalaganje v podatkovno bazo, kjer se lahko uporablja za analitične ali operativne namene.

4.4 Avtomatizacija in nadzor

Celoten proces pridobivanja, obdelave in nalaganja podatkov v podatkovno bazo je popolnoma avtomatiziran s pomočjo orodja Google Cloud Scheduler, ki skrbi za redno in zanesljivo izvajanje cevovoda brez ročnega posredovanja. Scheduler je konfiguriran tako, da se skripta samodejno zažene vsako polno uro med 6. in 22. uro, kar zagotavlja sprotno osveževanje podatkovnih virov in ažurnost prikazanih rezultatov v aplikaciji. Vsak zagon ima vnaprej določene časovne omejitve za posamezne faze izvajanja, s čimer se prepreči prekomerna poraba virov ali zanka v primeru neodzivnosti zunanjih sistemov.

Za spremljanje delovanja so vpeljani mehanizmi nadzora, ki vključujejo podrobno beleženje vseh dogodkov in rezultatov izvajanja v Cloud Logging, kar omogoča sprotno analizo in zgodovinski vpogled v izvajanje procesa.

4.5 Kontrola kakovosti podatkov

Za zagotavljanje zanesljivosti sistema in kakovosti podatkov bomo implementirali več nivojev preverjanj, ki bodo našli potencialne težave že v zgodnjih fazah procesiranja. Naslednja pomembna kontrola je detekcija podvojenih vnosov. Ker sistem deluje periodično in prenašamo podatke redno, obstaja možnost, da bi določeni podatki bili v bazo naloženi večkrat. Sistem bo zato preverjal, ali vnos s kombinacijo ključnih atributov že obstaja v bazi, preden ga poskuša vstaviti. Tako preprečimo nepotrebno podvajanje in zagotovimo integriteto podatkov. [2] Validacija podatkovnih tipov je ključna za pravilno delovanje celotnega sistema. Sistem bo preveril, ali so numerične vrednosti res številke, ali so datumi v pravilnem formatu in ali besedilna polja ne vsebujejo nepričakovanih znakov ali vsebin. Če naleti na vrednosti, ki ne ustrezajo pričakovanemu tipu, bo zabeležil opozorilo in se odločil, ali lahko vrednost pretvori ali jo mora zavrniti. [6] Posebno pozornost bomo namenili tudi preverjanju in upravljanju manjkajočih vrednosti. Sistem bo identificiral, kateri stolpci imajo manjkajoče podatke in glede na pomembnost polja odločil, kako ravnati. Pri nekritičnih poljih lahko manjkajoče vrednosti nado-

mestimo z privzetimi vrednostmi ali jih pustimo prazne, medtem ko bodo pri kritičnih poljih manjkajoče vrednosti povzročile zavrnitev celotnega vnosa. Vse te odločitve bodo jasno dokumentirane v logih za kasnejši pregled.

Poglavje 5

Rezultati in evalvacija

5.1 Merila uspešnosti

Uspešnost sistema bomo vrednotili preko dveh dimenzij. Učinkovitost bo merjena s časom izvajanja celotnega ETL cikla, ki mora biti zaključen v manj kot 5 minutah, ter s popolno eliminacijo trenutnih 2-4 ur mesečnega ročnega dela. Zanesljivost sistema bo ocenjena preko uspešnosti mesečnih izvajanj, kjer pričakujemo najmanj 95% uspešnih procesiranj.

5.2 Način evalvacije

Za preverjanje popolnosti in pravilnosti podatkov bomo redno primerjali število zapisov v bazi s številom zapisov v vhodni datoteki ter preverjali, ali se posamezni podatki med seboj ujemajo. Dodatno bomo vse teste izvajali tudi v ločenem testnem okolju baze, kjer bomo preverili, ali so vsi podatki pravilno vnešeni, dosegljivi in konsistentni z vhodnimi datotekami.

5.3 Kriteriji uspeha

Sistem bo ocenjen kot uspešen, če bo dosegel zastavljene pragove na vseh ključnih metrikah. Povprečen čas izvajanja mora biti konsistentno pod 5 mi-

nut, z najmanj 95% uspešnih izvajanj v produkcijskem obdobju. V podatkih ne sme biti manjkajočih vrednosti za kritična polja. Dodatno mora sistem omogočati popolno sledljivost s shranjevanjem vseh vmesnih rezultatov v staging okolju.

5.4 Rezultati

Na podlagi implementacije in izvedenih funkcionalnih ter validacijskih testov lahko z potrdimo, da je sistem v celoti dosegel in presegel zastavljene cilje. Uvedena popolna avtomatizacija procesa je uspešno odpravila potrebo po mesečnem ročnem upravljanju in preverjanju podatkov, s čimer se je dosegla neposredna časovna optimizacija v obsegu približno 2–4 ur kvalificiranega dela na mesec. Ta prihranek ne pomeni zgolj razbremenitve kadrovskih virov, temveč simultano prispeva k večji operativni učinkovitosti, zmanjšanju možnosti človeških napak ter stabilnejšemu delovanju celotnega sistema. Vzpostavljena staging arhitektura v okolju Google Cloud Storage se je izkazala kot zanesljiva in fleksibilna rešitev, ki omogoča varno shranjevanje različnih verzij podatkov ter njihovo ponovno obdelavo v katerikoli fazi življenjskega cikla. Ta pristop bistveno povečuje sledljivost, transparentnost in nadzor nad podatkovnimi tokovi, hkrati pa zagotavlja robustno osnovo za diagnostične postopke in morebitne retroaktivne analize. Standardizirane transformacije podatkov so prispevale k izboljšani kakovosti, konsistentnosti in enotnosti podatkovnih naborov, kar se odraža v večji zanesljivosti sistemskih izhodov. Dolgoročno uspešna implementacija jasno nakazuje visok potencial za razširitev sistema tudi na druge operaterje električne infrastrukture po Združenem kraljestvu. S tem se vzpostavlja trdna podlaga za razvoj celovitega, razširljivega in trajnostnega sistema za spremljanje ter analizo električne infrastrukture na nacionalni ravni, kar predstavlja korak k bolj učinkovitemu upravljanju energetskih virov in podpori strateškemu načrtovanju v energetskem sektorju.

Poglavje 6

Zaključek

6.1 Zaključki

Predstavljena diplomska naloga obravnava razvoj avtomatiziranega sistema za pridobivanje in obdelavo podatkov o električni infrastrukturi iz platforme National Grid. Sistem uspešno rešuje problem zamudnega ročnega pridobivanja podatkov z implementacijo robustnega ETL procesa, ki temelji na Python skriptah, Google Cloud Storage staging okolju in PostgreSQL podatkovni bazi. Razvita rešitev izpolnjuje vse zastavljene cilje. Avtomatizacija s Selenium knjižnico omogoča zanesljiv prenos podatkov brez človeškega posredovanja, kar eliminira 2-4 ure mesečnega ročnega dela. Implementacija staging faze v GCS zagotavlja popolno sledljivost in možnost ponovne obdelave podatkov, konsistentnost in kakovost podatkov. Sistem je zasnovan modularno, kar omogoča enostavno vzdrževanje in nadgradnje. Ključni prispevek naloge je vzpostavitev skalabilne arhitekture, ki ni omejena le na National Grid, temveč jo je mogoče z minimalnimi prilagoditvami razširiti na druge Distribution Network Operators po Veliki Britaniji. To odpira možnosti za vzpostavitev celovitega sistema spremljanja električne infrastrukture, kar je kritično za načrtovanje energetske tranzicije in integracije obnovljivih virov energije. Praktična vrednost sistema se kaže v takojšnjem dostopu do ažurnih podatkov o razpoložljivih kapacitetah (Demand Headroom), kar omogoča hi-

trejše in bolj informirane odločitve pri načrtovanju novih projektov. Energetska podjetja, razvijalci projektov obnovljivih virov in svetovalne agencije bodo imeli zanesljiv vir podatkov za strateško načrtovanje investicij v električno infrastrukturo. Nadaljnji razvoj sistema bi lahko vključeval implementacijo napredne analitike za napovedovanje trendov porabe, integracijo z dodatnimi viri podatkov ter razvoj uporabniškega vmesnika za vizualizacijo podatkov. Dolgoročno bi sistem lahko postal temelj za nacionalno platformo spremljanja energetske infrastrukture, ki bi podpirala prehod na trajnostno energetsko prihodnost.

6.2 Možnosti nadaljnjega razvoja

Trenutna implementacija predstavlja trdno osnovo za številne razširitve in izboljšave. V prihodnosti bi bilo smiselno implementirati napredne analitične funkcionalnosti, vključno s prediktivnimi modeli za napovedovanje prihodnjih kapacitet na podlagi zgodovinskih trendov. Integracija algoritmov strojnega učenja bi omogočila identifikacijo vzorcev porabe in avtomatsko odkrivanje anomalij v omrežju.

Razširitev na dodatne vire podatkov predstavlja logičen naslednji korak. Poleg drugih DNO operaterjev v Veliki Britaniji bi sistem lahko integriral podatke iz evropskih TSO (Transmission System Operators) platform. Razvoj spletnega vmesnika z interaktivnimi zemljevidi in vizualizacijami bi demokratiziral dostop do podatkov tudi netehničnim uporabnikom. Dolgoročno bi sistem lahko postal osnova za nacionalno platformo energetskega načrtovanja, ki bi z uporabo umetne inteligence optimizirala postavitev novih proizvodnih kapacitet, predlagala ojačitve omrežja ter simulirala različne scenarije energetske tranzicije.

Literatura

[1] Renzo Angles in Claudio Gutierrez. "An Introduction to Graph Data Management". V: *Graph Data Management: Fundamental Issues and Recent Developments*. Ur. George Fletcher, Jan Hidders in Josep Lluís Larriba-Pey. Cham: Springer International Publishing, 2018, str. 1–32. ISBN: 978-3-319-96193-4. DOI: 10.1007/978-3-319-96193-4_1. URL: https://doi.org/10.1007/978-3-319-96193-4_1.

[2] Kimberly A. Barchard in Larry A. Pace. "Preventing human error: The impact of data entry methods on data accuracy and statistical results". V: *Computers in Human Behavior* 27.5 (2011). 2009 Fifth International Conference on Intelligent Computing, str. 1834–1839. ISSN: 0747-5632. DOI: https://doi.org/10.1016/j.chb.2011.04.004. URL: https://www.sciencedirect.com/science/article/pii/S0747563211000707.

[3] Chandan Biswas in sod. "Solution to Web Scraping". V: *2023 11th International Conference on Internet of Everything, Microwave Engineering, Communication and Networks (IEMECON)*. 2023, str. 1–5. DOI: 10.1109/IEMECON56962.2023.10092327.

[4] Lorenzo Bravi in Alessio Ghelli. "Web Scraping for Official Statistics: a Critical Review". V: *Journal of Official Statistics* 39.1 (2023), str. 1–22. DOI: 10.2478/jos-2023-0001.

[5] Joe Celko. "Chapter 1 - Graphs, Trees, and Hierarchies". V: *Joe Celko's Trees and Hierarchies in SQL for Smarties*. Ur. Joe Celko. The Morgan Kaufmann Series in Data Management Systems. San Francisco: Morgan Kaufmann, 2004, str. 3–15. ISBN: 978-1-55860-920-4. DOI: https://doi.org/10.1016/B978-155860920-4/50002-7. URL: https://www.sciencedirect.com/science/article/pii/B9781558609204500027.

[6] Wenfei Fan, Floris GEERTS in Xibei Jia. "Improving data quality: consistency and accuracy". en. V: *Proceedings of the 33rd International Conference on Very Large Databases (VLDB)*. 2007. ACM, 2007, str. 315–326. ISBN: 78-1-59593-649-3.

[7] Google Cloud. *Google Cloud Platform Documentation*. 2025. URL: https://cloud.google.com/docs (pridobljeno 29. 6. 2025).

[8] Sinan Küfeoğlu in Michael G. Pollitt. "The impact of PVs and EVs on domestic electricity network charges: A case study from Great Britain". V: *Energy Policy* 127 (2019), str. 412–424. ISSN: 0301-4215. DOI: https://doi.org/10.1016/j.enpol.2018.12.012. URL: https://www.sciencedirect.com/science/article/pii/S0301421518308085.

[9] Ryan Mitchell. *Web Scraping with Python: Collecting More Data from the Modern Web*. 2nd. O'Reilly Media, 2018. ISBN: 978-1491985571.

[10] National Grid. *Network Capacity Map Application*. 2025. URL: https://www.nationalgrid.co.uk/our-network/network-capacity-map-application (pridobljeno 29. 6. 2025).

[11] PostgreSQL Global Development Group. *PostgreSQL Documentation*. 2025. URL: https://www.postgresql.org/docs/ (pridobljeno 29. 6. 2025).

[12] Murari Ramuka. *Data Analytics with Google Cloud Platform*. accessed 2025-11-26. BPB Publications, 2019. URL: https://books.google.si/books?id=c7lIEAAAQBAJ.

[13] Selenium Project. *Selenium WebDriver Documentation*. 2025. URL: https://www.selenium.dev/documentation/webdriver/ (pridobljeno 26. 11. 2025).

[14] Alkis Simitsis, Spiros Skiadopoulos in Panos Vassiliadis. "The History, Present, and Future of ETL Technology (invited)". V: *International Workshop on Data Warehousing and OLAP*. 2023. URL: https://api.semanticscholar.org/CorpusID:258216485.

[15] Frauke Wiese in sod. "Open Power System Data–Data platform". V: *Joule* 3.12 (2019), str. 2919–2924. DOI: 10.1016/j.joule.2019.10.006.

[16] Yihong Zhou, Chaimaa Essayeh in Thomas Morstyn. "Datasets of Great Britain primary substations integrated with household heating information". V: *Data in Brief* 54 (2024), str. 110483. ISSN: 2352-3409. DOI: https://doi.org/10.1016/j.dib.2024.110483. URL: https://www.sciencedirect.com/science/article/pii/S2352340924004529.